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

[H2020 - Grant Agreement No.101016608]

Deliverable D4.2

EVOLVED-5G Factory of the Future (FoF) NetApps

Editor	Yannis Karadimas (MAG)
Contributors	(TID), (NCSRD), (MAG), (ATOS), (INTRA), (LNV), (IMM), (GMI), (INF), (CAF), (ININ), (UMA), (ZORT), (CSIC), (8BELLS), (FOGUS), (IQBT), (PAL), (UML)
Version	1.0
Date	June 8th, 2022
Distribution	PUBLIC (PU)

DISCLAIMER

REVISION HISTORY

Revision	*Date*	*Responsible*	*Comment*
0.1	March 31st, 2022	Yannis Karadimas (MAG)	Edit ToC
0.2	April 5th, 2022	Yannis Karadimas (MAG)	Initial contributions
0.3	April 7th, 2022	Yannis Karadimas (MAG)	Revised content
0.4	May 16th, 2022	Yannis Karadimas (MAG)	Contributions incorporated
0.5	May 25th, 2022	Yannis Karadimas (MAG)	Internal review
0.6	June 3rd, 2022	Yannis Karadimas (MAG)	SC review
0.9	June 6th, 2022	Yannis Karadimas (MAG)	Final review
1.0	June 7th, 2022	Yannis Karadimas (MAG)	Final version

LIST OF AUTHORS

TID	*TELEFONICA INVESTIGACION Y DESARROLLO SA*	*J. Garcia* *D. Artuñedo*
NCSRD	*NATIONAL CENTER FOR SCIENTIFIC RESEARCH "DEMOKRITOS"*	*H. Koumaras, G. Makropoulos, D. Fragkos, A. Gogos*
ATOS	*ATOS IT SOLUTIONS AND SERVICES IBERIA SL*	*R. Marco* *P. Encinar*
INTRA	*NETCOMPANY INTRASOFT SA*	*A. Dimitriou*
MAG	*MAGGIOLI SPA*	*Y. Karadimas*
LNV	*Lenovo (Deutschland) GmbH*	*A. Salkintzis,* *D. Dimopoulos*
IMM	*IMMERSION*	*C. Bailly*
GMI	*GMI AERO*	*G. Kanterakis* *M.O. Sauer*
INF	*INFOLYSIS P.C.*	*T. Dounia* *A. Varkas,* *C. Sakkas,* *G. Theodoropoulos*
CAF	*CAFA TECH OU*	*T. Järvet* *M.Rannu* *M. Hiiemaa*
ININ	*INTERNET INSTITUTE, COMMUNICATIONS SOLUTIONS AND CONSULTING LTD*	*L. Koršič,* *J. Cijan* *J. Sterle* *R. Susnik*
ZORTENET	*ZORTENET P.C.*	*A. Kourtis, A. Oikonomakis,* *G. Xilouris*
UMA	*UNIVERSIDAD DE MALAGA*	*B.Garcia, R. Lopez, J. Canca*
CSIC	*AGENCIA ESTATAL CONSEJO SUPERIOR DE INVESTIGACIONES CIENTIFICAS (IHSM CSIC)*	*R. Fernandez* *J. Losada*
8BELLS	*EIGHT BELLS LTD*	*I.Margaritis*
FOGUS	*FOGUS INNOVATIONS & SERVICES P.C.*	*D. Tsolkas,* *S. Charismiadis*
IQBT	*INQBIT INNOVATIONS SRL*	*J. Stylianou*
PAL	*PAL ROBOTICS SL*	*T. Peyrucain*
UML	*UNMANNED SYSTEMS LIMITED*	*D. Cupello* *A. Marino*

GLOSSARY

Abbreviations/Acronym	*Description*
AI	*Artificial Intelligence*
ACK	*Acknowledgement*
API	*Application Programming Interface*
AR	*Augmented Reality*
CAPIF	*Common API Framework*
CI/CD	*Continuous Integration / Continuous Development*
CLI	*Command Line Interface*
CSS	*Cascading Style Sheets*
CV	*Computer Vision*
DSCP	*Differentiated Services Code Point*
DoC	*Degree of Curing*
EC	European Commission
FDR	Frequency Domain Reflectometry
FoF	Factories of the Future
gNodeB / gNB	*Next Generation (5G) Base Station*
GUI	*Graphical User Interface*
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transfer Protocol*
IDE	*Integrated Development Environment*
IoT	*Internet of Things*
IIoT	*Industrial Internet of Things*
IP	*Internet Protocol*
JSON	*JavaScript Object Notation*
KPI	*Key Performance Indicator*
M2M	*Machine to Machine*
NEF	*Network Exposure Function*
NetApp	*Network Application*
NIC	*Network Interface Card*
NPN	*Non-Public Network*
OAuth	*Open Authorization*
OVS	*Open Virtual Switch*
PHP	*Hypertext Pre-processor*
PPE	*Personal Protective Equipment*
QoS	*Quality of Service*
REST	*Representational State Transfer*
ROS	*Robot Operating System*

RTPS	*Real Time Publish Subscribe*
SDK	*Software Development Kit*
SLA	*Service Level Agreement*
SLS	*Service Level Specification*
SME	*Small and Medium-sized Enterprises*
SQL	*Structured Query Language*
TCP	*Transmission Control Protocol*
UE	*User Equipment*
UI	*User Interface*
ULCCP	*Unmanned Life Central Control Platform*
vAPP	*Vertical Application*
VCL	*Visual Components Library*
VPD	*Vapour Pressure Deficit*
WiFi	*Wireless Fidelity*

EXECUTIVE SUMMARY

EVOLVED-5G responds to the *5G PPP ICT-41-2020 5G innovations for verticals with third party services* call, whose main goal is to deliver enhanced experimentation facilities on top of which third party experimenters (e.g., SMEs or any service provider and target vertical users) will have the opportunity to test their applications.
The EVOLVED-5G project realises this vision by encouraging the creation of a NetApp ecosystem revolving around a 5G facility which will provide the tools and processes for the development, verification, validation and certification of NetApps as well as their validation on top of actual 5G network infrastructures, and mechanisms for market releasing.
This document describes the initial release of NetApps based on the early prototypes provided by SMEs partners for the four Smart manufacturing pillars (Industry 4.0)

The main purpose of this deliverable is summarised in the following:

- To provide details on the implementation of the NetApps developed and presented as early prototypes by presenting the relevant tools that facilitate the development along with their functionalities.
- To introduce detailed presentation and analysis of the initial versions of the NetApps by explaining the architecture.
- To describe the advantages introduced using 5G technology in each of the use cases

The work presented in this deliverable will guide the project towards the next versions of the NetApps, since it provides an initial implementation, design and realization methodology that each NetApp must follow to be completed either as standalone or as part of a vertical application (vApp).

TABLE OF CONTENTS

1 INTRODUCTION AND TARGET AUDIENCE 1
1.1 Purpose of the document 1
1.2 Structure of the document 1
1.3 Target Audience 1
2 NETAPP IMPLEMENTATION FOR INDUSTRY 4.0 PILLARS 3
2.1 Introduction 3
2.2 Development process and tools framework 3
2.3 Verification process 4
3 Release A Of The NetApps And Use Cases 6
3.1 Pillar 1: Interaction of Employees and Machines (IEM) 6
3.1.1 Remote assistance in AR NetApp 6
3.1.2 Chatbot assistant NetApp 8
3.1.3 Digital/physical twin NetApp 11
3.2 Pillar 2: Efficiency in FoF Operations (FoF) 15
3.2.1 CAFA-NetMapper NetApp 15
3.2.2 Smart irrigation 5G Agriculture NetApp 18
3.2.3 Industrial grade 5G connectivity with assured QoS and integrated SLA/SLS monitoring capabilities NetApp 21
3.2.4 Anomaly Detection NetApp 24
3.3 Pillar 3: Security guarantees and risk analysis 28
3.3.1 Traffic Management NetApp 28
3.3.2 ID Management and Access Control NetApp 31
3.3.3 5G SIEM NetApp 33
3.4 Pillar 4: Agility in the Production Line Infrastructure 36
3.4.1 5G Teleoperation NetApp 36
3.4.2 Localisation NetApp 40
4 Conclusion and Next steps for NetApp development 46
5 References 47

LIST OF FIGURES

Figure 1. Technical architecture of the IMM's NetApp 7
Figure 2. Screenshots of the IMM NetApp 7
Figure 3. Chatbot use case architecture 9
Figure 4. Chatbot Assistant frontend demo 10
Figure 5. NetApp source code 10
Figure 6. FastAPI server 11
Figure 7. Frontend interface 11
Figure 8. Digital/physical twin NetApp architecture 14
Figure 9 Geographic retrieval and display notifications 14
Figure 10 Retrieving and displaying QoS 15
Figure 11. CAFA's NetApp architecture 16
Figure 12 NetApp creates a subscription and notifies Flask server about QoS change 17
Figure 13 Flask server receives the QoS change message and sends it to the vApp 17
Figure 14. Smart Irrigation use case architecture 19
Figure 15. IDE and section of code of the UMA-CSIC NetApp 20
Figure 16. Dockerfile of the UMA-CSIC NetApp 20
Figure 17: Integration of the NetApp in the "Industrial grade 5G connectivity with assured QoS and integrated SLA/SLS monitoring capabilities" environment 22
Figure 18: NetApp operation workflow example 23
Figure 19: NetApp "env" sample 23
Figure 20: Overall Communication between components 24
Figure 21: Map of NEF form the vApp Dashboard 25
Figure 22: Example of Location subscription for specific UE 26
Figure 23: Sample of received logs 26
Figure 24: Example of policy applied denoting the permitted cells 27
Figure 25: Example of alerts as shown from vApp Dashboard 27
Figure 26: NetApp – vApp (L7 Switch) schematic 29
Figure 27: NEF emulator – QOS notification received 30
Figure 28:   vApp configuration 30
Figure 29:   ID Management and Access Control NetApp Architecture 32
Figure 30 Workflow describing the testing scenario 33
Figure 31. Fogus NetApp architecture 34
Figure 32. NetApp deployed with docker-compose 34
Figure 33. NetApp logs on running phase 35
Figure 34. Create one-time monitoring subscription from NetApp 35
Figure 35. Periodic call-backs from Monitoring Event API appearing on NetApp 36
Figure 36. Teleoperation use case architecture 37
Figure 37. Teleoperation NetApp interactions 38
Figure 38 functions to create a valid request to the NEF Emulator1 39
Figure 39 Functions to create a valid request to the NEF Emulator2 39
Figure 40 Python code that uses the EvolvedAPI helpers and endpoint 40
Figure 41. Localization use case architecture 41
Figure 42 Request to the cell ID endpoint 42
Figure 43 Instantiate different classes to monitor Location information, events 43
Figure 44 Creation of endpoints using FastAPI to monitor a call-back 43
Figure 45 Code for cell_node.py 44

Figure 46 destroy the node and to publish the cell ID in RTPS 44
Figure 47 Code for the main.py.. 45

1 INTRODUCTION AND TARGET AUDIENCE

1.1 Purpose of the document

One of the main objectives of EVOLVED-5G project is the design, development and testing of the NetApps, which fall under the four Industry 4.0 pillars.

The current document "EVOLVED-5G Factory of the Future (FoF) NetApps" provides details on the development process and tools used to implement all the different use cases for each of the Industry 4.0 pillars. For each NetApp, a brief summary of the related use-case is provided first, followed by a description of the "pain" that this use case is trying to address and also the "gain" that the NetApp contributes. Then, details of the NetApp implementation are given.

1.2 Structure of the document

- Section 1 "Introduction and Target Audience"
- Section 2 "NetApp Implementation for Industry 4.0 Pillars" describes in detail the development process and tools used to implement all the different NetApps addressing the use cases for each of the Industry 4.0 pillars.
- Section 3 "Conclusion and Next steps for NetApp development" presents the next steps for the development of the NetApps.

1.3 Target Audience

The release of the deliverable is public, intending to describe in detail the overall EVOLVED-5G ecosystem and NetApps Lifecycle design to a wide variety of research individuals and communities.

Different target audiences for D4.2 are identified as detailed below:

- **Project Consortium**: To validate that all objectives and proposed technological advancements have been analysed and to ensure that, through the proposed NetApp Lifecycle phases and the various Environments, further work can be concretely derived. Furthermore, the deliverable sets to establish a common understanding among the consortium with regards to:
    - The actual development of the NetApps as categorized in four different Industry 4.0 domains including the process and the tools used.
- **Industry 4.0 and FoF (factories of the future) vertical groups:** To crystallise a common understanding of technologies, and tools that were used for the development of the NetApps. A non-exhaustive list of Industry 4.0-related groups is as follows:
    - Manufacturing industries (including both large and SMEs) and IIoT (Industrial Internet of Things) technology providers.
    - European, national, and regional manufacturing initiatives, including funding programs, 5G-related research projects, public bodies and policy makers.
    - Technology transfer organizations and market-uptake experts, researchers, and individuals.
    - Standardisation Bodies and Open-Source Communities.
    - Industry 4.0 professionals and researchers with technical knowledge and expertise, who have an industrial professional background and work on industry 4.0-related areas.
    - Industry 4.0 Investors and business angels.

- **Other vertical industries and groups**: To seek impact on other 5G-enabled vertical industries and groups in the long run. Indeed, all the architectural components of the facility are designed to secure interoperability beyond vendor specific implementation and across multiple domains. The same categorization as the above but beyond Industry 4.0 can be of application.
- **The scientific audience, general public and the funding EC Organisation:** To document the work performed and justify the effort reported for the relevant activities. The scientific audience can also get an insight of the NetApps' development process and tools.

2 NETAPP IMPLEMENTATION FOR INDUSTRY 4.0 PILLARS

2.1 INTRODUCTION

The EVOLVED-5G ecosystem as described in D3.1 [1] has been designed to provide the necessary tools, processes and functionalities for the development and software testing (i.e., verification, validation and certification) of the NetApps. The EVOLVED-5G ecosystem is organized around the concept of different environments, which are Tier-1/Core architectural components of the ecosystem and are specifically suited for supporting each one of the different phases of the NetApp lifecycle, along with other functional blocks and tools that provide the expected functionalities and processes needed for the NetApp implementation.
As properly described in D2.1 [2] and D3.1 [1], the first environment in the lifecycle of the NetApp is the Workspace, which the developer uses in order to develop the NetApp locally, making use a collection of tools provided by EVOLVED-5G in order to facilitate someone in this process and then to verify with a list of tests that the specifications of the NetApp are correctly implemented. In respect to the implementation of the NetApps, the workflow of the development process refers to the request of the vApp to the NetApp which in turn is translated to one or more requests of the NetApp to the 5G API Core. The reply that comes from the 5GC includes the data that have been requested and are in turn forwarded to the vApp.

The tools provided in the EVOLVED-5G workspace environment for the development of a NetApp are the following:

- SDK tools, it has been implemented a template, libraries for the 5G APIs and a Command Line (CLI) tool to create the NetApp based on such template and run the pipelines from the CI/CD.
- The CI/CD is based on a Jenkins solution, the implementation carries out pipelines such as build, deploy or destroy for NetApps.
- GitHub repository, a version control repository where actually the code of NetApps, instructions and tools such NEF and CAPIF are stored.
- Open Repository, after successfully pass the different phases of its lifecycle the NetApp binaries are stored, specific folders have been created to store them.

The NetApps that are being developed under the scope of EVOLVED-5G are based on a container image, offering Representational State Transfer (REST) Access Point Interface (API) endpoints (a.k.a. Business APIs) to the vApp as well as to the 5G system for subscription-based events when applicable.

2.2 DEVELOPMENT PROCESS AND TOOLS FRAMEWORK

The NetApp development is a process in which the NetApp developer will make use of the SDK tools provided in EVOLVED-5G towards the creation of a NetApp. The SDK tool is offered to the developer in a straightforward way as a package installation, therefore the SDK CLI is ready to be use which, is the first step for the developer to start the development process for a NetApp. The recommended programming language within the EVOLVED-5G ecosystem is Python, based on a requirement collection process by the participating SMEs on which language is more convenient for them to build the NetApps. Of course, a NetApp developer can use any programming language, the whole support material for the developers in EVOLVED-5G is focused on Python.

Henceforth, the developer can start programming the functionality of its NetApp. To facilitate the implementation of 5G connectivity functionalities, a Network Exposure Function (NEF) emulator has been implemented within EVOLVED-5G to test the 5G Core connectivity. In addition, the SDK CLI provides QoS Awareness and Location Subscription libraries related to NEF to help the developer accomplish the communication between the NetApp and the 5G Core.

For the use of the SDK the developers can find installation and usage instructions [3], more details regarding functionality and how the SDK has been implemented are included in deliverables D3.1 [1] and D4.1 [4]. The SDK tools [5] are updated continuously and therefore improvements are performed regularly, the main upgrades impacting the benefit of the developers are described below:

- Automatization of the SDK package creation. Anytime a new improvement is merged in the main branch, a new python package is created and uploaded to Python Package Index (Pypi), a software repository for Python programming language. In such way, the developer only needs to update or (re)install the SDK software package in order to use the new implemented updates.
- Errors catching. When launching the pipelines (build, deploy, destroy) the developer must provide some parameters, if by mistake (or misspelled) some of the parameters are wrong (i.e., the repository name doesn't exist), the SDK will catch the error(s) and stops launching the pipeline rather than, continuing with the process of launching the pipeline and causing a failure in the pipeline, therefore saving some time to developers, as well as some overload to the CI/CD.
- SDK NEF libraries update. Location Subscriber SDK class has been extended with the ability to retrieve location information for a specific Cell Id (Cell Identification). Before this release the developer would get a notification if a user device changes its connectivity, i.e., if it connects to a different cell. This notification contains only the Cell Id number that the device connected to. With the new version the NetApp developer can additionally retrieve the coordinates (latitude, longitude) for this cell.

2.3 Verification Process

Once the development of the NetApp is completed, the next phase within the Workspace is the verification stage with the goal of verifying the following properties of the NetApp: (i) code quality through static code analysis, (ii) NetApp containerization implementation, and (iii) communication with the NEF emulator [6] and the CAPIF core function [7]. The verification process is the target of task T3.1, and some details about the process can be found in the deliverable D3.1 [1]. Progress has already been made in this respect, since the pipelines can be executed to verify if the NetApp can be built, deployed and destroyed in a virtualized environment (without considering the vApp). More verification tests are already being implemented and, will be reflected in the corresponding deliverables of WP5 and specifically D5.2 that is planned to be submitted in M20. For self-containment, a rough description of the process follows.

Taking advantage of the appropriate tools, static code analysis is going to be performed using SonarQube [8], scanning for vulnerabilities in containers is going to be performed using Trivy [9]. Also, unitary tests to verify the communication with 5G Core APIs are going to be available to NetApp developers through the SDK using Robot Framework [10]. For facilitating the communication of each developed NetApp with the 5G APIs emulators (i.e., NEF and CAPIF) a dummy NetApp has been developed, which is planned to serve as a wrapper of communication between each NetApp and the 5G infrastructure. For the dummy NetApp, exhaustive tests for each NEF and CAPIF API calls are currently being implemented using Robot Framework. Some of tests are planned to be implemented in the form of pipelines in the CI/CD infrastructure of the EVOLVED-5G project and are currently being integrated into the SDK provided to the NetApp developers. This way, a common testing process is laid out for all NetApps to enable the verification of their fundamental functional properties, which are not mandatory to let them pass to the validation phase.

3 Release A Of The NetApps And Use Cases

This section describes the development of Release A for the NetApps, utilising the SDK and following up the preliminary version described in D4.1 [4], within the four pillars in the EVOLVED-5G context: Interaction of Employees and Machines (IEM), Efficiency in Factory of the Future operations (FoF), Security Guarantees and risk Analysis (SEC), as well as Production Line Infrastructure (PLI). The description of the development work performed for each NetApp is composed of the following subsections: a brief description of the use case by which the NetApp is being utilized, the current limitations of the use cases, the added value of the 5G and the APIs provision as a consequence, the detailed architecture as well as the tools that were exploited by the developers during the development process.

3.1 Pillar 1: Interaction of Employees and Machines (IEM)

3.1.1 Remote assistance in AR NetApp

3.1.1.1 Use case short description

As described in deliverables D2.1 [2] and D4.1 [4] , IMM's use case is based on remote assistance in Augmented Reality (AR) for industrial tasks. One the one hand, The AR aspects are handled by IMM's vApp. On the other hand, IMM's NetApp is dedicated to:

- Secure and time-sensitive communications through the 5G network
- Autonomous adaptation to network performance and user needs

3.1.1.2 Current limitations of the use case

Augmented Reality (AR) is a promising technology for Industry 4.0 but has strong requirements in terms of performance. Beyond the raw computational power to display virtual objects superimposed on the environment, network performance is also a key aspect for synchronous remote collaboration in AR. There is a strong need in terms of low and reliable latency and significant bandwidth adapted to video streaming and complex 3D object sharing. More and more AR devices can use Wi-Fi networks to communicate with each other, but these networks may not be available in industrial or outdoor contexts.

3.1.1.3 Added value of 5G

5G has the potential to offer these required network performances. Time-sensitive networking mechanisms can ensure a smooth real-time collaboration for end users. Besides the raw performances, 5G also gives access to tools to monitor the state of the network. This is a crucial feature to be able to notify end-users as soon as possible when a network issue occurs. The goal is to trigger the most appropriate adaptations so that end-users can continue their work without being too disturbed, which could otherwise be hazardous in the industrial contexts we target.

3.1.1.4 Detailed architecture

The IMM NetApp is a Python (version 3.10) application communicating with the Augmented Reality vApp (Unity, C#) and the 5G NEF emulator. All three applications are running locally on the same MS Windows 10 computer but the vApp can also be executed on an AR Head-Mounted Display (the Hololens 2) on the same Wi-Fi network.

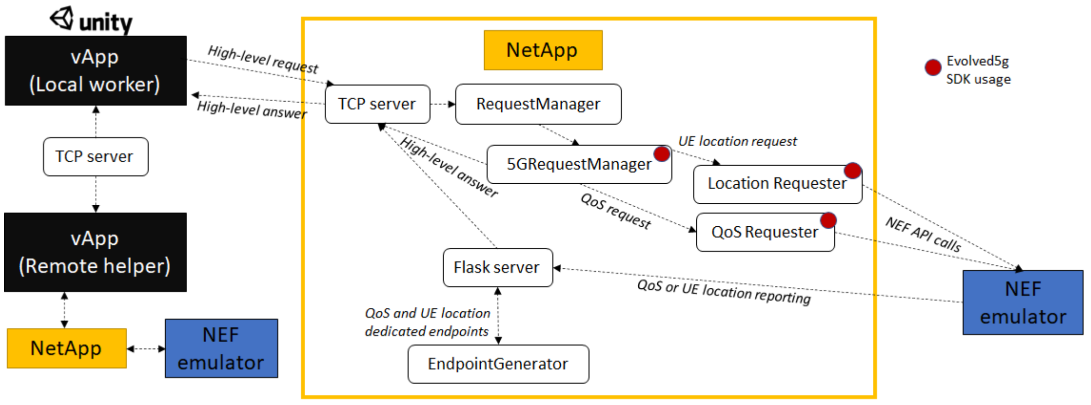
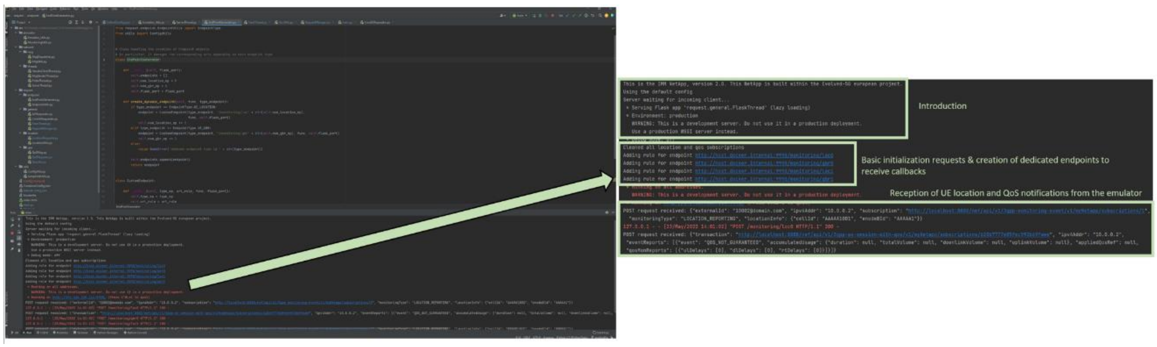


Figure 1. Technical architecture of the IMM's NetApp

Figure 1 gives an overview of the components comprising the NetApp. The communications between the vApp and the NetApp are performed through a dedicated Transmission Control Protocol (TCP) client-server architecture. High-level requests from the vApp are given to the RequestManager. This component will dispatch the request to the 5GCoreRequester and send a first acknowledgment (ACK) notification to the vApp. The 5GCoreRequester handles the general requests and communications with the NEF emulator like token-based authentication. It also delegates the creation of NEF API calls to dedicated Requester (1 per API and more specifically, LocationRequester for *MonitoringEvent* API and QoSRequester for *AsSessionWithQoS* API). The Evolved5g SDK is used to create NEF API calls. When callbacks are needed, Requesters use a Flask server running in a dedicated thread and custom endpoints through the EndpointGenerator. Upon receiving a direct answer or an endpoint notification from the emulator, the NetApp creates the corresponding high-level message and transmits it to the vApp.

So far, the NetApp is mainly used to facilitate the interaction with the NEF emulator by sending high-level requests and receiving high-level answers. The next step with future functionalities in the SDK and emulator is to add an AdaptationManager component linked to the RequestManager in order to analyse further notifications received by the Flask server and propose adaptations to the vApp. This will increase the added value of the NetApp by proposing autonomously QoS compromises and triggering the suggested adaptation mechanisms on the vApp side.

Finally, there can be two instances of vApp+NetApp running at the same time: one for the local worker within the factory, the other one for the remote helper. Upon launching the vApp, the user chooses one profile or the other and communications are then handled by a TCP client/server.



```
This is the IMM NetApp, version 2.0. This NetApp is built within the Evolved-5G european project.
Using the default config
Server waiting for incoming client...
 * Serving Flask app 'request.general.FlaskThread' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
```

Introduction

```
Cleaned all location and qos subscriptions
Adding rule for endpoint [illegible]
Adding rule for endpoint [illegible]
Adding rule for endpoint [illegible]
Adding rule for endpoint [illegible]
```

Basic initialization requests & creation of dedicated endpoints to receive callbacks

```
   WARNING: This is a development server. Do not use it in a production deployment.
```

Reception of UE location and QoS notifications from the emulator

```
POST request received: {"externalId": "10002@domain.com", "ipv4Addr": "10.0.0.2", "subscription": "[illegible]",
 "monitoringType": "LOCATION_REPORTING", "locationInfo": {"cellId": "AAAAA1001", "enodeBId": "AAAAA1"}}
127.0.0.1 - - [23/May/2022 14:01:02] "POST /monitoring/loc0 HTTP/1.1" 200 -
POST request received: {"transaction": "[illegible]", "ipv4Addr": "10.0.0.2",
 "eventReports": [{"event": "QOS_NOT_GUARANTEED", "accumulatedUsage": {"duration": null, "totalVolume": null, "downlinkVolume": null, "uplinkVolume": null}, "appliedQosRef": null,
 "qosMonReports": [{"ulDelays": [0], "dlDelays": [0], "rtDelays": [0]}]}]}
```

Figure 2. Screenshots of the IMM NetApp

3.1.1.5 NetApp's Development tools

Based on the development process defined by the project, the IMM NetApp uses the Evolved-5G SDK to create requests for the NEF emulator. The NEF emulator is running through a Docker container and communicates with a Flask server. In addition, a containerized version of the NetApp (again with Docker) is available to facilitate future CI/CD integration efforts. For the development of the NetApp, the PyCharm IDE [11] as well as the Jsonpickle library [12] have been used so as to handle the (de)serialisation of Json objects. In the light of the above, the code of the NetApp can be seen in the following repository https://github.com/EVOLVED-5G/ImmersionNetApp

3.1.2 Chatbot assistant NetApp

3.1.2.1 Use case short description

Deliverables D2.1 and D4.1 have introduced, in a high-level description, the use case scenario that is to be realised by INF for the EVOLVED-5G project. More specifically, the scenario targets the chatbot-assisted maintenance procedures that are being performed in the premises of a factory environment. Mainly, a chatbot assistant vertical application coupled with the respective non-standalone chatbot assistant NetApp will be developed to support the factory workers whenever they encounter a faulty machinery equipment by providing both guidelines on reporting the problem and documentation for the necessary repair, while at the same time ensuring the workers' safety.

3.1.2.2 Current limitations of the use case

For the realization of the identified use case scenario, as described in D4.1, it is necessary that the chatbot can access the indoor factory location of the workers. The large number of technical equipment distributed among the wide area of a factory can trouble the online localization of the desirable equipment and prevent procedures from being done efficiently. Instead of manually inserting a unique machine identifier, the procedure can be automated by narrowing the list of the available machines down to the ones in close proximity to the authorised maintenance worker. In that way, the workers can easily search the machine that they are interested in, fetch on their mobile device all the related documentation, and finally perform any necessary action with the support of just the chatbot device. Consequently, the indoor localization of a worker with a certain level of accuracy, as required by the INF chatbot, cannot be efficiently achieved using just existing 4G access or other positioning technologies, such as GPS.

3.1.2.3 Added value of 5G

5G openness provides the location information of the cell that a UE is connected, which can be used directly by the service in order to gain location awareness of the workers. Indeed, a locally installed 5G network has the potential to offer such indoor localization information of each connected UE based on the NEF API. More specifically, for the chatbot use case, the factory will be mapped into cells according to the coverage of the installed 5G small cell antennas. The respective database will indicate the machines that fall under each of the factory areas providing the functionality of a specific machines' list.

3.1.2.4 Detailed architecture

The general architecture of the interaction between the components of the identified chatbot use case reflecting the developments made so far, can be seen in Figure 3.

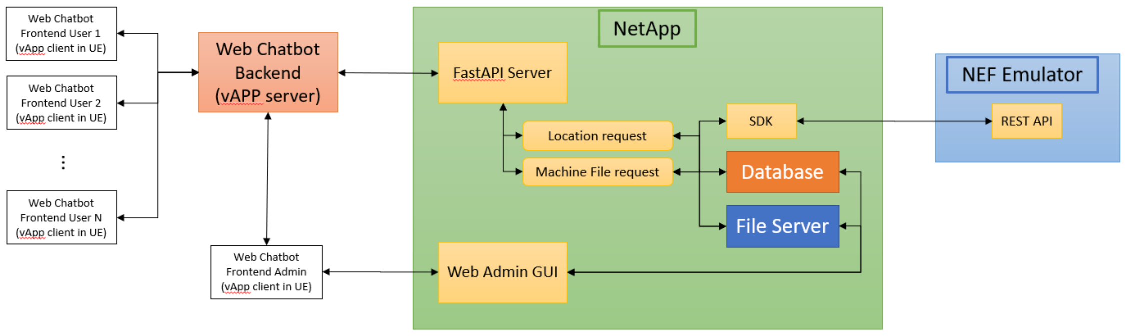

Web Chatbot Frontend User 1 (vApp client in UE)
Web Chatbot Frontend User 2 (vApp client in UE)
Web Chatbot Frontend User N (vApp client in UE)
Web Chatbot Backend (vAPP server)
Web Chatbot Frontend Admin (vApp client in UE)
NetApp
FastAPI Server
Location request
Machine File request
SDK
Database
File Server
Web Admin GUI
NEF Emulator
REST API

Figure 3. Chatbot use case architecture

More specifically, the NetApp architecture includes the following elements:

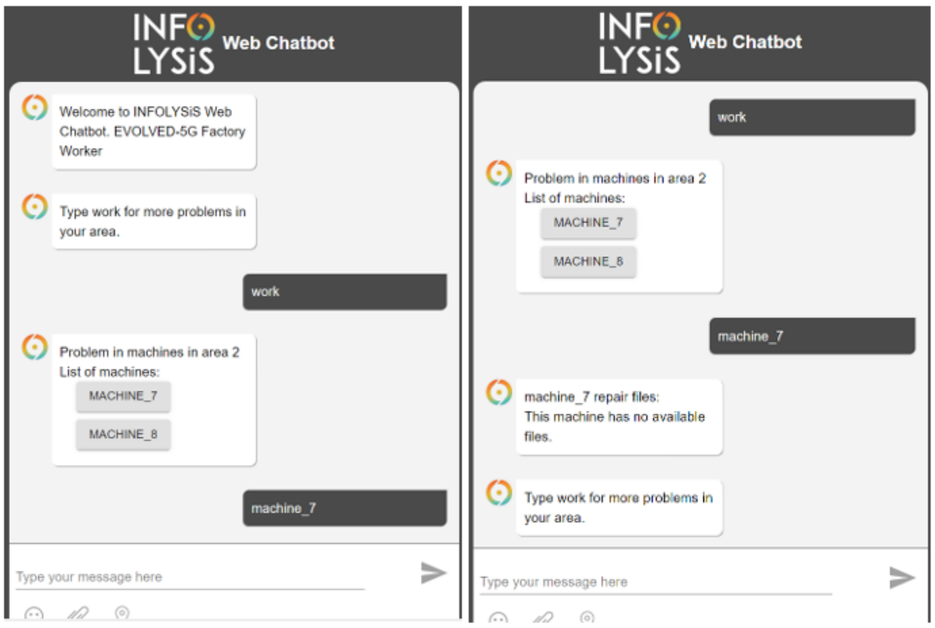
- The FastAPI Server which exposes the APIs that are being used for the interaction between the vApp and the NetApp.
- Location request and machine file request which according to the API call will return the user location or the necessary machine files.
- The SDK library which is used for the communication with the NEF Emulator.
- The Database and the File Server that hold administrative information necessary for the realization of the use case scenario.
- The Web Admin GUI that interacts with the database and the file server and serves as the user interface for handling the administrative information.

Considering that the user is logged in the chatbot application (the authorization procedure will be defined on the next version of the NetApp towards the integration of more functionalities from the general use case), the procedure is initiated from the chatbot end user by typing on the word “work”. Consequently, the following sequence takes place:

- The command is transmitted to the Web Chatbot Backend.
- The Web Chatbot Backend calls the endpoint of the NetApp (http://185.184.71.39:8000/AreaMachines/) sharing the appropriate worker id.
- The NetApp, through the aforementioned endpoint, calls the 5Gemulator through the SDK and retrieves the cell id according to the worker’s indoor location.
- Using the retrieved cell id, the NetApp accesses the database for the retrieval of the machines located under the respective cell id, flagged as malfunctioning, and returns that list to the vApp. The list is displayed on the chatbot screen as buttons, available for the end user to choose from.
- Afterwards, the user chooses the desirable machine, by pressing the respective button linked to this machine, and this command, the chosen machine id, is transmitted to the Web Chatbot Backend.
- Then, the Web Chatbot Backend calls the endpoint of the NetApp (http://185.184.71.39:8000/MachineFiles/) sharing the appropriate machine id.
- According to the machine id, the NetApp accesses the database, retrieving the files related to the specific machine, and returns them as a list to the vApp. The list is is displayed on the chatbot screen as buttons according to the list of machines mentioned before.
- Finally, the user can view each of the files by pressing on the linked Button.

3.1.2.5 NetApp's Development tools

Currently, the developments for the chatbot assistant support a use case where the logged-in user can choose from the machines in proximity and, in turn, view all the related files. More specifically, Figure 4 illustrates the current frontend of the chatbot assistant along with the provided functionalities. As shown in Figure 5 the NetApp is developed in Python and the exposed APIs use the FastAPI server as depicted in Figure 6while the versioning of the NetApp is being performed using GitHub. As indicated by the architecture, the NetApp also utilizes a database developed in MySQL and a file server that uses Apache. For the interaction between the database and the file server a frontend as illustrated in Figure 7, is also being developed using Hypertext Preprocessor (PHP), Cascading Style Sheets (CSS), and HyperText Markup Language (HTML). The requests from the NetApp to the 5G Emulator are being done using the SDK library. Additionally, both the NetApp and the 5G emulator are now running through a Docker container. This containerized NetApp version has already been tested, during the work of WP5, for the facilitation of future CI/CD integration efforts. The code of the NetApp can be seen in the following repository: GitHub repo link: https://github.com/EVOLVED-5G/InfolysisNetApp.git



Welcome to INFOLYSiS Web Chatbot. EVOLVED-5G Factory Worker

Type work for more problems in your area.

work

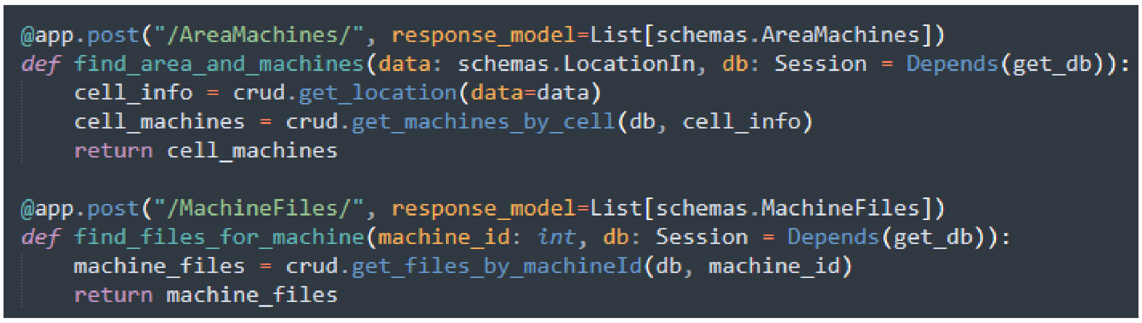
Problem in machines in area 2
List of machines:
MACHINE_7
MACHINE_8

machine_7

machine_7 repair files:
This machine has no available files.

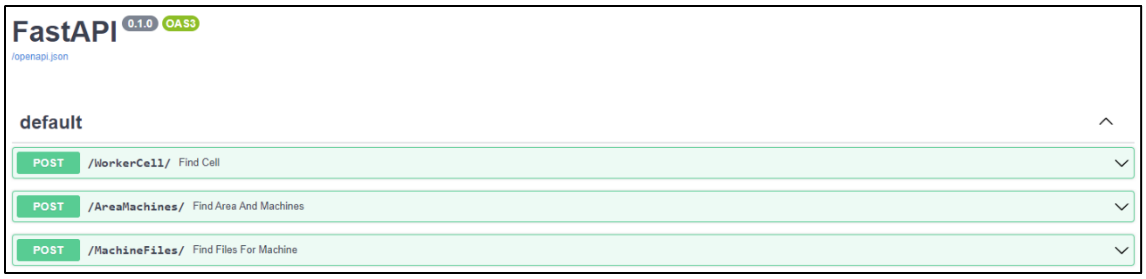
Type work for more problems in your area.

Figure 4. Chatbot Assistant frontend demo



```
@app.post("/AreaMachines/", response_model=List[schemas.AreaMachines])
def find_area_and_machines(data: schemas.LocationIn, db: Session = Depends(get_db)):
    cell_info = crud.get_location(data=data)
    cell_machines = crud.get_machines_by_cell(db, cell_info)
    return cell_machines

@app.post("/MachineFiles/", response_model=List[schemas.MachineFiles])
def find_files_for_machine(machine_id: int, db: Session = Depends(get_db)):
    machine_files = crud.get_files_by_machineId(db, machine_id)
    return machine_files
```

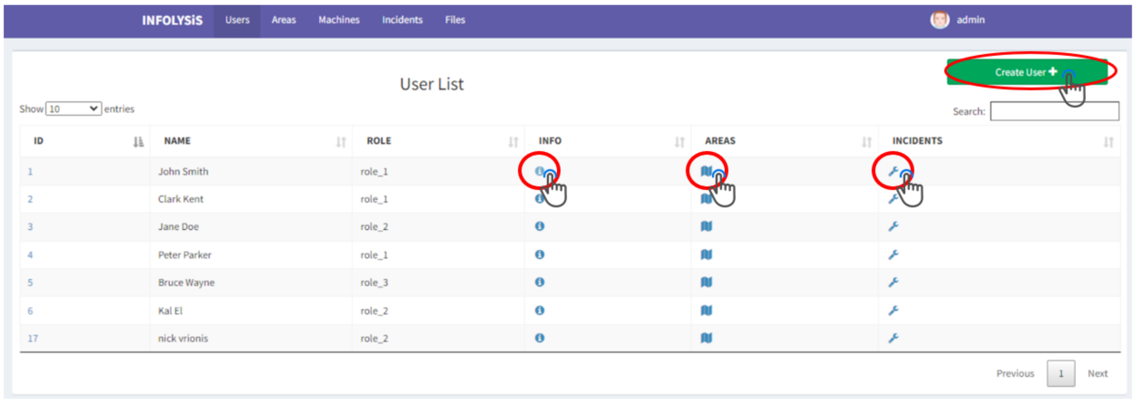
Figure 5. NetApp source code


FastAPI 0.1.0 OAS3
/openapi.json

default

POST	/WorkerCell/	Find Cell
POST	/AreaMachines/	Find Area And Machines
POST	/MachineFiles/	Find Files For Machine

Figure 6. FastAPI server


INFOLYSiS Users Areas Machines Incidents Files admin

User List

Create User +

Show 10 entries

Search:

ID	NAME	ROLE	INFO	AREAS	INCIDENTS
1	John Smith	role_1	ℹ	🗺	🔧
2	Clark Kent	role_1	ℹ	🗺	🔧
3	Jane Doe	role_2	ℹ	🗺	🔧
4	Peter Parker	role_1	ℹ	🗺	🔧
5	Bruce Wayne	role_3	ℹ	🗺	🔧
6	Kal El	role_2	ℹ	🗺	🔧
17	nick vrionis	role_2	ℹ	🗺	🔧

Previous 1 Next

Figure 7. Frontend interface

The vApp interacts with the NetApp through the exposed APIs and thus the only customization needed for the integration is to configure the chatbot application to properly apply the information provided by the NetApp. The vApp is developed using REACT for the frontend and botman PHP [13] for the backend.

3.1.3 Digital/physical twin NetApp

3.1.3.1 Use case short description

When a bonded composite repair is performed, all repair data (temperature, humidity, vacuum level etc.) are recorded, in order to certify that the overall process has been performed according to specifications and confirm the physical and mechanical properties of the repaired part.
Within the frame of EVOLVED-5G the ANITA hot bonder(s) [14] used for repair curing will be connected to the 5G network at the repair area, in order to transmit in real-time all related data to the Engineering Centre of aircraft manufacturer / airline / MRO creating a Digital / Physical Twins of the actual repair. This will help in optimizing the integration of systems in the airframe along with the validation of important structural advances and to make progress on the production efficiency and manufacturing of structures. Solutions will assist in avoiding part scraping during manufacturing, as well as in MROs, airlines and composite plants, by increasing the range of application of bonded composite repairs. In addition, application of Physical / Digital Repair Twins is expected to directly reduce structural composite repair certification costs and increase aircraft availability, while assisting in continuous airworthiness of commercial aircraft fleet.

3.1.3.2 Current limitations of the use case

Current procedures in bonded composite repairs of aircraft structures dictate certification "by similarity", meaning the need to prove that the curing cycle to which the aircraft parts' have been submitted in terms of Temperature, Time, Vacuum and related environmental conditions is fully and everywhere within the specifications and the related tolerances provided by the aircraft manufacturer. While this may be relatively easy for simple repairs, it becomes extremely difficult and challenging when large, geometrically complex and / or remotely performed repairs are concerned, which is the case that the EVOLVED-5G tries to address through the development of the Digital / Physical Twin related applications.

Today, when a bonded composite repair is remotely performed, all repair data (temperature, humidity, vacuum level etc.) have to be recorded and transmitted to the aircraft manufacturer and the aeronautical authorities AFTER the performance of the repair. When available to the competent engineers, a post-repair evaluation will be performed so as to confirm whether the specifications and tolerances are met. Should the slightest deviation occur, as several repairs take place “on-wing” and remotely (maybe even outside of hangars) at challenging environmental conditions, due to geographical location (extremely low temperature, increased humidity, very high altitude etc.), then the repair would have to be discarded, removed and performed again. This induces severe financial consequences to the airlines, not only because of the maintenance cost to re-apply the repair but, mainly, due to the unavailability of the aircraft, the grounding of which costs several hundreds of thousands of Euros per day, not to mention the damage related to the image of the company because of induced delays and cancellation of flights.

The 5G added value for Digital / Physical Twin development within EVOLVED-5G is enabling full connectivity of ANITA hot bonder(s) used for repair curing. Thanks to the 5G network at the repair area real-time data will be transmitted to the Engineering Centre of aircraft manufacturer / airline / MRO certification authorities (EASA, FAA etc.) This data will be used either to create in real-time a “Replica” repair using a second bonding console, identical to the “Source” repair (Physical-Twin) or to use such data for calculation of the DoC applying corresponding material curing equations (Digital Twin).

Overall, the 5G network will be assisting in the reinforcement of the competitiveness and the performance of EU transport manufacturing industries and related services, facilitating the development of next generation of transport means, further exploiting the advantages of light composite structures, while enabling new manufacturing and maintenance techniques for both existing and new composite structures, in order to retain areas of EU leadership in the transport sector. GMI being part of this ecosystem will be directly positively affected by these global advancements and innovations.

3.1.3.3 Added value of 5G

There are three main benefits gained through access to 5G capabilities for the development of Digital / Physical composite repair twins:

a. The application entitled” Check communications between end-users' devices”. The current app refers to the scenario where both end-users, on the primary Anita and on the secondary Anita, need to initialize secured communication necessary for the realization of the twin repair. The expected outcomes involve the checking that a satisfying QoS can be reached and the Security (Authorize users). Concerning the Current Status, the Anita HMI will be adapted with the authentication process and the communication settings. Additionally, Anita cloud system and database are already deployed and in the context of EVOLVED-5G Innovations, the goal is to enable sufficient quality of service to be achieved through 5G infrastructure, and to also benefit from authentication services that may be offered. As far as the Evaluation criteria are concerned, the KPI: Quality of connection is of crucial importance. The expected outcomes include checking whether the NetApp can allow both vApp to communicate. Additional prerequisites, constraints, restrictions involve the Anita computer actually use a Wi-Fi antenna for Internet access, or it may be necessary to use a dongle to reach 5G.

b. The application entitled" Retrieve User Equipment (UE) location information". The objective is to retrieve information about the geographical location of both devices: primary and secondary Anita. Moreover, for any type of composite repair, it is important to document the process as much as possible. In the case of the twin repair, geographic location information will be essential and should be included in the curing report. This information, gathered through the location service, can be used to obtain other data regarding climatic and environmental conditions, always with the goal of documenting the repair as much as possible. The user can also enter other information manually in the location and environment category. Finally, the expected outcomes are to obtain and store the geographic location information for both devices involved in the twin repair. Concerning the current status, currently there is no notion of location on the Anita device.

c. The third benefit of 5G access consists in ensuring that the communication will be maintained during the process. The name of the use case is "Start repair on primary Anita, retrieve and process data on secondary Anita". This third and final step is to initiate the repair cycle. It is essential that the data be transmitted continuously to ensure the twin repair process. On the primary Anita side, the probe data and the setpoint instruction must be sent. On the other hand, on the secondary Anita, the data must be read and then evaluated. Either to apply the same setpoint on the sample repair in real time, or to store the data for a study by a thermal study software. The setpoint instruction must be refreshed, received and processed by the secondary Anita, every 10 to 20 seconds maximum to ensure proper twin repair. The transfer time of the data packet must respect this refreshment constraint. Concerning the expected outcomes, we are interested in both the verification if a satisfying QoS can be reached during all the process, as well as the data integrity. Currently the status is that we have the possibility to transfer the baking data to our cloud database. This data is evaluated by web applications. It will be necessary to develop a reading module that will be used in the case of a slave Anita. EVOLVED-5G Innovations: Implementing an efficient architecture for our system to fully use the capabilities of the 5G network in terms of QoS (bandwidth, latency). In conclusion the evaluation criteria involve mainly the bandwidth and stability.

3.1.3.4 Detailed architecture

The NetApp developed by GMI uses the Python language and more precisely the FastAPI framework. This application receives requests from the Anita HMI application (vApp), then queries the NEF emulator to return desired results. These results are passed to the vApp using JSON format. The NetApp and the NEF emulator run on a specific machine, while the vApp runs on an embedded computer inside the Anita machine. This computer offers a 5G connection possibility by using a specific add-on card.

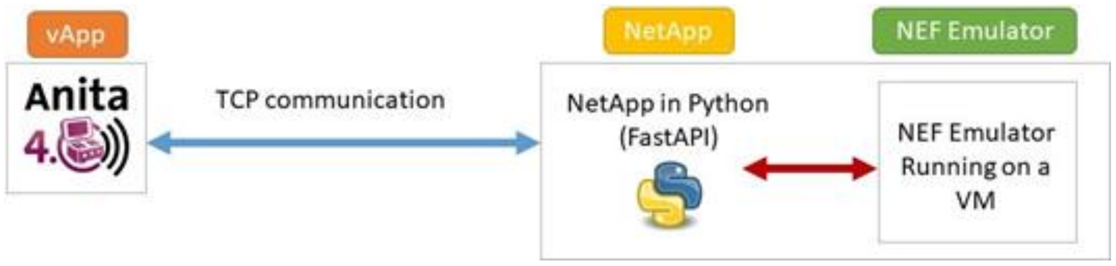
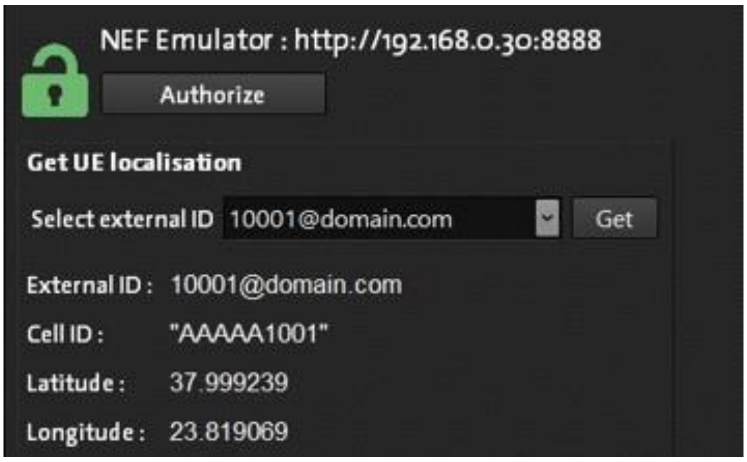


Figure 8. Digital/physical twin NetApp architecture

3.1.3.5 NetApp's Development tools

The development process follows the evolution of the features offered by the NEF emulator. Initially, requests were made directly to the emulator using the different APIs offered.
For the second version of the application that is described in this section, the Evolved-5G SDK was installed on the "server" machine. Now the requests use the functions offered by this SDK instead of directly querying the emulator via the APIs. In order to facilitate the deployment of the application and its dependencies, a file allowing to create a container has been set up. This file allows the deployment of the Python Framework used, the internal server used (uvicorn), and the installation of the Evolved-5G SDK. The code of the NetApp can be seen in the following repository: https://github.com/EVOLVED-5G/GmiAeroNetApp.

For the vApp side, the programming language used is C++ around the Embarcadero [15] development environment and the Visual Components Library (VCL) visual layer:



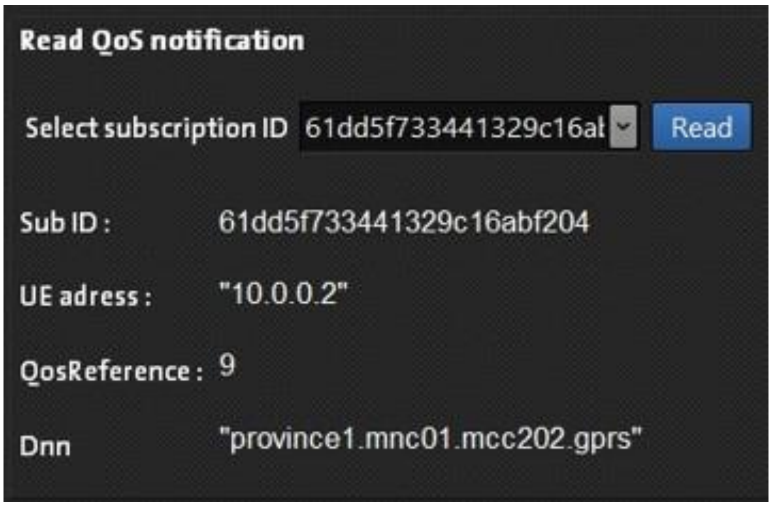
NEF Emulator : http://192.168.0.30:8888

Get UE localisation

Select external ID 10001@domain.com

External ID : 10001@domain.com
Cell ID : "AAAAA1001"
Latitude : 37.999239
Longitude : 23.819069

Figure 9 Geographic retrieval and display notifications



Read QoS notification

Select subscription ID 61dd5f733441329c16al Read

Sub ID : 61dd5f733441329c16abf204

UE adress : "10.0.0.2"

QosReference : 9

Dnn "province1.mnc01.mcc202.gprs"

Figure 10 Retrieving and displaying QoS

3.2 Pillar 2: Efficiency in FoF Operations (FoF)

3.2.1 CAFA-NetMapper NetApp

3.2.1.1 Use case short description

The computer vision (CV) vApp - CAFA - detects ArUco markers [16] and calculates camera coordinates. Later we plan to add a helmet or other Personal Protective Equipment (PPE) detection so we can calculate and send the message about the location of the safety equipment not worn.

3.2.1.2 Current limitations of the use case

The network speed and low latency are the crucial key elements of real-time –CV- applications. Indeed, this real-time computer vision applications need to have a powerful computer near the camera or in the local network to provide sufficient speed and latency. This would make the whole system more expensive and more difficult to maintain its hardware and software, especially if there are more than one camera points (for example, moving robots) in multiple locations.

3.2.1.3 Added value of 5G

The high network speed and low latency of 5G networks allows running real-time computer vision applications close to the performance as if they run in local computer. The low latency is crucial in directing moving robot in confined indoor environments to avoid its collision with people, building structures and other equipment. The notification about the QoS loss allows the vApp to stop the computer vision and send the signal to the robot to stop its movement. Access to 5G capabilities will allow using one central computer with fast connections to multiple camera points.

3.2.1.4 Detailed architecture

CAFA-NetMapper creates a GBR (Guaranteed Bit Rate) discrete automation subscription to the NEF Emulator to make it send notifications about QoS (Quality of Service) to NetApp’s Flask server’s endpoint at port 5555. Then the Flask server forwards the message to CAFA CV Flask server at port 5000 which changes the qos_state.txt file. The vApp’s ArUco_detection.py conducts its computer vision task when (according to the qos state file) QOS_GUARANTEED or quits the computer vision to prevent the unreliable results if QOS_NOT_GUARANTEED. The Python files – one for starting the Flask server and writing the QoS State message to file, and the ArUco marker detection program are located in the same directory. The OpenCV-based computer vision program detects ArUco markers from the IP-camera video feed and calculates the moving robot's camera coordinates in region defined by region.csv and markers.csv. Later we plan to add a helmet (or other safety equipment) detection. The architecture of the NetApp is presented in Figure 11.

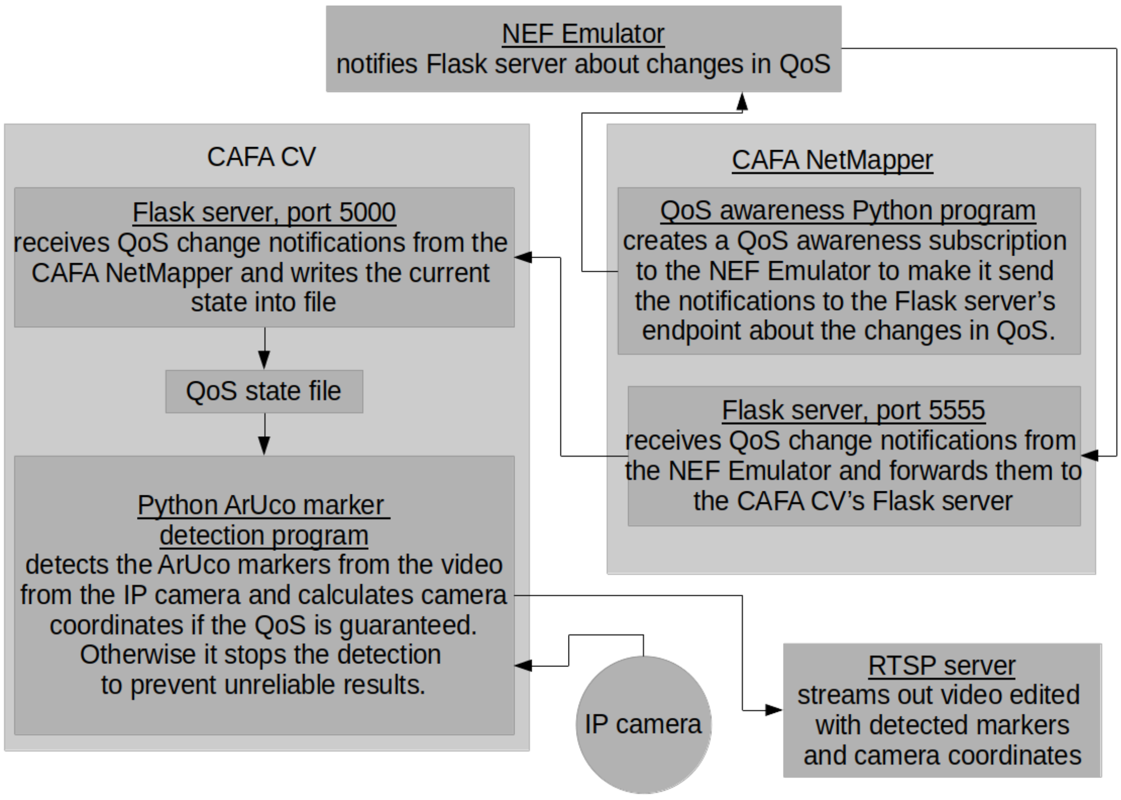

NEF Emulator
notifies Flask server about changes in QoS
CAFA CV
Flask server, port 5000
receives QoS change notifications from the CAFA NetMapper and writes the current state into file
QoS state file
Python ArUco marker detection program
detects the ArUco markers from the video from the IP camera and calculates camera coordinates if the QoS is guaranteed. Otherwise it stops the detection to prevent unreliable results.
CAFA NetMapper
QoS awareness Python program
creates a QoS awareness subscription to the NEF Emulator to make it send the notifications to the Flask server's endpoint about the changes in QoS.
Flask server, port 5555
receives QoS change notifications from the NEF Emulator and forwards them to the CAFA CV's Flask server
IP camera
RTSP server
streams out video edited with detected markers and camera coordinates

Figure 11. CAFA's NetApp architecture

```python
from evolved5g.swagger_client.rest import ApiException
from evolved5g.sdk import QosAwareness
import emulator_utils
from evolved5g.swagger_client import UsageThreshold


netapp_id = "CAFA-NetMapper"
equipment_network_identifier = "10.0.0.1"
host = emulator_utils.get_host_of_the_nef_emulator()
token = emulator_utils.get_token()
qos_awareness = QosAwareness(host, token.access_token)
network_identifier = QosAwareness.NetworkIdentifier.IP_V4_ADDRESS

gigabyte = 1024 * 1024 * 1024
usage_threshold = UsageThreshold(duration= None, # not supported
                                 total_volume=10 * gigabyte,  # 10 Gigabytes of total volume
                                 downlink_volume=5 * gigabyte,  # 5 Gigabytes for downlink
                                 uplink_volume=5 * gigabyte  # 5 Gigabytes for uplink
                                 )

notification_destination="http://172.17.0.2:5555/monitoring/callback"

def create_guaranteed_bit_rate_subscription_for_discrete_automation():
    discrete_automation = QosAwareness.GBRQosReference.DISCRETE_AUTOMATION
    uplink = QosAwareness.QosMonitoringParameter.UPLINK
    # Minimum delay of data package during uplink, in milliseconds
    uplink_threshold = 20

    subscription = qos_awareness.create_guaranteed_bit_rate_subscription(
        netapp_id=netapp_id,
        equipment_network_identifier=equipment_network_identifier,
        network_identifier=network_identifier,
        notification_destination=notification_destination,
        gbr_qos_reference=discrete_automation,
        usage_threshold=usage_threshold,
        qos_monitoring_parameter=uplink,
        threshold=uplink_threshold,
        wait_time_between_reports=10
    )
```

Figure 12 NetApp creates a subscription and notifies Flask server about QoS change

```python
from flask import Flask, request
from flask_cors import CORS
import requests

app = Flask(__name__)
CORS(app)
app.config["DEBUG"] = True

qos_state = "NOT ASSIGNED"

@app.route('/', methods=['GET'])
def index():
    return "NetApp web-server started"

@app.route('/monitoring/callback', methods=['POST'])
def qos_reporter():
    requests.post("http://172.17.0.3:5000/monitoring/callback", json=request.get_json())
    qos_state = request.get_json()['eventReports'][0]['event']
    print("Current state: " + qos_state)
    return qos_state

if __name__ == '__main__':
    print("initiating")
    app.run(host='0.0.0.0', port=5555)
```

Figure 13 Flask server receives the QoS change message and sends it to the vApp

3.2.1.5 NetApp's Development tools

The development process was carried out on local Ubuntu 20.04 computer, where the NEF Emulator 1.4.1 was built and run. The code of our detection CAFA CV and CAFA NetMapper has been developed following the example in SDK and was edited in Visual Studio Code. Docker was used to make a container including all the prerequisites for the CAFA CV. The code of the NetApp can be seen in the following repository: https://github.com/EVOLVED-5G/CafaTechNetApp2 and the code for the vApp can be found in https://github.com/EVOLVED-5G/CAFA_vApp respectively.

3.2.2 Smart irrigation 5G Agriculture NetApp

3.2.2.1 Use case short description

The smart irrigation use case is based on the deployment within the fields of a large number of sensors that are able to measure several parameters of interest, such as the Vapour Pressure Deficit (VPD) or Frequency Domain Reflectometry (FDR), which can be used to monitor the status of the terrain. This information, along with other parameters can be used for the creation of optimized irrigation plans, in order to make better use of hydric resources. In a later stage this use case will be extended with the inclusion of agricultural drones, in order to provide additional information for use during the irrigation plan processing.

3.2.2.2 Current limitations of the use case

In order to be effectively implemented, the smart agriculture use case requires the deployment of a large number of sensors to be distributed in the plantation, oftentimes in zones that are difficult to reach and impossible (at least cost-effectively) to communicate using wired technologies. This translates to a strong need for wireless solutions that make an efficient use of energy resources, making them easy to deploy and sustainable for longer periods of time without human intervention.

3.2.2.3 Added value of 5G

In this stage of the use case implementation, we can already identify two main benefits derived from the use of 5G connectivity:

- Sensors do not need to be pre-configured with information about the location where they will be deployed. This greatly reduces the complexity when preparing the terrain, and also allows immediate reuse of the sensors in another location when needed.
- Since the location information is taken from the network, sensors do not need to make use of GPS signals, which reduces both the energy consumption and the cost per sensor.

3.2.2.4 Detailed architecture

The general architecture of the components used in the smart irrigation test case can be seen in Figure 14. From left to right, we find several datalogger vApps, which obtain measurements from a set of sensors and sends them to the NetApp via 5G connectivity. The NetApp then makes use of the functionality provided by the NEF emulator in order to obtain information about the particular cell there the information was transmitted. This information is used for complementing the received data with information about the location where such measurements were obtained and is stored in a central database.

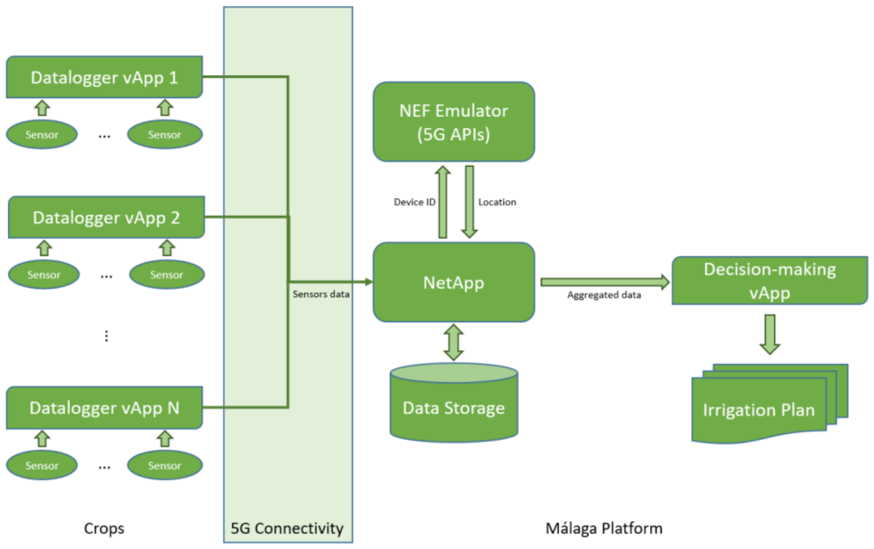


Figure 14. Smart Irrigation use case architecture

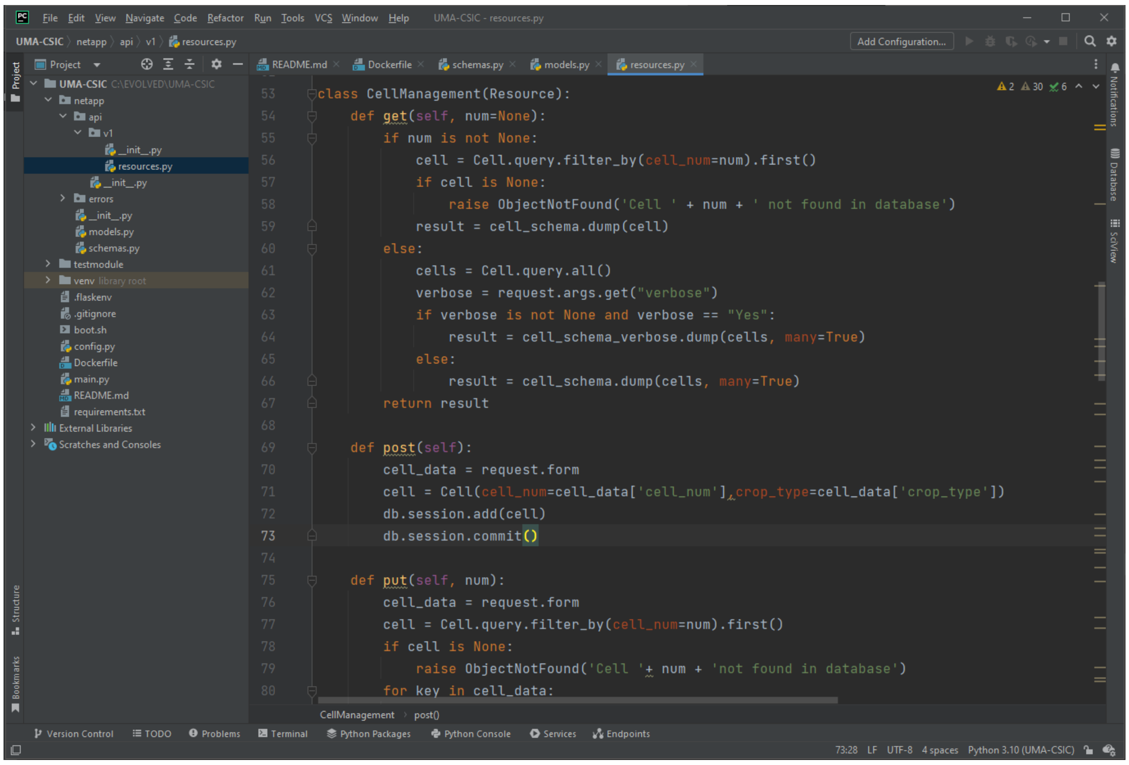
When required, the decision-making vApp can request data to the NetApp, which can be used in order to generate an optimized irrigation plan given the conditions of the terrain.
Internally, the NetApp is a Docker container, which exposes a REST API implemented using the Flask framework [17]. This API supports both for the reception of measurements from the dataloggers and the retrieval of data by the decision-making vApp, implemented.
Considering that at this stage it is not necessary to store a large quantity of data, and given that such data is only used for functionality testing, the NetApp stores all measurement using an internal SQLite database, however, database management has been implemented using the SQLAlchemy library [18], which allows the selection of a different backend, such as dedicated Postgresql or MySQL database.
The code of the NetApp can be seen in the following repository: https://github.com/EVOLVED-5G/UmaCsicNetApp/tree/evolved5g, which is part of the EVOLVED-5G organization in Github.

3.2.2.5 NetApp's Development tools

The NetApp has been developed using the EVOLVED-5G SDK libraries, and following the guidelines defined by the project. The resulting container has been deployed in the Málaga platform along with the NEF Emulator, which provides the required 5G functionality during this stage. Additionally, two kinds of vertical apps have been developed:

- Due to the impossibility of accessing the terrain and real dataloggers at this early stage, a datalogger vApp, which is able to send synthetic data to the NetApp has been used for testing the processing and storage of measurements. This vApp will be replaced by equivalent functionality but performed by the real dataloggers and sensors.
- An initial version of the decision-making vApp, which is in charge of creating the irrigation plan. This vApp has been used for testing the retrieval of processed information from the NetApp and is capable of displaying and calculating statistical data based on the received information.

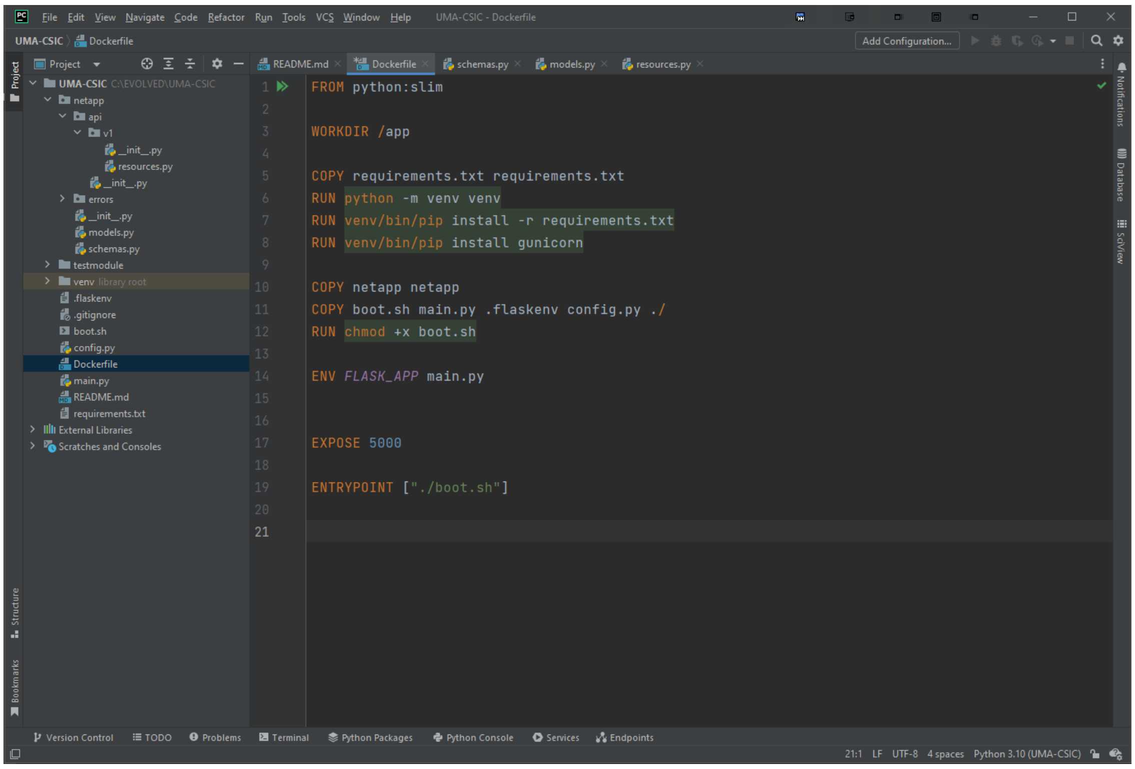
All of these applications have been developed using Python 3.10, and, depending on the environment of the developer, either Visual Studio Code or Pycharm as IDE. Figure 11 shows part of the code in the development environment.



```python
class CellManagement(Resource):
    def get(self, num=None):
        if num is not None:
            cell = Cell.query.filter_by(cell_num=num).first()
            if cell is None:
                raise ObjectNotFound('Cell ' + num + ' not found in database')
            result = cell_schema.dump(cell)
        else:
            cells = Cell.query.all()
            verbose = request.args.get("verbose")
            if verbose is not None and verbose == "Yes":
                result = cell_schema_verbose.dump(cells, many=True)
            else:
                result = cell_schema.dump(cells, many=True)
        return result

    def post(self):
        cell_data = request.form
        cell = Cell(cell_num=cell_data['cell_num'],crop_type=cell_data['crop_type'])
        db.session.add(cell)
        db.session.commit()

    def put(self, num):
        cell_data = request.form
        cell = Cell.query.filter_by(cell_num=num).first()
        if cell is None:
            raise ObjectNotFound('Cell '+ num + 'not found in database')
        for key in cell_data:
```

Figure 15. IDE and section of code of the UMA-CSIC NetApp

For local testing, Docker files have been created in order to ease the instantiation of the NetApp and vApps, one of them can be seen in Figure 12. For coordination between the developers and code versioning Git has been used.



```dockerfile
FROM python:slim

WORKDIR /app

COPY requirements.txt requirements.txt
RUN python -m venv venv
RUN venv/bin/pip install -r requirements.txt
RUN venv/bin/pip install gunicorn

COPY netapp netapp
COPY boot.sh main.py .flaskenv config.py ./
RUN chmod +x boot.sh

ENV FLASK_APP main.py


EXPOSE 5000

ENTRYPOINT ["./boot.sh"]

```

Figure 16. Dockerfile of the UMA-CSIC NetApp

3.2.3 Industrial grade 5G connectivity with assured QoS and integrated SLA/SLS monitoring capabilities NetApp

3.2.3.1 Use case short description

Based on the assumption specific IoT and M2M devices used in FoF require a stable communication environment (described by certain requirements such as bandwidth, latency, security policy, etc.), the use case contributes to the efficiency in FoF operations by research and development on a NetApp which provides industrial 5G connectivity for those IoT and M2M devices, assuring them required QoS with the help of integrated SLA/SLS monitoring capabilities as part of the NetApp. The NetApp is integral part of the *5G IoT System*, a product that serves for providing 5G connectivity for IoT/M2M devices being connected to it via various physical interfaces (e.g., serial, USB, Ethernet). More thorough high-level description and details of the use case can be found in deliverables D2.1 and D4.1.

3.2.3.2 Current limitations of the use case

Stable communication link connecting multiple data acquiring, collecting, and processing modules involved in a complex factory/production process, where these modules are distributed physically (e.g., end-device, edge cloud, central cloud), is of vital importance for the positive outcome of the factory process. Solutions, available nowadays, addressing the challenge are mostly limited to wired connections, e.g., industrial ethernet, while wireless/mobile solutions of that kind are currently less usual, although they may provide additional value for the customer such as enabling mobility of end-devices, no need for cabling, etc.

3.2.3.3 Added value of 5G

Since 5G has the potential to offer network performances required by the factory process, the solution/NetApp (in combination with the corresponding vApp) presented utilizes 5G network functions available, to provide assured QoS for the communication links, i.e., the NetApp's responsibility is to configure and monitor the 5G radio network QoS according to SLA requirements set via vApp. 5G enables the NetApp to collect various radio, network and cloud related performance metrics (e.g. location-based info such as Cell Id/gNB id, QoS status) used as a quality indicator of the 5G NR, as well, within the 5G, the user has, among others, a possibility to select the preconfigured slice to accommodate the SLA requirements.

3.2.3.4 Detailed architecture

The NetApp is implemented as an additional module within ININ's *5G IoT System* (vApp) environment that consists of:

- IoT Gateway (serving as UE device, providing radio KPIs, forwarding of critical and non-critical data),
- IoT Management (provides centralized management of IoT gateways),
- IoT Collector (central storage for collecting data from IoT gateways),
- IoT Reporter (KPIs visualization).

The overall architecture in the Figure 17 shows the interaction of the NetApp with other components of the *5G IoT System* (vApp). The main role of the NetApp is to provide:

- API endpoint for the requests coming from IoT Management component (e.g., registration request for a specific IoT Gateway/UE),

- API client towards the 5G NEF,
- Receiving callbacks/notifications from the NEF (e.g., Monitoring Event Reporting, QoS status)
- Pushing gathered KPIs from NEF to IoT Collector (e.g., for post-analytics).

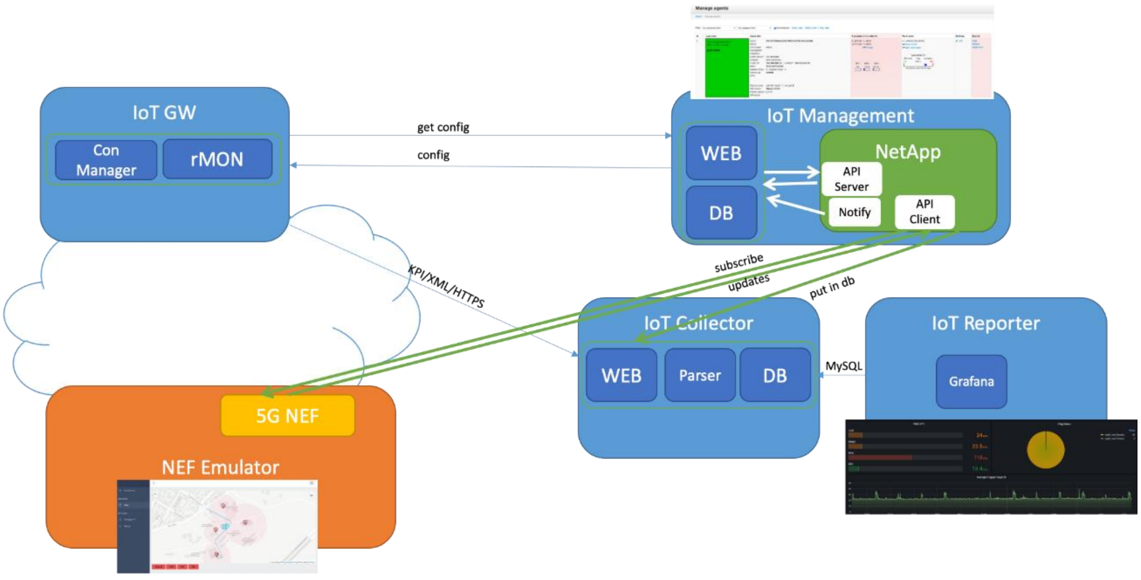


Figure 17: Integration of the NetApp in the "Industrial grade 5G connectivity with assured QoS and integrated SLA/SLS monitoring capabilities" environment

On the API Client side, the NetApp itself integrates with the 5G NEF by supporting authentication mechanism (i.e., obtaining and validating the token) and subscribing to the to the 5G NEF API functions (currently "Event Monitor" and "Session with QoS") and additionally, providing "Callback URL" to the 5G NEF. The "Callback URL" is then used by the 5G NEF to provide notifications about the state of the 5G network and/or UE (i.e., Cell ID of the gNB serving the UE, QoS GUARANTEED/NOT GUARANTEED status) to the NetApp that is listening on the interface that must be reachable from the 5G NEF via "Callback URL".

The NetApp's API server exposes NEF capabilities to the 5G IoT Management component. With this, the 5G IoT Management user is able to register the UE managed by the 5G IoT Management to the 5G NEF API and also request that the UE is assigned selected QoS profile. The procedure includes the subscription to 5G NEF "Monitoring Event" API, which is done automatically on register, and if successful, the subscription to "Session with QoS" API is performed afterwards.

After the UE is assigned the correct QoS profile and the NetApp is receiving API updates from the 5G NEF, the NetApp also pushes the received KPIs to the *IoT Collector* for long term storage and post-analytics scenarios while also pushing real-time updates back to the IoT *Management* component which uses the info to dynamically control *5G IoT Gateway* services (e.g., traffic steering).

The example of described workflow is presented in the following figure.

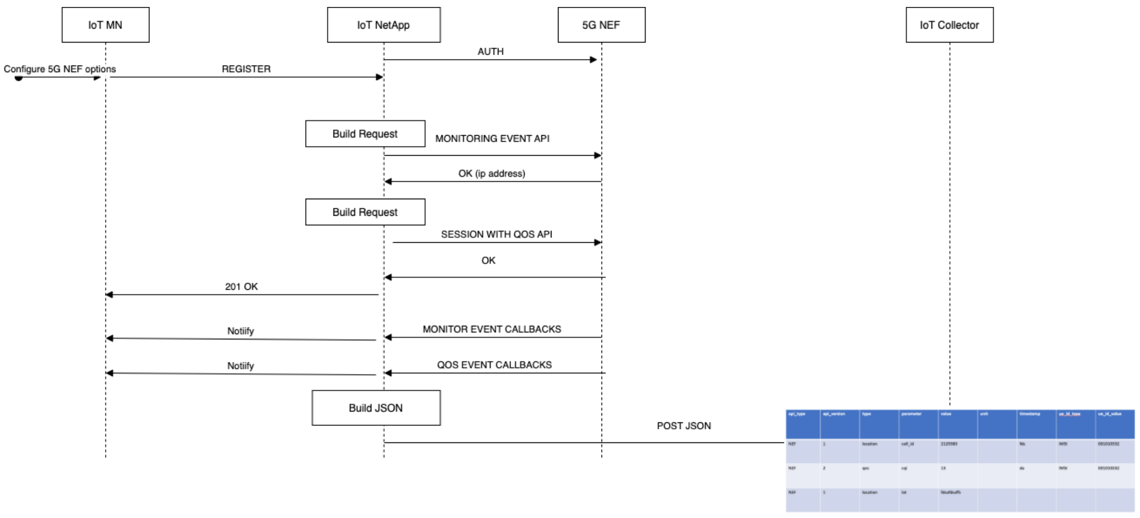
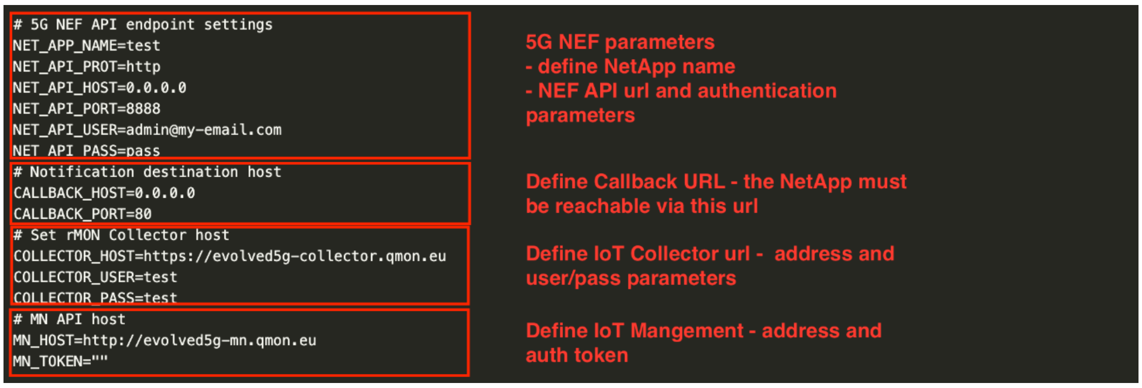


Figure 18: NetApp operation workflow example

3.2.3.5 NetApp's Development tools

The NetApp source code is developed in python3 language and is packaged as a docker container. All external parameters are configured via environmental variables that can be passed to the container at runtime. For the convenience (e.g., CI/CD, scripted deployments), the NetApp comes with docker-compose template and ".env" sample which is presented below:



```
# 5G NEF API endpoint settings
NET_APP_NAME=test
NET_API_PROT=http
NET_API_HOST=0.0.0.0
NET_API_PORT=8888
NET_API_USER=admin@my-email.com
NET_API_PASS=pass
# Notification destination host
CALLBACK_HOST=0.0.0.0
CALLBACK_PORT=80
# Set rMON Collector host
COLLECTOR_HOST=https://evolved5g-collector.qmon.eu
COLLECTOR_USER=test
COLLECTOR_PASS=test
# MN API host
MN_HOST=http://evolved5g-mn.qmon.eu
MN_TOKEN=""
```

5G NEF parameters
- define NetApp name
- NEF API url and authentication parameters

Define Callback URL - the NetApp must be reachable via this url

Define IoT Collector url - address and user/pass parameters

Define IoT Mangement - address and auth token

Figure 19: NetApp "env" sample

The NetApp is integrated with the vApp as shown in the architecture (represented in Figure 17). Communication with the vApp is based on REST API offered by the NetApp with the vApp acting as a client towards the NetApp (e.g., to register IoT GW in the 5G NEF). At the same time, the vApp is acting as a server to support notifications and gathering relevant data from the NetApp to the vApp side (e.g., show current Cell ID in the IoT Management UI). The vApp side is based on well-known web technologies, such as PHP/MySQL for the IoT Management API/UI and MySQL for IoT Collector, with appropriate extensions developed to support presented integration with the NetApp.

For testing and development purposes, the local instance of 5G NEF emulator was deployed. The NetApp's API client functionality which targets 5G NEF API was fully integrated using 5G NEF Python SDK developed during EVOLVED-5G project. The code of the NetApp can be seen in the following repository: https://github.com/EVOLVED-5G/IninRmonNetApp.

3.2.4 Anomaly Detection NetApp

3.2.4.1 Use case short description

Zortenet's use case as described in D2.1 and D4.1 is the detection of network anomalies caused by potential cyber-attacks to industrial 5G NPNs and the application of mitigation actions to ensure the normality of the operations that may take place in a factory.

The developed NetApp is responsible to:

- collect and process 5G network traffic
- forward processed analytics to an anomaly detection framework that is deployed to a Vertical App.

3.2.4.2 Current limitations of the use case

The missing standards-based mechanisms that allow analytics and information from the 5G System to be accessible at the Network Monitoring and Anomaly Detection applications, affect both the operation and accuracy of the anomaly detection as well as make all solutions become proprietary and customised for each particular deployment. In most of the cases were 5G specific information is required, the proper operation is achieved either by 3rd party appliances installed in-premises (customer side) and/or proprietary interfaces and engineering hacks are required.

3.2.4.3 Added value of 5G

The availability of rich and on-demand access to 5G system monitoring and analytics via NEF, allows a variety of inputs to become available at the Network Anomaly detection application. The development, integration and validation burden are minimised and solutions can be built on top on well-behaving, tested and certified software APIs. The need to provide workarounds in accessing in-premises information is limited. The accuracy of anomaly detection, can now be improved by exploiting 5G esoteric information otherwise not available outside of customised Non-Public Network (NPN) installations.

3.2.4.4 Detailed architecture

Zortenet's NetApp is an application based on python's flask module that retrieves information from NEF emulator. The vApp (also based on flask) receives processed information from the NetApp and notifies the user for an anomaly through an administration dashboard.

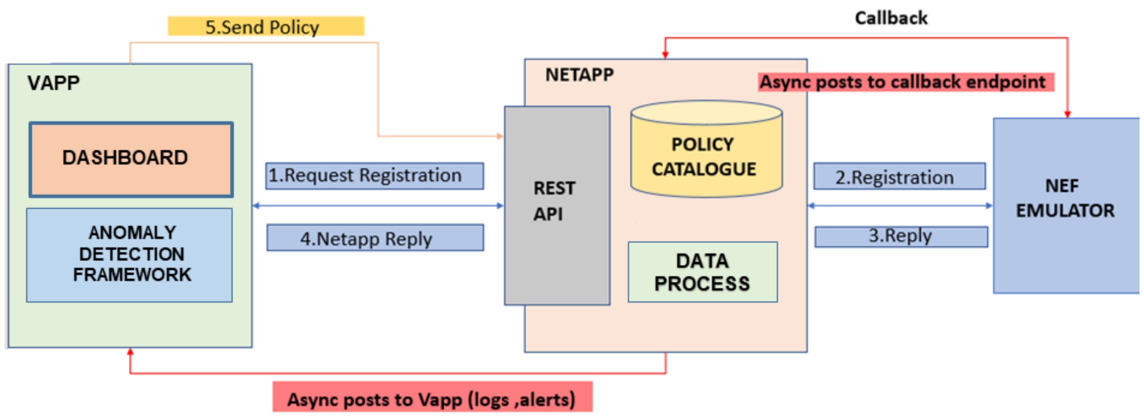


Figure 20: Overall Communication between components

The figure above provides a high-level overview of the communication between the components involved in Zortenet's use case. The communication between the NetApp and the NEF emulator is achieved by using the Evolved5G SDK.
For this early phase of the project and the current functionalities of the NEF emulator some abstractions were made to facilitate the development process. Instead of network traffic we collect location information, and we apply policies regarding the cell usage of each UE. With this proof-of-concept demo we intend to show that we can achieve a monitoring capability that is able to do some processing and then relay this information to a third-party application (vApp).
The whole sequence is:

- NetApp is running and getting an access token
- vApp admin subscribes to receive location information about UEs
- NetApp receives the subscription and forwards it to NEF emulator
- NetApp is receiving monitoring events and relays them to vApp
- vApp is receiving those events and shows them to the Dashboard
- vApp admin is creating a policy regarding the permitted cells
- NetApp receives and stores the policy to its internal database
- NetApp process each new record and tags them as an anomaly if any of the stored policies is violated and forwards this new information to vApp
- vApp is receiving logs and differentiates normal and anomalous behaviour.

Following we provide a step-by-step explanation of the whole sequence. Using the default scenario running on NEF emulator there are three different UEs moving as we can see from the integrated map to the vApp Dashboard.

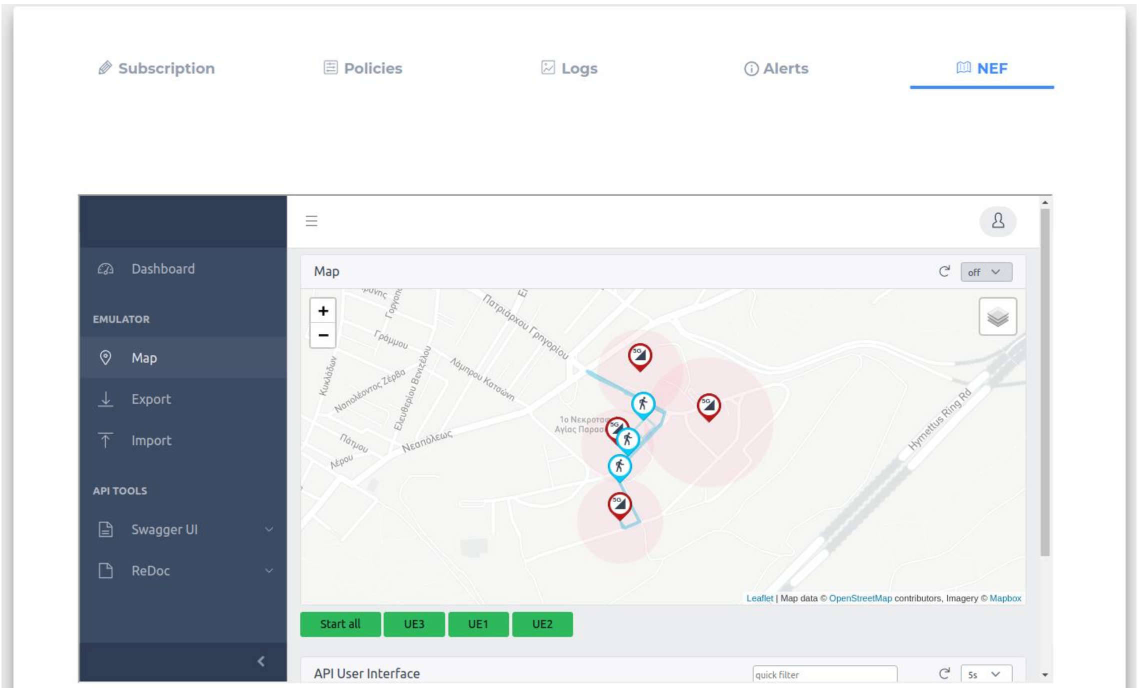
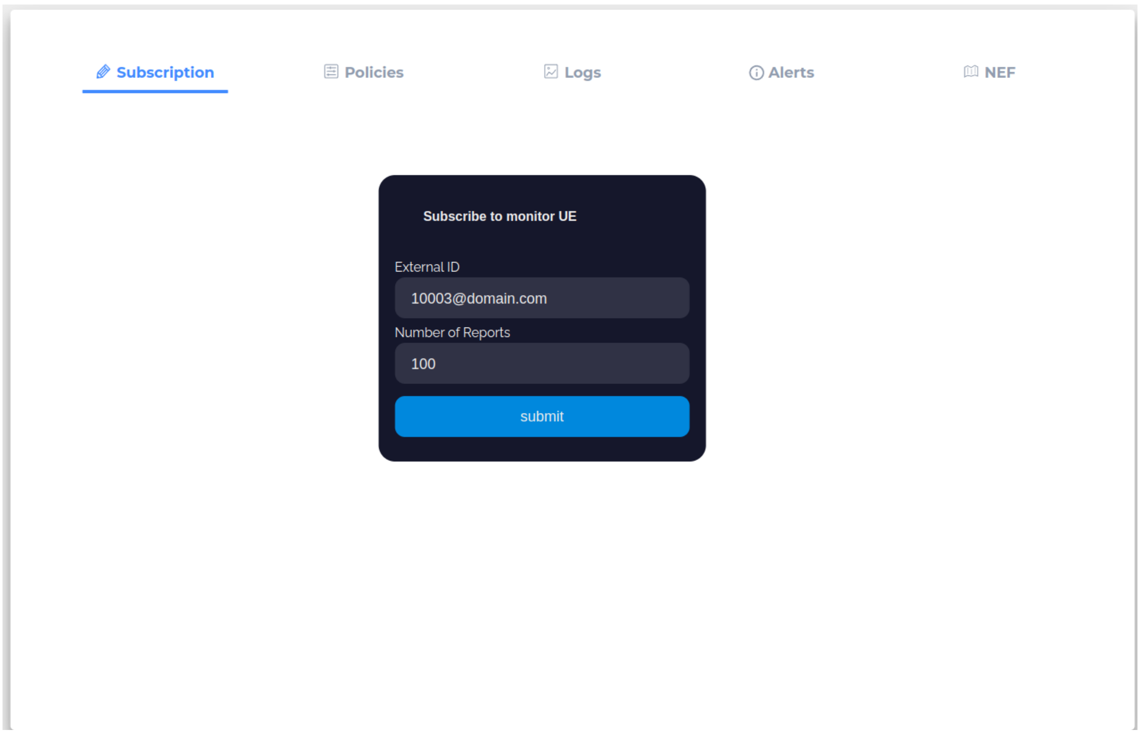


Figure 21: Map of NEF form the vApp Dashboard

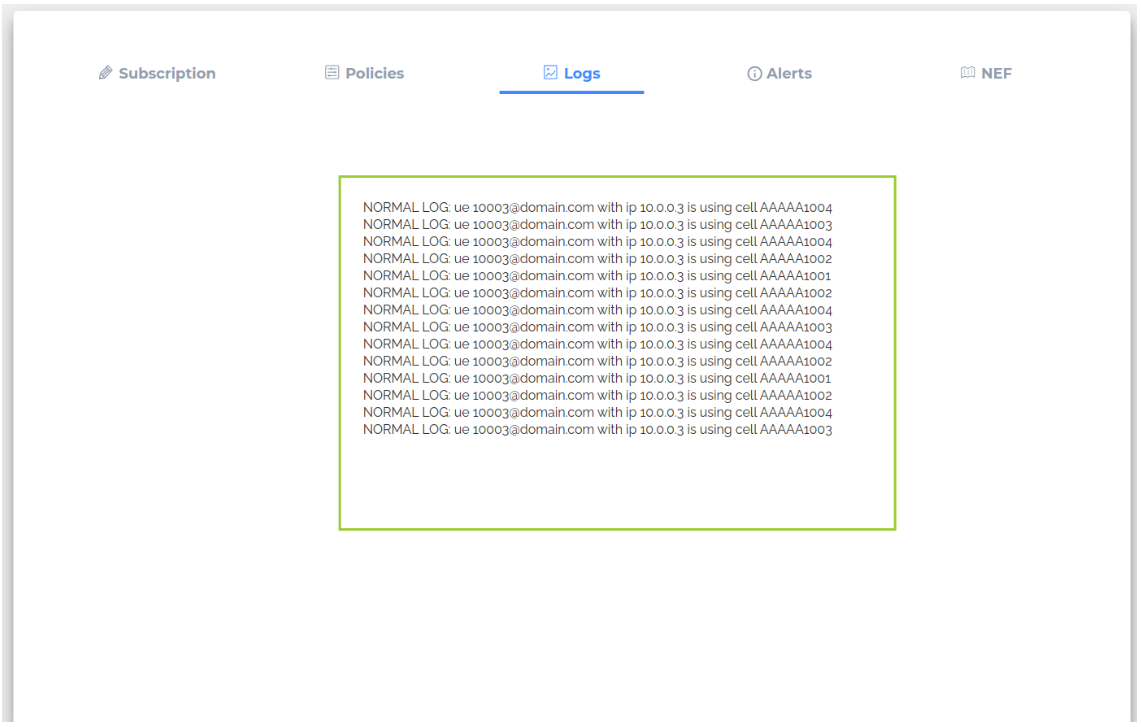
When the NetApp is up and running a request is made to the NEF emulator to get a token that provides access to NEF's API functionalities. The administrator of the vApp is able now to create a location subscription for a specific or multiple UEs and receive logs using the LocationSubscriber class of the Evolved5G SDK.



External ID: 10003@domain.com
Number of Reports: 100

Figure 22: Example of Location subscription for specific UE

After the subscription is made the administrator can receive logs for the specified UE as we can see in the figure below.



```
NORMAL LOG: ue 10003@domain.com with ip 10.0.0.3 is using cell AAAAA1004
NORMAL LOG: ue 10003@domain.com with ip 10.0.0.3 is using cell AAAAA1003
NORMAL LOG: ue 10003@domain.com with ip 10.0.0.3 is using cell AAAAA1004
NORMAL LOG: ue 10003@domain.com with ip 10.0.0.3 is using cell AAAAA1002
NORMAL LOG: ue 10003@domain.com with ip 10.0.0.3 is using cell AAAAA1001
NORMAL LOG: ue 10003@domain.com with ip 10.0.0.3 is using cell AAAAA1002
NORMAL LOG: ue 10003@domain.com with ip 10.0.0.3 is using cell AAAAA1004
NORMAL LOG: ue 10003@domain.com with ip 10.0.0.3 is using cell AAAAA1003
NORMAL LOG: ue 10003@domain.com with ip 10.0.0.3 is using cell AAAAA1004
NORMAL LOG: ue 10003@domain.com with ip 10.0.0.3 is using cell AAAAA1002
NORMAL LOG: ue 10003@domain.com with ip 10.0.0.3 is using cell AAAAA1001
NORMAL LOG: ue 10003@domain.com with ip 10.0.0.3 is using cell AAAAA1002
NORMAL LOG: ue 10003@domain.com with ip 10.0.0.3 is using cell AAAAA1004
NORMAL LOG: ue 10003@domain.com with ip 10.0.0.3 is using cell AAAAA1003
```

Figure 23: Sample of received logs

Since no policy is applied yet this behaviour is considered normal. Using the policy management tab, the administrator can create a policy and restrict the cell usages.

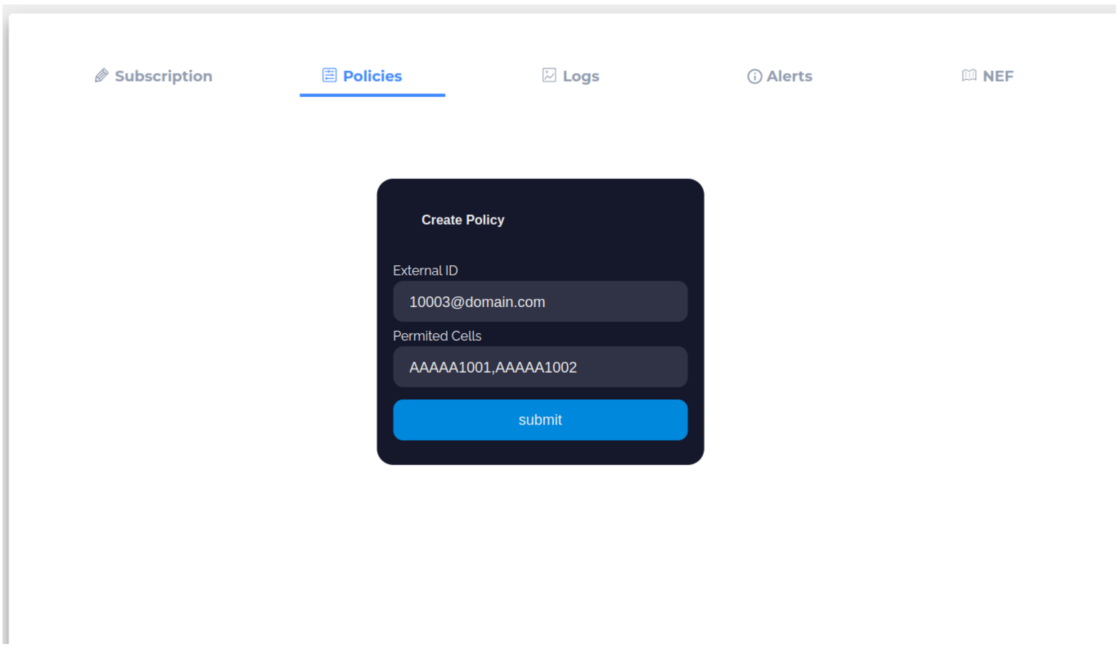
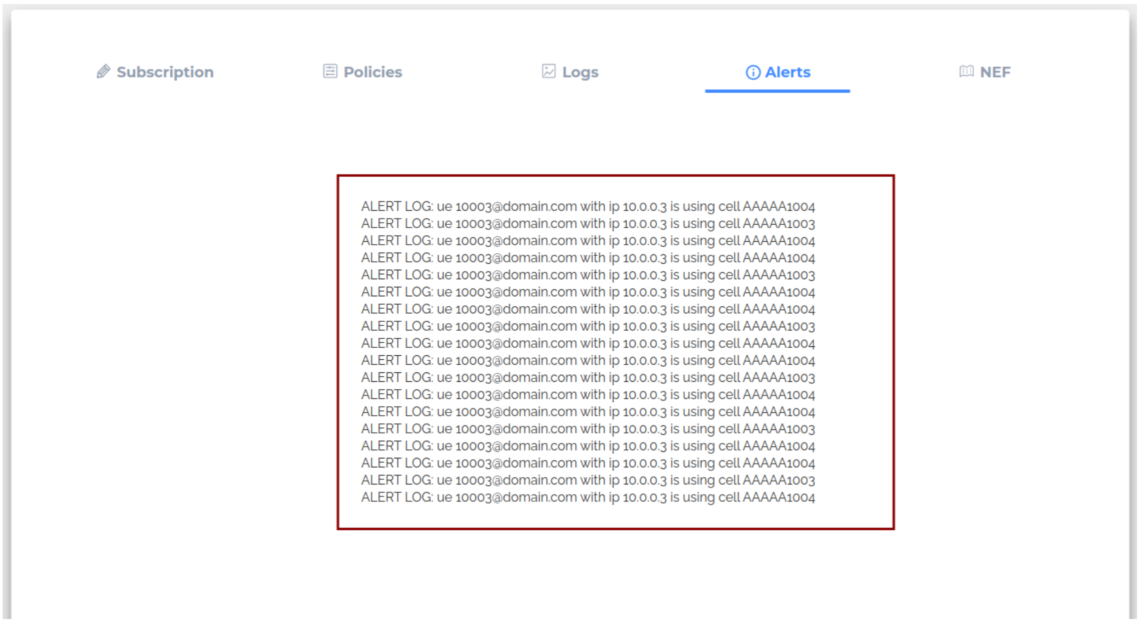


Figure 24: Example of policy applied denoting the permitted cells

When the policy is applied the vApp makes a post request to NetApp and the policy is stored to an internal database (mongodb). Each new record received from the NEF emulator is examined by the data process sub-component of the NetApp to check if the policy is violated (The specified UE is using restricted cells). If the policy is violated the record is tagged as an anomaly and is then forwarded to the vApp that will make the differentiation between a normal log and an alert and will finally display the alert to the dedicated tab.



```
ALERT LOG: ue 10003@domain.com with ip 10.0.0.3 is using cell AAAAA1004
ALERT LOG: ue 10003@domain.com with ip 10.0.0.3 is using cell AAAAA1003
ALERT LOG: ue 10003@domain.com with ip 10.0.0.3 is using cell AAAAA1004
ALERT LOG: ue 10003@domain.com with ip 10.0.0.3 is using cell AAAAA1004
ALERT LOG: ue 10003@domain.com with ip 10.0.0.3 is using cell AAAAA1003
ALERT LOG: ue 10003@domain.com with ip 10.0.0.3 is using cell AAAAA1004
ALERT LOG: ue 10003@domain.com with ip 10.0.0.3 is using cell AAAAA1004
ALERT LOG: ue 10003@domain.com with ip 10.0.0.3 is using cell AAAAA1003
ALERT LOG: ue 10003@domain.com with ip 10.0.0.3 is using cell AAAAA1004
ALERT LOG: ue 10003@domain.com with ip 10.0.0.3 is using cell AAAAA1004
ALERT LOG: ue 10003@domain.com with ip 10.0.0.3 is using cell AAAAA1003
ALERT LOG: ue 10003@domain.com with ip 10.0.0.3 is using cell AAAAA1004
ALERT LOG: ue 10003@domain.com with ip 10.0.0.3 is using cell AAAAA1004
ALERT LOG: ue 10003@domain.com with ip 10.0.0.3 is using cell AAAAA1003
ALERT LOG: ue 10003@domain.com with ip 10.0.0.3 is using cell AAAAA1004
ALERT LOG: ue 10003@domain.com with ip 10.0.0.3 is using cell AAAAA1004
ALERT LOG: ue 10003@domain.com with ip 10.0.0.3 is using cell AAAAA1003
ALERT LOG: ue 10003@domain.com with ip 10.0.0.3 is using cell AAAAA1004
```

Figure 25: Example of alerts as shown from vApp Dashboard

3.2.4.5 NetApp's Development tools

The NetApp is a dockerized application and is consisted by two containers. One for the flask application and another for the internal mongodb. As mentioned, the communication between the NetApp and the NEF emulator is achieved by using the Evolved5G SDK and is provided in the requirements file in order to be installed via pip when the image of the application is built. The vApp is also using the flask module for the creation of its API and the dashboard has beencreated with pure HTML, CSS, javascript and the EEL module that binds javascript with python. The code of the NetApp can be seen in the following repository: https://github.com/EVOLVED-5G/ZortenetNetApp .

3.3 Pillar 3: Security guarantees and risk analysis

3.3.1 Traffic Management NetApp

3.3.1.1 Use case short description

The use cases that are implemented for the and are applied to the Traffic Management NetAppcan be described as:

•Use Case 1: The accurate measurement of traffic over an interface, specific to a device and the performing of a simple check of "unregistered" traffic outside the 5G Core network.

•Use Case 2: The lessening of the burden on a reportedly congested device in the network, applicable to many different aspects of a FoF.

Traffic Management NetApp, being developed for security guarantees risk analysis pillar and in the scope of the EVOLVED-5G project, offers the following services:

- The accurate measurement of traffic over an interface, specific to a device and perform a simple check of "unregistered" traffic outside the 5G Core network.
- The lessening of the burden on a reportedly congested device in the network, applicable to many different aspects of a FoF.
- Real-time monitoring of events and managing the security of Internet Protocol (IP)

3.3.1.2 Current limitations of the use case

Current networks lack the security protection and real time monitoring of threats, which are necessary in order to ensure the functionality of an industrial environment. Consequently, network monitoring and data acquisition and analysis cannot be efficient by using existing 4G technologies. 8Bells, with the development of its NetApp and vApp, aims to enhance the security level between the 5G Core and the external communication network.

3.3.1.3 Added value of 5G

Increased bandwidth, lower latency and increased availability are offered by 5G networks and are mandatory in order to properly secure the network. The security administrator of the network is able not only to counter a threat more efficiently, but also to avoid one, using the increased security measures provided by the 5G Core Network. 8Bells proposed solution with NetApp (in conjunction with vApp) provides monitoring of the 5G core through the proper use of suitable APIs.

3.3.1.4 Detailed architecture

The main operation and mechanism (for detecting congestion and limits or redirects) being performed by Eight Bells proposed and developed system, is the following:

- L7 Switch requests for congestion statistics from NetApp which is subscribed to MonitoringEvent API
- The processed congestion info (of NetApp) supplies the list with devices/destination IPs with high congestion stats
- L7 Switch requests for new congestion statistics from NetApp
- The processed congestion info resupplies the list with devices/destination IPs with high congestion stats
- L7 Switch requests for further processing of traffic Filters which are subscribed to MonitoringEvent API
- The processed congestion info resupplies the list with devices/destination IPs with high congestion stats
- No further processing takes place

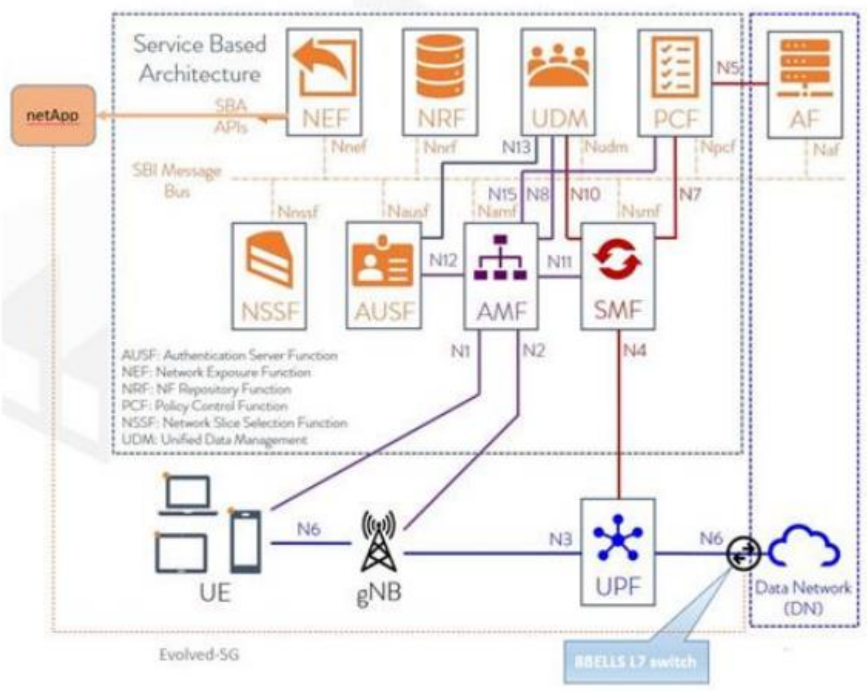
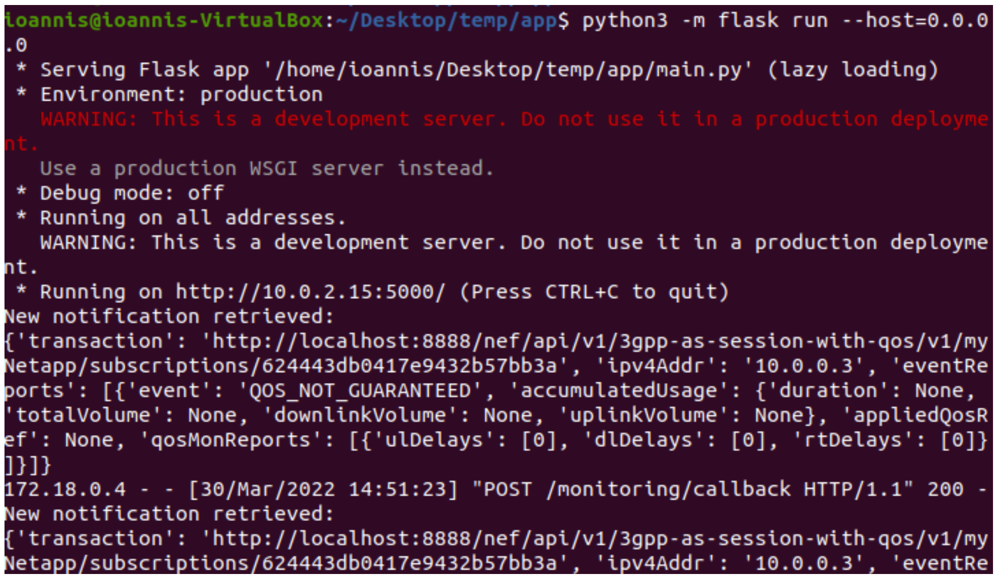

Service Based Architecture
netApp
SBA APIs
NEF
NRF
UDM
PCF
AF
N5
Nnef
Nnrf
N13
Nudm
Npcf
Naf
SBI Message Bus
Nnssf
Nausf
N15 Namf
N8
N10
Nsmf
N7
NSSF
AUSF
N12
AMF
N11
SMF
N1
N2
N4
AUSF: Authentication Server Function
NEF: Network Exposure Function
NRF: NF Repository Function
PCF: Policy Control Function
NSSF: Network Slice Selection Function
UDM: Unified Data Management
UE
N6
gNB
N3
UPF
N6
Data Network (DN)
Evolved-5G
8BELLS L7 switch

Figure 26: NetApp – vApp (L7 Switch) schematic

3.3.1.5 NetApp's Development tools

The NetApp is developed using flask-python. The NEF emulator runs locally, on the same machine as the NetApp. Currently NEF supports two APIs: one for location and one for QOS monitoring. Currently, the NetApp uses post and get requests to those endpoints of the NEF emulator, in order to get notifications and data that will be used by the vApp in its current version, the NetApp utilizes AsSessionwithQoS API in order to get the IP addresses of all subscribed UEs from the emulator, and the endpoint regarding the cell id of UEs, in order to detect congestions. The Evolved SDK is also implemented in order to make requests to the NEF emulator from the NetApp. The code of the NetApp can be seen in the following repository: https://github.com/EVOLVED-5G/8BellsNetApp.



```
ioannis@ioannis-VirtualBox:~/Desktop/temp/app$ python3 -m flask run --host=0.0.0
.0
 * Serving Flask app '/home/ioannis/Desktop/temp/app/main.py' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployme
nt.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on all addresses.
   WARNING: This is a development server. Do not use it in a production deployme
nt.
 * Running on http://10.0.2.15:5000/ (Press CTRL+C to quit)
New notification retrieved:
{'transaction': 'http://localhost:8888/nef/api/v1/3gpp-as-session-with-qos/v1/my
Netapp/subscriptions/624443db0417e9432b57bb3a', 'ipv4Addr': '10.0.0.3', 'eventRe
ports': [{'event': 'QOS_NOT_GUARANTEED', 'accumulatedUsage': {'duration': None,
'totalVolume': None, 'downlinkVolume': None, 'uplinkVolume': None}, 'appliedQosR
ef': None, 'qosMonReports': [{'ulDelays': [0], 'dlDelays': [0], 'rtDelays': [0]}
]}]}
172.18.0.4 - - [30/Mar/2022 14:51:23] "POST /monitoring/callback HTTP/1.1" 200 -
New notification retrieved:
{'transaction': 'http://localhost:8888/nef/api/v1/3gpp-as-session-with-qos/v1/my
Netapp/subscriptions/624443db0417e9432b57bb3a', 'ipv4Addr': '10.0.0.3', 'eventRe
```

Figure 27: NEF emulator – QOS notification received

vApp (L7 Switch) performs two functions:

- Firewall capabilities: apply to UC1 based on the IP addresses that the NetApp provides.
- Traffic limiting capabilities: apply to UC2 on congested devices that don't pass the QOS check.

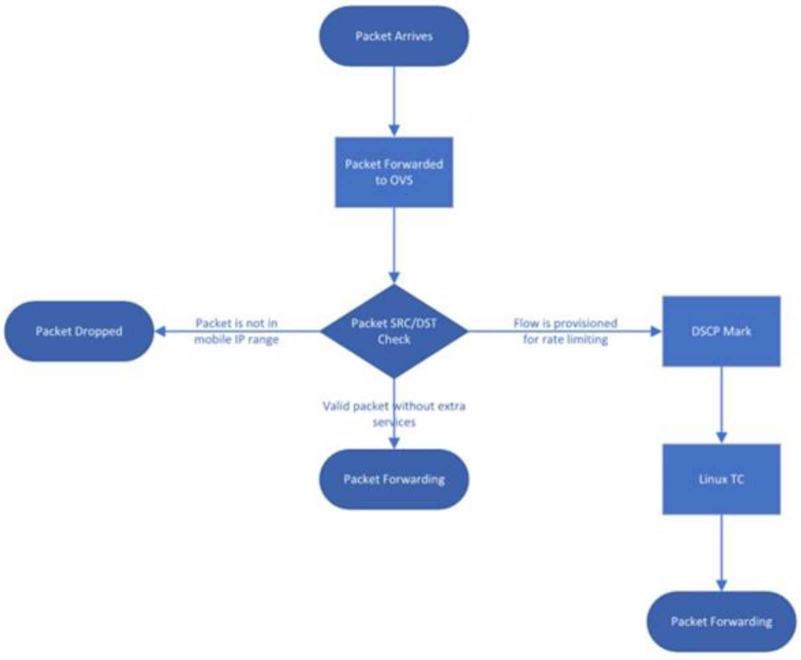


Figure 28: vApp configuration

The vApp (switch) uses standard OVS and TC software to provide firewalling and traffic limiting capabilities. By default, two Open Virtual Switch (OVS) enabled bridges are created on the physical machine hosting the switch.

The first bridge (OVSbr1) is used to route traffic between firewall bridge and system routing engine including the physical NIC of the host. The second bridge (namely firewall) is used to provide firewalling capabilities and also tag flows that have to be throttled.

USE Case 1: Firewall (*running*)
By default, all flows coming through the firewall bridge are dropped. Only flows that are provisioned by the NetApp are allowed and routed. In more details, NetApp provisions flows that are allow to table 100 and 102 of firewall bridge (table 100 include the source IP address of the terminals attached to 5G network and 102 a similar rule with the same IP address as destination to match ingress flows toward the terminal).

USE Case 2: Throttling
We use OVS tables and Differentiated Services Code Point (DSCP) marking to classify flows that need to be throttled. Similarly, flows that need to be throttled are provisioned to table 101 and 103. OVS marks packets with a DSCP value that is agreed and assigned to a specific value of maximum bitrate, before forwarding them to OVSbr1 (using a “patch” port). TC on the physical host throttles all traffic with aforementioned DSCP value as traffic flows towards physical Network Interface Card (NIC) attached to Operating System (OS).

3.3.2 ID Management and Access Control NetApp

3.3.2.1 Use case short description

- NetApps identity validation and authorization.

The IQB’s NetApp has the purpose of authenticating other NetApps using OpenID Connect (a simple identity layer on top of the OAuth 2.0 protocol [19]), as well as authorizing the consumption of NEF APIs under the event of successful authentication

3.3.2.2 Current limitations of the use case

The innovation of identity validation applied in mobile network authentication implies slightly inferior performance due to the overhead added for the identity management. Computational power and network performance are variables that have limited the implementation of such security measures until now.

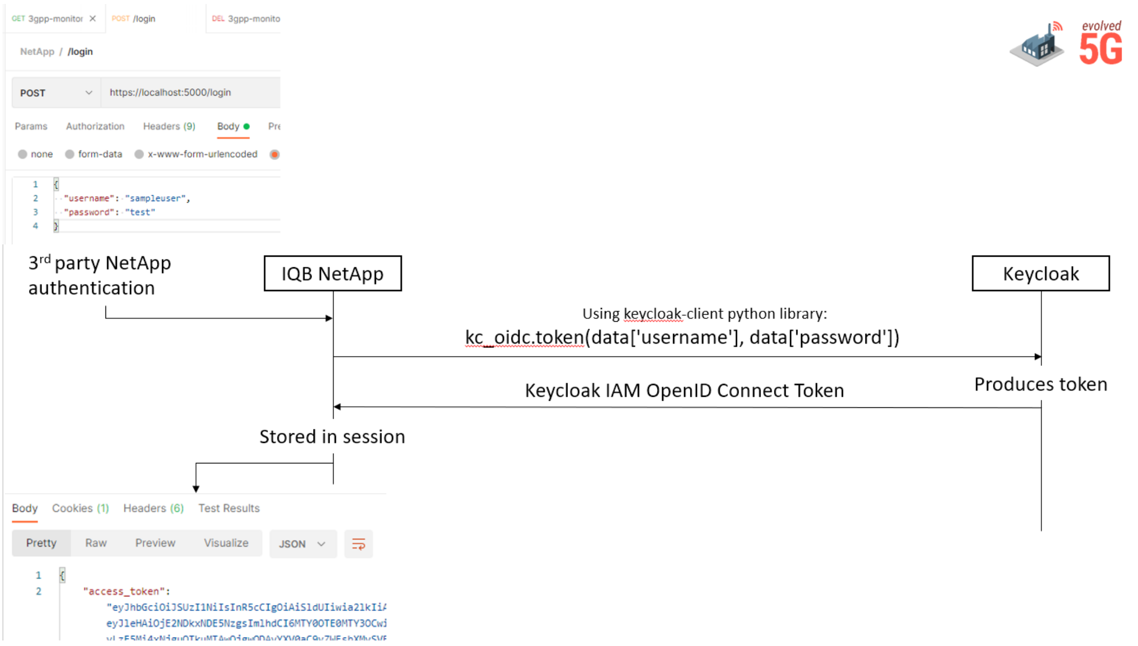
3.3.2.3 Added value of 5G

The technology of 5G has two main benefits, increased bandwidth and lower latency. The enhanced operation of 5G networks allow the usage of different SaaS to communicate rapidly from different locations, in order to perform identity and access management without obstructing network service. Furthermore, the capability of 5G to monitor and obtain information about UEs gives the opportunity to perform continuous authentication and restrict or lower the access level in case of suspicious behaviour.

3.3.2.4 Detailed architecture

The NetApp is a Python (flask) API running inside a docker container. The container communicates with another container running the Keycloak open-source Identity and Access Management solution [20] to provide OpenID Connect Functionality. The NetApp provides endpoints for authentication and consumption of the NEF Emulator APIs. A 3rd party can only consume the NEF Emulator APIs after authentication and authorization, otherwise access is declined. The 3rd party can be de-authenticated in case of misuse of the NetApp.

Once a 3rd party attempts to authenticate with the correct credentials stored in Keycloak, the NetApp confirms their identity and stores the OAuth2.0 Token generated by Keycloak in the session variable.



```
https://localhost:5000/login
```

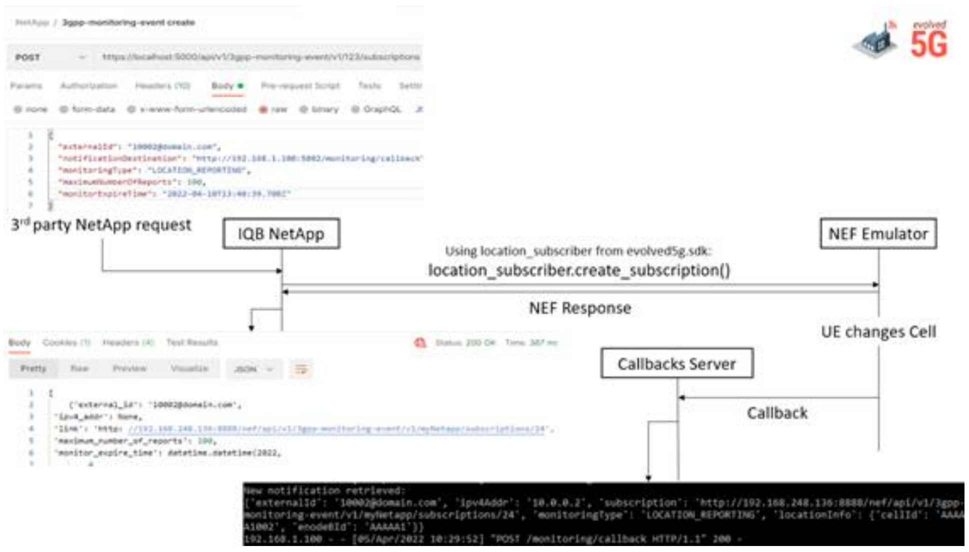
```
{
  "username": "sampleuser",
  "password": "test"
}
```

```
kc_oidc.token(data['username'], data['password'])
```

```
{
    "access_token":
        "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA
        eyJleHAiOjE2NDkxNDE5NzgsImlhdCI6MTY0OTE0MTY3OCwi
```

Figure 29: ID Management and Access Control NetApp Architecture

Every endpoint uses a wrapper function from the functools library to authorize the 3rd party to consume each endpoint. Upon de-authentication of the 3rd party either by using the /logout endpoint or due to misuse of the NetApp, the OAuth2.0 Token is removed from the session and therefore the 3rd Party fails to be authorized.
A pre-defined scenario was tested to check whether the 3rd party would be denied access. Then, the authenticated 3rd party will be authorized to consume NEF APIs. Finally, the 3rd party is de-authenticated using the /logout endpoint or by misuse of the NetApp (i.e., call of non-existing endpoint or non-existing method), and then is denied the consumption of NEF APIs



```
https://localhost:5000/api/v1/3gpp-monitoring-event/v1/123/subscriptions
```

```
{
  "externalId": "10002@domain.com",
  "notificationDestination": "http://192.168.1.100:5002/monitoring/callback",
  "monitoringType": "LOCATION_REPORTING",
  "maximumNumberOfReports": 100,
  "monitorExpireTime": "2022-04-10T13:40:39.708Z"
}
```

```
location_subscriber.create_subscription()
```

```
[
    {'external_id': '10002@domain.com',
  'ipv4_addr': None,
  'link': 'http: //192.168.248.136:8888/nef/api/v1/3gpp-monitoring-event/v1/myNetapp/subscriptions/24',
  'maximum_number_of_reports': 100,
  'monitor_expire_time': datetime.datetime(2022,
```

```
New notification retrieved:
{'externalId': '10002@domain.com', 'ipv4Addr': '10.0.0.2', 'subscription': 'http://192.168.248.136:8888/nef/api/v1/3gpp-
monitoring-event/v1/myNetapp/subscriptions/24', 'monitoringType': 'LOCATION_REPORTING', 'locationInfo': {'cellId': 'AAAA
A1002', 'enodeBId': 'AAAAA1'}}
192.168.1.100 - - [05/Apr/2022 10:29:52] "POST /monitoring/callback HTTP/1.1" 200 -
```

Figure 30 Workflow describing the testing scenario

3.3.2.5 NetApp's Development tools

The NetApp was originally developed and tested on a Windows 10 Host, while the NEF Emulator was hosted inside an Ubuntu 20.04 Virtual Machine. The NetApp has been fully transported to a docker container. The Evolved5G SDK is used to make requests towards the NEF Emulator from the NetApp. Docker Toolbox is used to run the NetApp container as well as the Keycloak container. For communication between the NetApp and Keycloak, the python library keycloak-client is used, while the Evolved5G SDK is used for communication between the NetApp and the NEF Emulator. The code of the NetApp can be found in the following repository: https://github.com/EVOLVED-5G/IQB-NetApp/tree/evolved5g

3.3.3 5G SIEM NetApp

3.3.3.1 Use case short description

In an industrial environment the department that is in charge of the IT and network systems faces the problem of unifying the various existing IP networks with a 5G NPN that supports mobile devices of the employees and 5G-enabled devices in the production line. The main scope is to interface with the core network of the 5G network and embed the information exchanged with 5G to the factory's security information and event management system (SIEM).

3.3.3.2 Current limitations of the use case

As the concept of Industry 4.0 evolves, apart from the IP network, industries will progressively establish small or large-scale 5G Non-Public Networks (NPN) to their premises to exploit the advanced capabilities of 5G technology (low-latency, high throughput etc.) for a set of their equipment. As such, the implementation of enhanced mechanisms to manage and ensure security in this industrial ecosystem is mandatory. For IP networks, SIEM systems offer security protection by performing real-time monitoring and analysis of events as well as tracking and logging of security data for compliance or auditing purposes. However, in 5G networks, security management is handled by the 5G Core Network. Therefore, today, in a unified industrial network (IP and 5G NPN), security information systems have no monitoring and control capabilities for industrial devices that use 5G access.

3.3.3.3 Added value of 5G

Extending a SIEM system with 5G capabilities enhances the platform by offering access to 5G security information, such as real-time monitoring and updates on the security status of the 5G NPN devices. As a result, the security administrator of an Industry 4.0 environment can have a clearer and more complete picture of the underlying industrial network. FOGUS, with the development of its NetApp, aims to bridge the communication gap between SIEM and 5G NPN devices.

3.3.3.4 Detailed architecture

FOGUS NetApp is a containerized application, following the stand-alone model defined in the EVOLVED-5G project, and resides between the vertical application (Alienvault OSSIM) and 5G Network (NEF Emulator). All the components (OSSIM – FOGUS NetApp – NEF Emulator) are deployed locally on our premises, each one on a separate host, but under the same network. The code is uploaded on the project's Github repository [https://github.com/EVOLVED-5G/FogusNetApp/tree/evolved5g] and a detailed architecture of the NetApp is depicted in Figure 31.

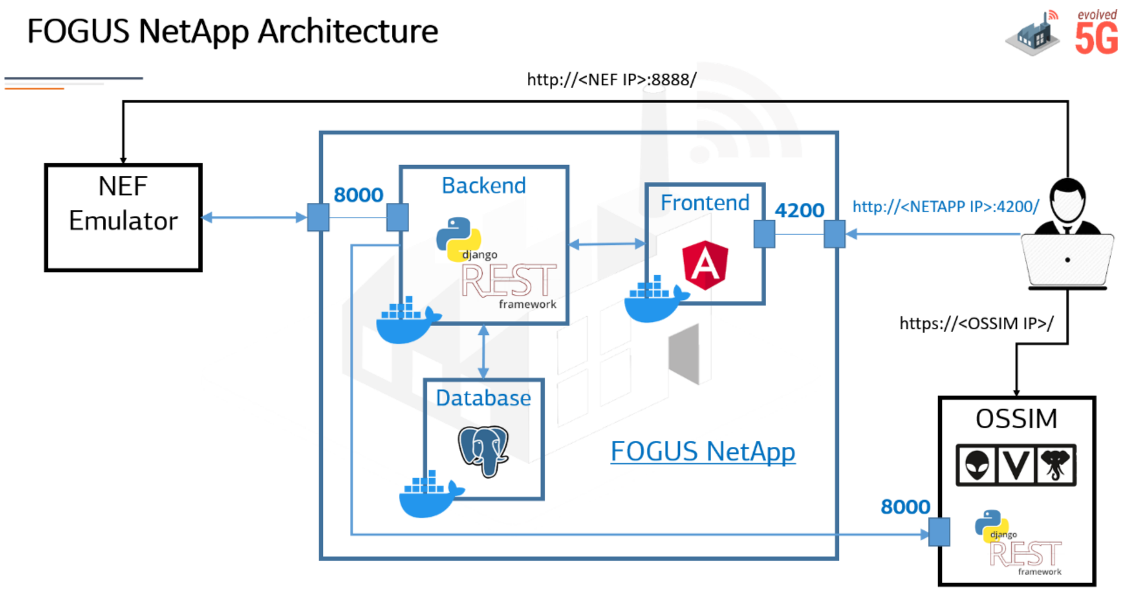
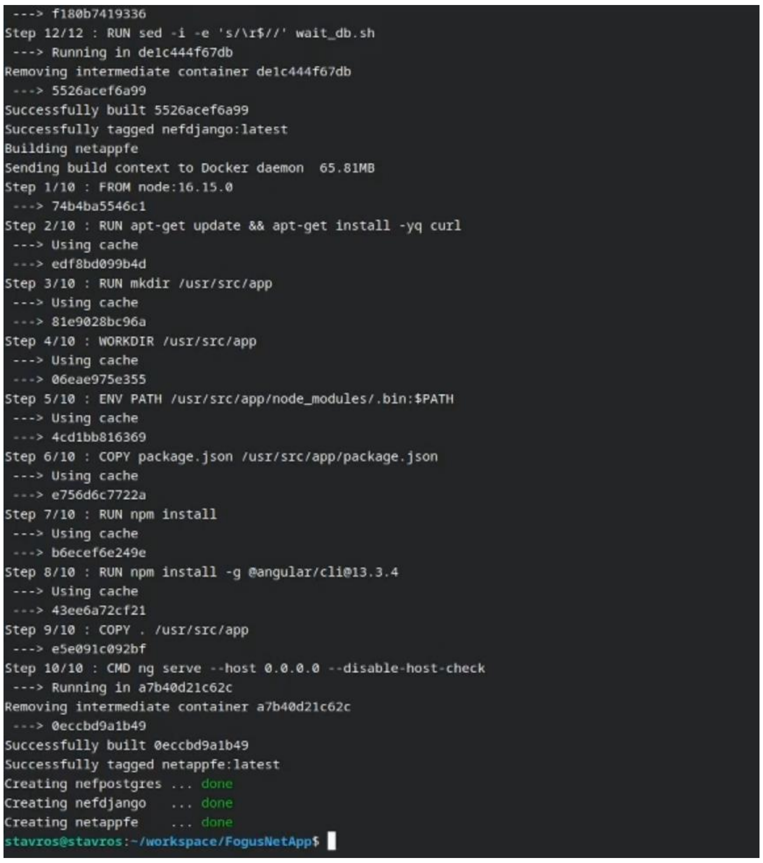


Figure 31. Fogus NetApp architecture

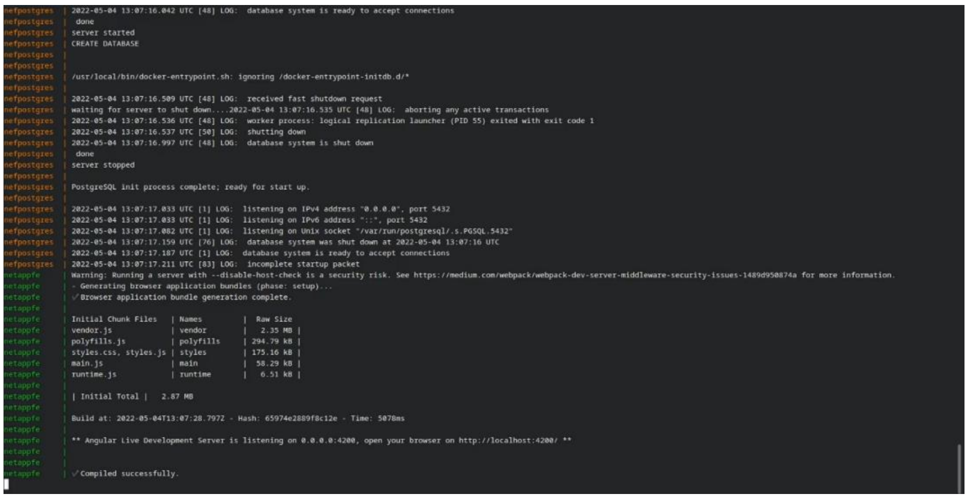
The NetApp consists of 3 services, each one deployed as a separate docker container:

- Frontend: A webpage, built on Angular framework, that enables the user to create subscriptions for the NEF APIs, as well as monitor NEF callbacks.
- Backend: A Python application, built on Django framework, implementing communication with the external components (NEF Emulator and OSSIM).
- Database: A Postgres database, that stores all data exchanged with NEF Emulator and OSSIM.



```
 ---> f180b7419336
Step 12/12 : RUN sed -i -e 's/\r$//' wait_db.sh
 ---> Running in de1c444f67db
Removing intermediate container de1c444f67db
 ---> 5526acef6a99
Successfully built 5526acef6a99
Successfully tagged nefdjango:latest
Building netappfe
Sending build context to Docker daemon  65.81MB
Step 1/10 : FROM node:16.15.0
 ---> 74b4ba5546c1
Step 2/10 : RUN apt-get update && apt-get install -yq curl
 ---> Using cache
 ---> edf8bd099b4d
Step 3/10 : RUN mkdir /usr/src/app
 ---> Using cache
 ---> 81e9028bc96a
Step 4/10 : WORKDIR /usr/src/app
 ---> Using cache
 ---> 06eae975e355
Step 5/10 : ENV PATH /usr/src/app/node_modules/.bin:$PATH
 ---> Using cache
 ---> 4cd1bb816369
Step 6/10 : COPY package.json /usr/src/app/package.json
 ---> Using cache
 ---> e756d6c7722a
Step 7/10 : RUN npm install
 ---> Using cache
 ---> b6ecef6e249e
Step 8/10 : RUN npm install -g @angular/cli@13.3.4
 ---> Using cache
 ---> 43ee6a72cf21
Step 9/10 : COPY . /usr/src/app
 ---> e5e091c092bf
Step 10/10 : CMD ng serve --host 0.0.0.0 --disable-host-check
 ---> Running in a7b40d21c62c
Removing intermediate container a7b40d21c62c
 ---> 0eccbd9a1b49
Successfully built 0eccbd9a1b49
Successfully tagged netappfe:latest
Creating nefpostgres ... done
Creating nefdjango   ... done
Creating netappfe    ... done
stavros@stavros:~/workspace/FogusNetApp$ 
```

Figure 32. NetApp deployed with docker-compose



```
nefpostgres  | 2022-05-04 13:07:16.042 UTC [48] LOG:  database system is ready to accept connections
nefpostgres  |  done
nefpostgres  | server started
nefpostgres  | CREATE DATABASE
nefpostgres  |
nefpostgres  |
nefpostgres  | /usr/local/bin/docker-entrypoint.sh: ignoring /docker-entrypoint-initdb.d/*
nefpostgres  |
nefpostgres  | 2022-05-04 13:07:16.509 UTC [48] LOG:  received fast shutdown request
nefpostgres  | waiting for server to shut down....2022-05-04 13:07:16.535 UTC [48] LOG:  aborting any active transactions
nefpostgres  | 2022-05-04 13:07:16.536 UTC [48] LOG:  worker process: logical replication launcher (PID 55) exited with exit code 1
nefpostgres  | 2022-05-04 13:07:16.537 UTC [50] LOG:  shutting down
nefpostgres  | 2022-05-04 13:07:16.997 UTC [48] LOG:  database system is shut down
nefpostgres  |  done
nefpostgres  | server stopped
nefpostgres  |
nefpostgres  | PostgreSQL init process complete; ready for start up.
nefpostgres  |
nefpostgres  | 2022-05-04 13:07:17.033 UTC [1] LOG:  listening on IPv4 address "0.0.0.0", port 5432
nefpostgres  | 2022-05-04 13:07:17.033 UTC [1] LOG:  listening on IPv6 address "::", port 5432
nefpostgres  | 2022-05-04 13:07:17.082 UTC [1] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
nefpostgres  | 2022-05-04 13:07:17.159 UTC [76] LOG:  database system was shut down at 2022-05-04 13:07:16 UTC
nefpostgres  | 2022-05-04 13:07:17.187 UTC [1] LOG:  database system is ready to accept connections
nefpostgres  | 2022-05-04 13:07:17.211 UTC [83] LOG:  incomplete startup packet
netappfe     | Warning: Running a server with --disable-host-check is a security risk. See https://medium.com/webpack/webpack-dev-server-middleware-security-issues-1489d950874a for more information.
netappfe     | - Generating browser application bundles (phase: setup)...
netappfe     | ✓ Browser application bundle generation complete.
netappfe     |
netappfe     | Initial Chunk Files   | Names         |  Raw Size
netappfe     | vendor.js             | vendor        |   2.35 MB |
netappfe     | polyfills.js          | polyfills     | 294.79 kB |
netappfe     | styles.css, styles.js | styles        | 175.16 kB |
netappfe     | main.js               | main          |  58.29 kB |
netappfe     | runtime.js            | runtime       |   6.51 kB |
netappfe     |
netappfe     | | Initial Total |   2.87 MB
netappfe     |
netappfe     | Build at: 2022-05-04T13:07:28.797Z - Hash: 65974e2889f8c12e - Time: 5078ms
netappfe     |
netappfe     | ** Angular Live Development Server is listening on 0.0.0.0:4200, open your browser on http://localhost:4200/ **
netappfe     |
netappfe     |
netappfe     | ✓ Compiled successfully.
```

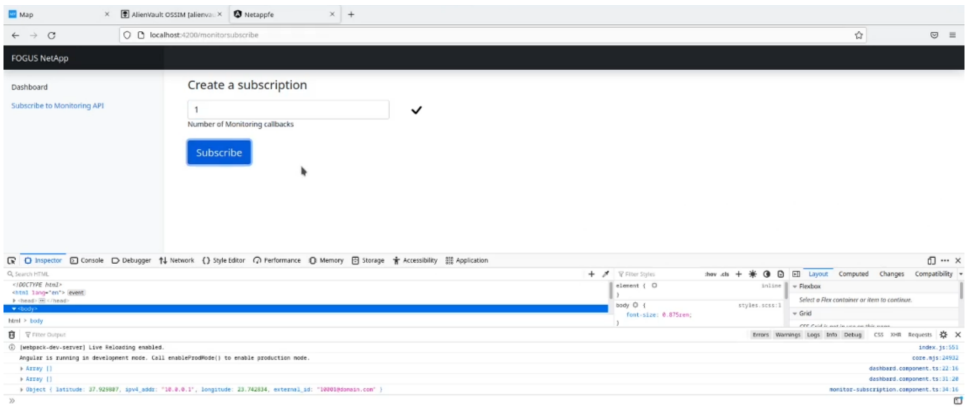
Figure 33. NetApp logs on running phase

The core functionality of the NetApp lies on the backend container. Using the Django framework and its Django-Rest-Framework (DRF) library, a set of REST APIs is implemented and exposed. These APIs are used by:

- The user, through the frontend web page, in order to communicate with NEF Emulator (e.g., make a NEF API subscription)
- NEF Emulator, in order to send periodic or non-periodic callbacks
- OSSIM, in order to send alerts or reports on the underlying 5G NPN devices monitored

The detailed steps of how FOGUS NetApp operates are described below:

1. The user creates a predefined scenario (cells and UEs) on the NEF Emulator
2. Then, the user accesses the web page of the NetApp and creates a subscription on a NEF API, which in the background makes a REST API call to backend container.
3. Backend performs the corresponding NEF API call (using SDK tools)
4. NEF Emulator sends its response(s) back to the NetApp
5. Backend stores the NEF response(s) to Postgres database and feeds both the frontend and the vApp with the latest data updates. NetApp updates UEs status on OSSIM by calling another Django REST API, implemented on OSSIM site.



Create a subscription

1

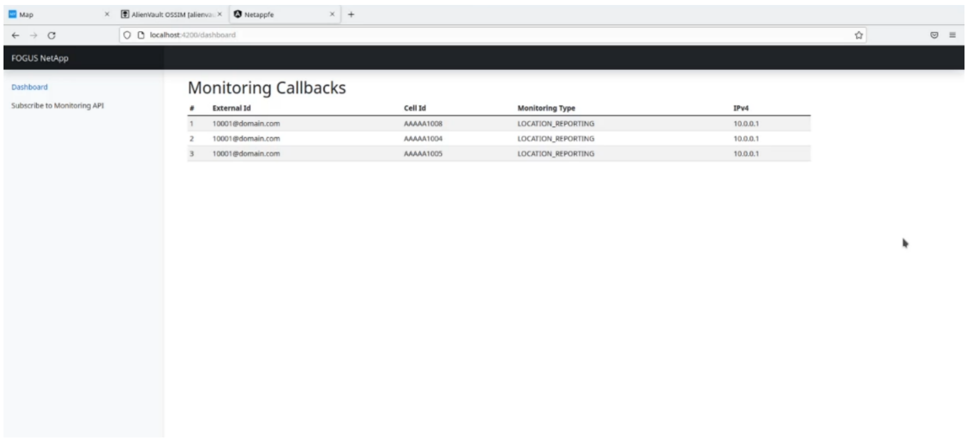
Number of Monitoring callbacks

Subscribe

```
<!DOCTYPE html>
<html lang="en"> event
▸ <head> ··· </head>
▾ <body>
html > body

element {                inline
}
body {                   styles.scss:1
  font-size: 0.875rem;
}

▾ Flexbox
Select a Flex container or item to continue.
▾ Grid
```

```
ⓘ [webpack-dev-server] Live Reloading enabled.                                                          index.js:551
Angular is running in development mode. Call enableProdMode() to enable production mode.                core.mjs:24932
▸ Array []                                                                                   dashboard.component.ts:22:16
▸ Array []                                                                                   dashboard.component.ts:31:20
▸ Object { latitude: 37.929887, ipv4_addr: "10.0.0.1", longitude: 23.742834, external_id: "10001@domain.com" }   monitor-subscription.component.ts:34:16
```

Figure 34. Create one-time monitoring subscription from NetApp



Monitoring Callbacks

#	External Id	Cell Id	Monitoring Type	IPv4
1	10001@domain.com	AAAAA1008	LOCATION_REPORTING	10.0.0.1
2	10001@domain.com	AAAAA1004	LOCATION_REPORTING	10.0.0.1
3	10001@domain.com	AAAAA1005	LOCATION_REPORTING	10.0.0.1

Figure 35. Periodic call-backs from Monitoring Event API appearing on NetApp

3.3.3.5 NetApp's Development tools

FOGUS NetApp is a totally containerized application, using Docker containers, and is developed in Angular, Python and Postgres. NetApp communicates with NEF Emulator using the EVOLVED-5G SDK tools, and with OSSIM through a set of exposed REST APIs implemented on the vApp side, using Python language, and more specifically Django framework.

3.4 PILLAR 4: AGILITY IN THE PRODUCTION LINE INFRASTRUCTURE

3.4.1 5G Teleoperation NetApp

3.4.1.1 Use case short description

- Secured Teleoperation experience over 5G with additional functionalities based on the QoS API

The use case focuses on the QoS API to make the robot teleoperation process adapt in function of the quality of the 5G Network. It will communicate with the vertical application through Robot Operating System ROS topic.

3.4.1.2 Current limitations of the use case

When Teleoperating from a long distance or from poor quality Wi-Fi the Teleoperation experience is hard to handle especially when wanting to grasp objects due to the latency. This latency also impacts the delay from the force torque sensor of the end-effector to the end-user haptic device, this delay could lead to not taking/placing well the object or not detecting collisions with the environment soon enough. Finally, most means of connections are vulnerable to being hacked.

3.4.1.3 Added value of 5G

The Teleoperation of TIAGo robot requires to transmit at high-speed video streams and readings about the sensors and the feedbacks. The connection also needs to be stable without a lot of jitters. And the connection also needs to be secured between the user and the robot. Thanks to the 5G capabilities we would know when the quality of connection required can be guaranteed and when not, this will allow our vertical application to be more robust and have several states in function of the 5G QoS. A direct secured 5G connection will also enable safe teleoperation.

3.4.1.4 Detailed architecture

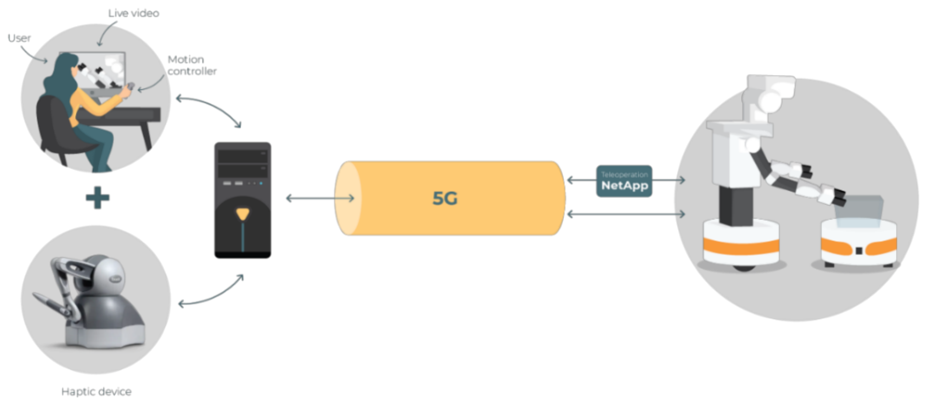
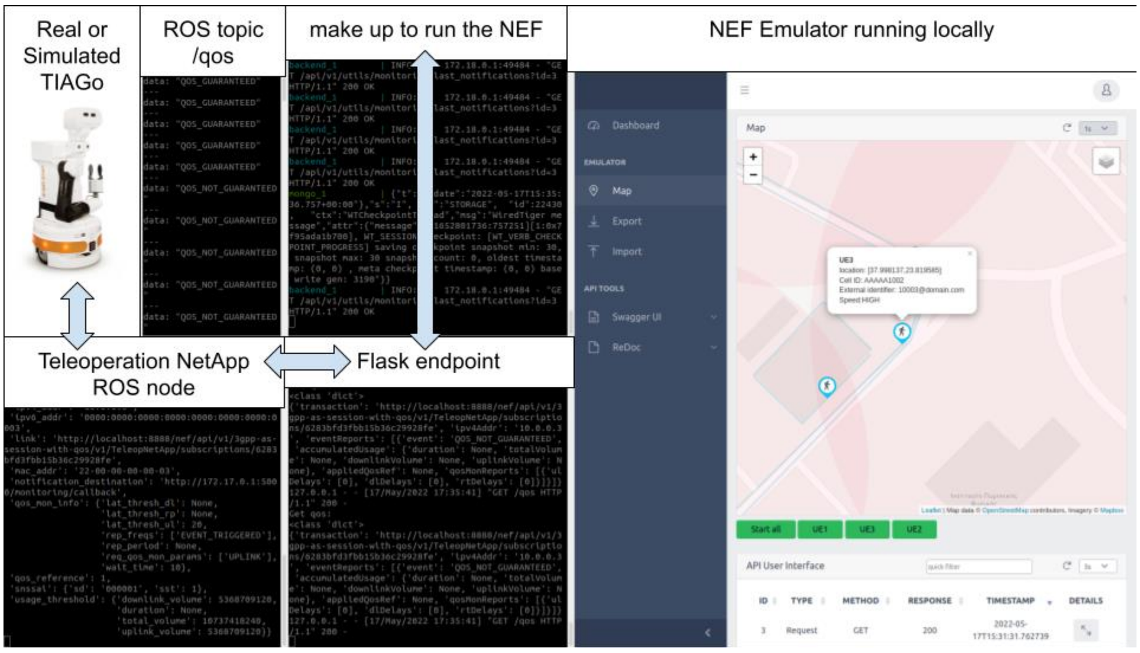


Figure 36. Teleoperation use case architecture

The Teleoperation NetApp is a python ROS node which communicates through the SDK to retrieve the QoS status. The QoS permits us to know if the quality is guaranteed and stable and when it is not. This information can then be used by our vertical application in order to stop the Teleoperation when the QoS is not guaranteed. It will always guarantee a smooth Teleoperation experience with low-latency force feedback that will allow the user to feel the touch of obstacles. If there is too much latency it can delay the force feedback and the operator video of the robot and therefore damage furniture or objects. That is why this new capability of the 5G infrastructure will improve our vertical applications and permit us to deploy the robot in more locations. First, there is the NEF emulator that is running locally on the machine that simulates the 5G QoS API and interactions with the UEs. In this figure there are 2 UEs on the same cell. For now, the state “QoS is not guaranteed” is triggered when at least 2 UEs are in the same cell. This will work differently on the real 5G but the output messages will be similar. In the middle, there is flask running the endpoint based on the configuration of the endpoint inside the TeleopNetApp.
Finally, there is the ROS node that is creating a valid subscription to the flask endpoint and publishing a rostopic /qos that takes the value “QOS_GUARANTEED” or “QOS_NOT_GUARANTEED” depending on where the UE is located. In the figure below, A successful interaction between the EU 3 from the NEF Emulator and the robot is presented.



```
data: "QOS_GUARANTEED"
---
data: "QOS_GUARANTEED"
---
data: "QOS_GUARANTEED"
---
data: "QOS_GUARANTEED"
---
data: "QOS_GUARANTEED"
---
data: "QOS_NOT_GUARANTEED
"
---
data: "QOS_NOT_GUARANTEED
"
---
data: "QOS_NOT_GUARANTEED
"
---
data: "QOS_NOT_GUARANTEED
"
---
data: "QOS_NOT_GUARANTEED
"
```

```
backend_1           | INFO:      172.18.0.1:49484 - "GE
T /api/v1/utils/monitori    last_notifications?id=3
HTTP/1.1" 200 OK
backend_1           | INFO:      172.18.0.1:49484 - "GE
T /api/v1/utils/monitori    last_notifications?id=3
HTTP/1.1" 200 OK
backend_1           | INFO:      172.18.0.1:49484 - "GE
T /api/v1/utils/monitori    last_notifications?id=3
HTTP/1.1" 200 OK
backend_1           | INFO:      172.18.0.1:49484 - "GE
T /api/v1/utils/monitori    last_notifications?id=3
HTTP/1.1" 200 OK
mongo_1             | {"t":    date":"2022-05-17T15:35:
36.757+00:00"},"s":"I",      ":"STORAGE",  "id":22430
,   "ctx":"WTCheckpointT    ad","msg":"WiredTiger me
ssage","attr":{"message"    1652801736:757251][1:0x7
f95ada1b700], WT_SESSION    eckpoint: [WT_VERB_CHECK
POINT_PROGRESS] saving c    kpoint snapshot min: 30,
 snapshot max: 30 snapsh    count: 0, oldest timesta
mp: (0, 0) , meta checkp    t timestamp: (0, 0) base
 write gen: 3198"}}
backend_1           | INFO:      172.18.0.1:49484 - "GE
T /api/v1/utils/monitori    last_notifications?id=3
HTTP/1.1" 200 OK
```

```
 'ipv6_addr': '0000:0000:0000:0000:0000:0000:0000:0
003',
 'link': 'http://localhost:8888/nef/api/v1/3gpp-as-
session-with-qos/v1/TeleopNetApp/subscriptions/6283
bfd3fbb15b36c29928fe',
 'mac_addr': '22-00-00-00-00-03',
 'notification_destination': 'http://172.17.0.1:500
0/monitoring/callback',
 'qos_mon_info': {'lat_thresh_dl': None,
                  'lat_thresh_rp': None,
                  'lat_thresh_ul': 20,
                  'rep_freqs': ['EVENT_TRIGGERED'],
                  'rep_period': None,
                  'req_qos_mon_params': ['UPLINK'],
                  'wait_time': 10},
 'qos_reference': 1,
 'snssai': {'sd': '000001', 'sst': 1},
 'usage_threshold': {'downlink_volume': 5368709120,
                     'duration': None,
                     'total_volume': 10737418240,
                     'uplink_volume': 5368709120}}
```

```
<class 'dict'>
{'transaction': 'http://localhost:8888/nef/api/v1/3
gpp-as-session-with-qos/v1/TeleopNetApp/subscriptio
ns/6283bfd3fbb15b36c29928fe', 'ipv4Addr': '10.0.0.3
', 'eventReports': [{'event': 'QOS_NOT_GUARANTEED',
 'accumulatedUsage': {'duration': None, 'totalVolum
e': None, 'downlinkVolume': None, 'uplinkVolume': N
one}, 'appliedQosRef': None, 'qosMonReports': [{'ul
Delays': [0], 'dlDelays': [0], 'rtDelays': [0]}]}]}
127.0.0.1 - - [17/May/2022 17:35:41] "GET /qos HTTP
/1.1" 200 -
Get qos:
<class 'dict'>
{'transaction': 'http://localhost:8888/nef/api/v1/3
gpp-as-session-with-qos/v1/TeleopNetApp/subscriptio
ns/6283bfd3fbb15b36c29928fe', 'ipv4Addr': '10.0.0.3
', 'eventReports': [{'event': 'QOS_NOT_GUARANTEED',
 'accumulatedUsage': {'duration': None, 'totalVolum
e': None, 'downlinkVolume': None, 'uplinkVolume': N
one}, 'appliedQosRef': None, 'qosMonReports': [{'ul
Delays': [0], 'dlDelays': [0], 'rtDelays': [0]}]}]}
127.0.0.1 - - [17/May/2022 17:35:41] "GET /qos HTTP
/1.1" 200 -
```

UE3
location: [37.998137,23.819585]
Cell ID: AAAAA1002
External identifier: 10003@domain.com
Speed HIGH

ID	TYPE	METHOD	RESPONSE	TIMESTAMP	DETAILS
3	Request	GET	200	2022-05-17T15:31:31.762739	

Figure 37. Teleoperation NetApp interactions

3.4.1.5 NetApp's Development tools

Main computer - industrial PC:

- Ubuntu 18.04 or Ubuntu 20.04 with a docker in 18.04
- ROS middleware
- ROS packages
- PAL packages
- Wi-Fi

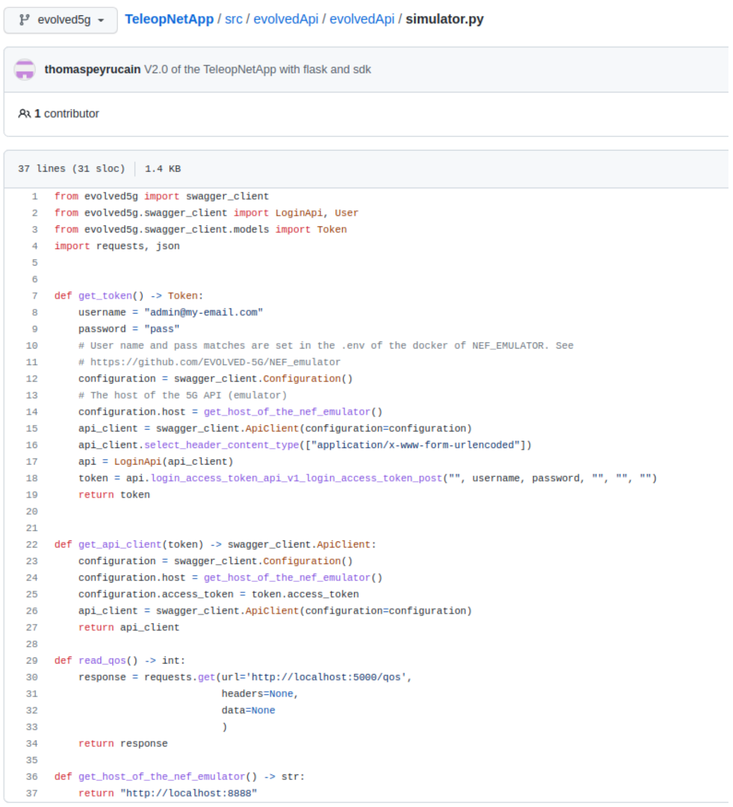
Tiago robot

- Ubuntu 18.04
- ROS middleware
- PAL packages
- ROS packages
- Sensors
- Wi-Fi

NEF Emulator was hosted inside an Ubuntu 20.04 Machine and the Teleoperation NetApp was installed and tested locally also on the same machine. To retrieve information from the emulator flask endpoints were used. For now, the QoS API is providing only 2 information: QoS guaranteed or not guaranteed. New states will be available in the future and will permit to adapt the speed of the arm of the robot depending on the state of the connection. Robot Operating System (ROS) [21] is used in TIAGo platform, and the NetApp is sending a ROS standard message to a ROS topic inside the robot in order for the Vertical APP to adapt. The TeleopNetApp is stored in GitHub under the evolved-5G branch: https://github.com/EVOLVED-5G/TeleopNetApp

It is composed of two specific parts:

- The first part is the evolvedApi that contains the functions to create a valid request to the NEF Emulator using the 5G SDK and the flask definition of the endpoint.



evolved5g TeleopNetApp / src / evolvedApi / evolvedApi / **simulator.py**

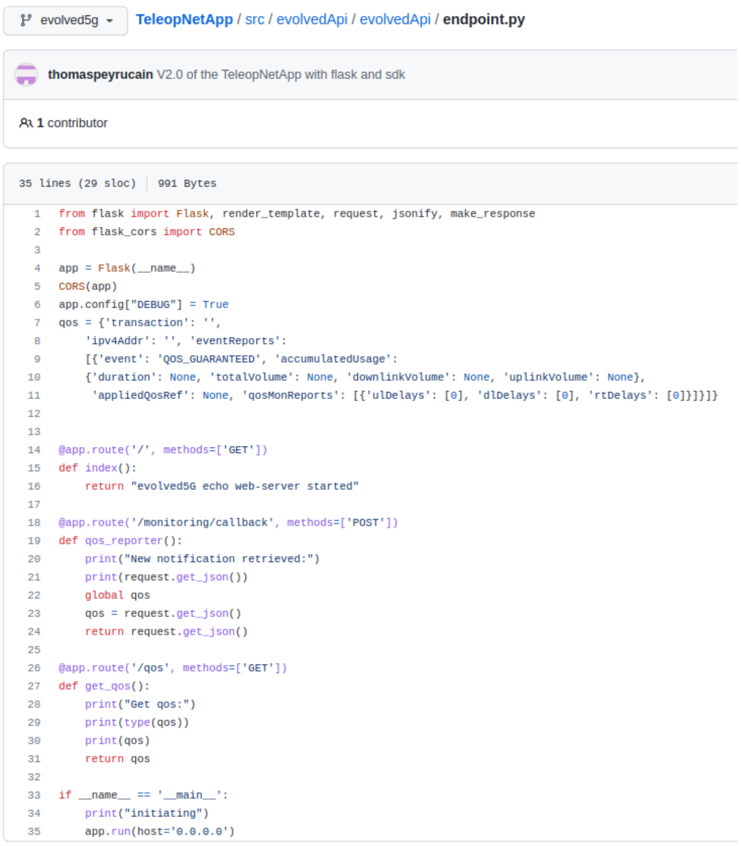
thomaspeyrucain V2.0 of the TeleopNetApp with flask and sdk

1 contributor

37 lines (31 sloc) | 1.4 KB

```python
from evolved5g import swagger_client
from evolved5g.swagger_client import LoginApi, User
from evolved5g.swagger_client.models import Token
import requests, json


def get_token() -> Token:
    username = "admin@my-email.com"
    password = "pass"
    # User name and pass matches are set in the .env of the docker of NEF_EMULATOR. See
    # https://github.com/EVOLVED-5G/NEF_emulator
    configuration = swagger_client.Configuration()
    # The host of the 5G API (emulator)
    configuration.host = get_host_of_the_nef_emulator()
    api_client = swagger_client.ApiClient(configuration=configuration)
    api_client.select_header_content_type(["application/x-www-form-urlencoded"])
    api = LoginApi(api_client)
    token = api.login_access_token_api_v1_login_access_token_post("", username, password, "", "", "")
    return token


def get_api_client(token) -> swagger_client.ApiClient:
    configuration = swagger_client.Configuration()
    configuration.host = get_host_of_the_nef_emulator()
    configuration.access_token = token.access_token
    api_client = swagger_client.ApiClient(configuration=configuration)
    return api_client

def read_qos() -> int:
    response = requests.get(url='http://localhost:5000/qos',
                            headers=None,
                            data=None
                            )
    return response

def get_host_of_the_nef_emulator() -> str:
    return "http://localhost:8888"
```

Figure 38 functions to create a valid request to the NEF Emulator1



evolved5g TeleopNetApp / src / evolvedApi / evolvedApi / **endpoint.py**

thomaspeyrucain V2.0 of the TeleopNetApp with flask and sdk

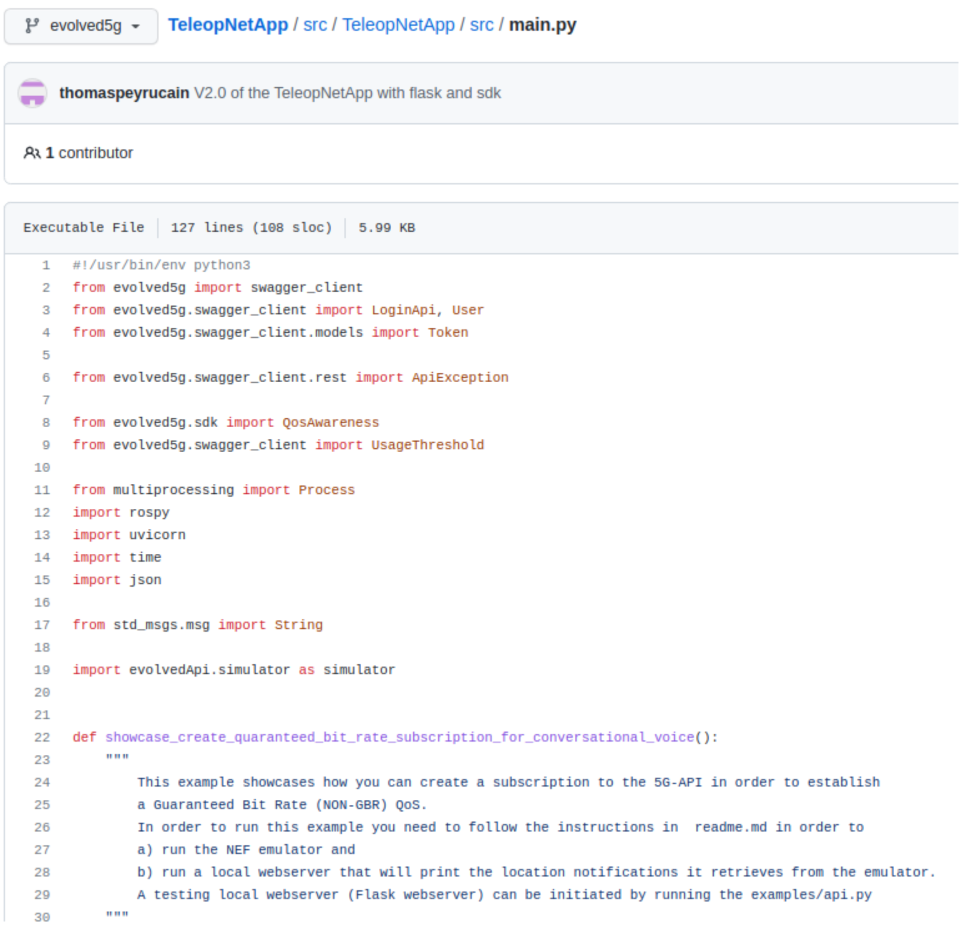
1 contributor

35 lines (29 sloc) | 991 Bytes

```python
from flask import Flask, render_template, request, jsonify, make_response
from flask_cors import CORS

app = Flask(__name__)
CORS(app)
app.config["DEBUG"] = True
qos = {'transaction': '',
    'ipv4Addr': '', 'eventReports':
    [{'event': 'QOS_GUARANTEED', 'accumulatedUsage':
    {'duration': None, 'totalVolume': None, 'downlinkVolume': None, 'uplinkVolume': None},
     'appliedQosRef': None, 'qosMonReports': [{'ulDelays': [0], 'dlDelays': [0], 'rtDelays': [0]}]}]}


@app.route('/', methods=['GET'])
def index():
    return "evolved5G echo web-server started"

@app.route('/monitoring/callback', methods=['POST'])
def qos_reporter():
    print("New notification retrieved:")
    print(request.get_json())
    global qos
    qos = request.get_json()
    return request.get_json()

@app.route('/qos', methods=['GET'])
def get_qos():
    print("Get qos:")
    print(type(qos))
    print(qos)
    return qos

if __name__ == '__main__':
    print("initiating")
    app.run(host='0.0.0.0')
```

Figure 39 Functions to create a valid request to the NEF Emulator2

- The second part is the TeleopNetApp ROS node, it contains the python code that uses the evolvedApi helpers and endpoint to access the information of the QoS API from the NEF Emulator and output it in a valid ROS message



evolved5g TeleopNetApp / src / TeleopNetApp / src / **main.py**

thomaspeyrucain V2.0 of the TeleopNetApp with flask and sdk

1 contributor

Executable File | 127 lines (108 sloc) | 5.99 KB

```python
#!/usr/bin/env python3
from evolved5g import swagger_client
from evolved5g.swagger_client import LoginApi, User
from evolved5g.swagger_client.models import Token

from evolved5g.swagger_client.rest import ApiException

from evolved5g.sdk import QosAwareness
from evolved5g.swagger_client import UsageThreshold

from multiprocessing import Process
import rospy
import uvicorn
import time
import json

from std_msgs.msg import String

import evolvedApi.simulator as simulator


def showcase_create_quaranteed_bit_rate_subscription_for_conversational_voice():
    """
        This example showcases how you can create a subscription to the 5G-API in order to establish
        a Guaranteed Bit Rate (NON-GBR) QoS.
        In order to run this example you need to follow the instructions in  readme.md in order to
        a) run the NEF emulator and
        b) run a local webserver that will print the location notifications it retrieves from the emulator.
        A testing local webserver (Flask webserver) can be initiated by running the examples/api.py
    """
```

Figure 40 Python code that uses the EvolvedAPI helpers and endpoint

3.4.2 Localisation NetApp

3.4.2.1 Use case short description

In D2.1 and D4.1 a detailed explanation of the use case can be found. To sum up the main objective is to design and implement a Localization NetApp to have a global positioning reference in indoor environments. Therefore, we are:

- Leveraging 5G to know which bands and antennas we are connected with and retrieve a position estimation.
- Provide the information by ensuring proper quality of service in the network in terms of latency, jitter, and security.

3.4.2.2 Current limitations of the use case

Robotic applications are time-sensitive. Since most of the commands need to be real-time and have low latency. It is also hard to scale up robotic industrial applications around the world and not only in private environments.

If a robot is turned off and moved around, when it is started again at another place it is hard to find its position in the already created map environment with the embedded sensors. From the point of view of the robot, it could be in a different place. And the more complex the map is the harder it gets.

3.4.2.3 Added value of 5G

As the 5G networks evolves and the industry begins to adopt public and private 5G networks a door opens regarding data-intensive applications with low latency.
Leveraging technologies like 5G will help to scale up robotic industrial applications around the world and not only in private environments.
5G provides several technologies which directly address problems of robotics:

- Related to network edge computation it can provide extremely efficient latencies to perform critical time-based robot control tasks.
- Enough bandwidth to externalize on-board computation to external computation.
- Robotics fleets are more scalable
- Network slicing to provide the necessary network quality of service for time-critical applications with high computation.
- As a use case: Roboticists can now address known problems such as global indoor positioning since the 5G core can provide this information through NetApps. In fact, 5G can provide an extended sensory system for every robotic system.
- The orchestration of a fleet of mobile robots to enable the centralized control of autonomous mobile robots to perform logistics tasks in the agile production line.

3.4.2.4 Detailed architecture

The localization NetApp is a Python application in a Docker container which communicates with the NEF-emulator through an SDK to retrieve the Cell ID of the antennas which a robot is connected. The Cell ID is linked to a geographical position to estimate the robot position in the global frame. The NetApp also communicates with the robots to send the Cell ID, therefore their geographical location. The communication between the vertical App which consist of the Unmanned Life Central Control Platform (ULCCP) and the Robots are done using Real Time Publish Subscribe (RTPS) protocol from ROS2 as described in the following image.

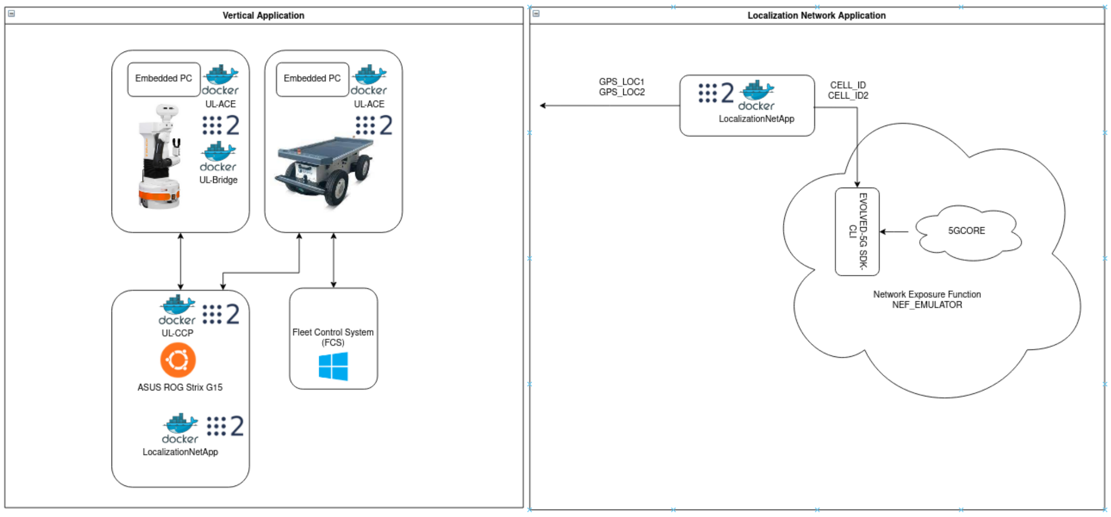


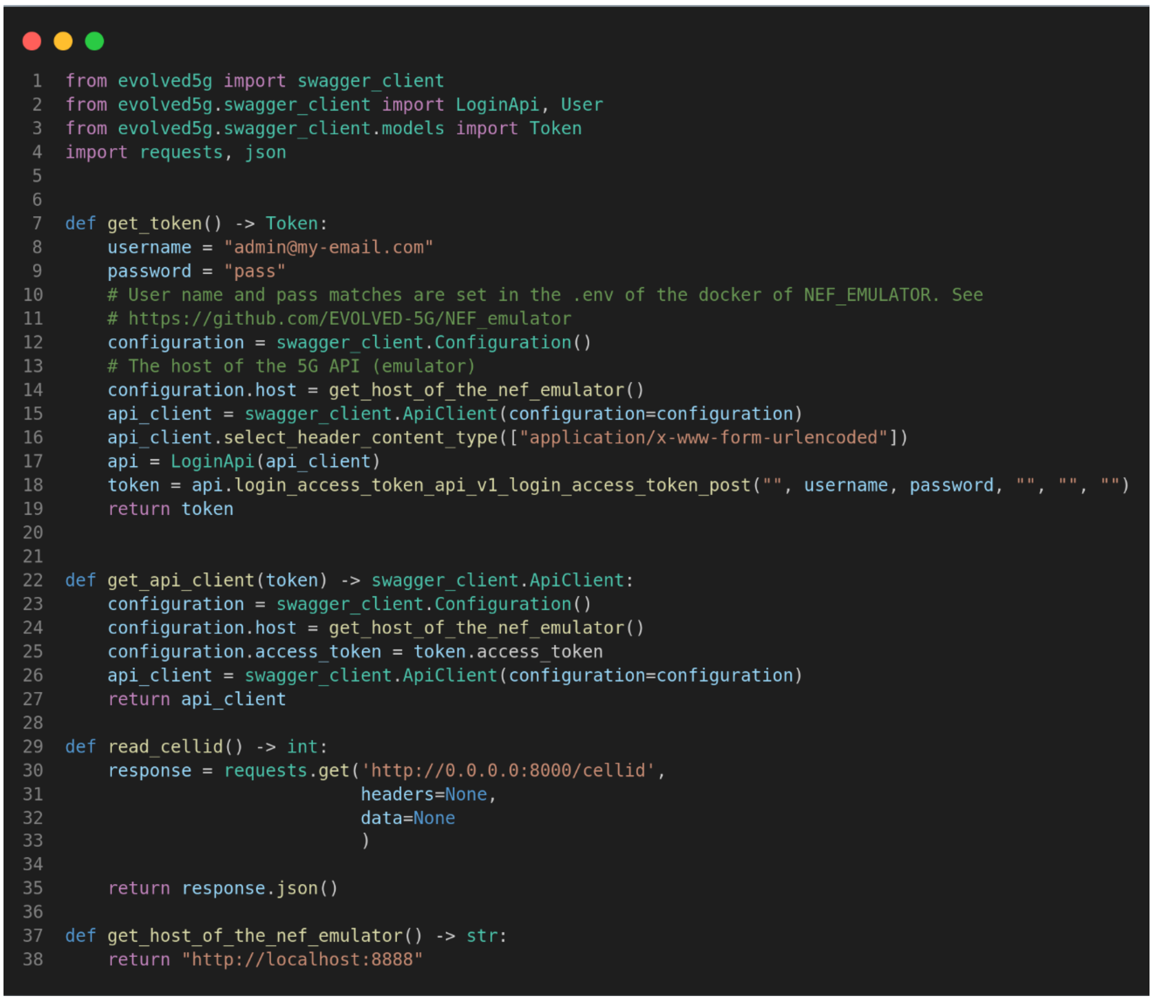
Figure 41. Localization use case architecture

3.4.2.5 NetApp's Development tools

The NetApp communicates to the NEF Emulator through the SDK. To extend functionalities in the NetApp a developer can add different endpoints to retrieve specific information required to compute more accurate geographical positioning by fusing the information. For now, we can retrieve the Cell ID and send it back to the vApp and react to it by navigating to various locations. Keep in mind that right now we are tightened to use Python, but the NetApp can be extended to use multiple languages such as C++, Java, or Rust thanks that the information can be trespassed using ROS2. Therefore, if it is required to use external libraries in a specific language to fuse information this is possible with the current architecture.

The development of the Localization NetApp is stored in the 5GEvolved GitHub repository. https://github.com/EVOLVED-5G/LocalizationNetApp.git in branch evolved5g as aligned with the consortium. The repository contains two packages:

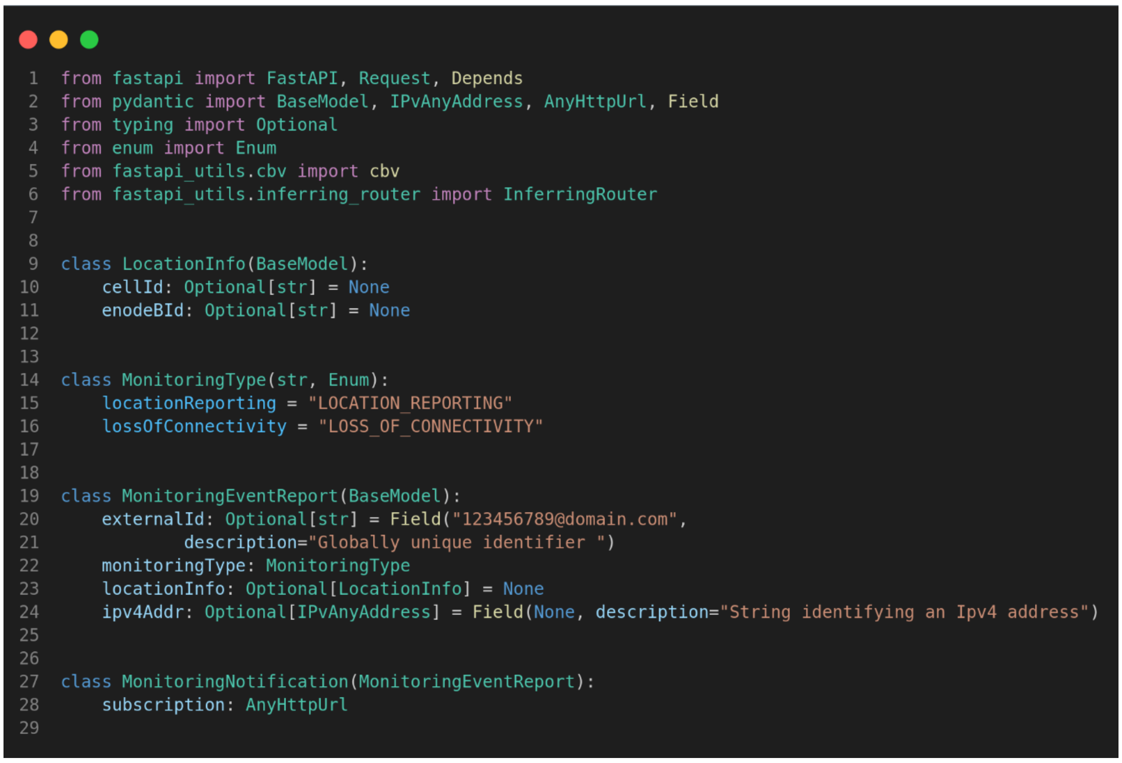
- evolvedAPI: Contains the python scripts to communicate with the NEF Emulator through the SDK. It mainly describes two scripts.
    - simulator.py: Is in charge to communicate to the Emulator and login to the GUI. Get the token and send request to the cellid endpoint.



```python
from evolved5g import swagger_client
from evolved5g.swagger_client import LoginApi, User
from evolved5g.swagger_client.models import Token
import requests, json


def get_token() -> Token:
    username = "admin@my-email.com"
    password = "pass"
    # User name and pass matches are set in the .env of the docker of NEF_EMULATOR. See
    # https://github.com/EVOLVED-5G/NEF_emulator
    configuration = swagger_client.Configuration()
    # The host of the 5G API (emulator)
    configuration.host = get_host_of_the_nef_emulator()
    api_client = swagger_client.ApiClient(configuration=configuration)
    api_client.select_header_content_type(["application/x-www-form-urlencoded"])
    api = LoginApi(api_client)
    token = api.login_access_token_api_v1_login_access_token_post("", username, password, "", "", "")
    return token


def get_api_client(token) -> swagger_client.ApiClient:
    configuration = swagger_client.Configuration()
    configuration.host = get_host_of_the_nef_emulator()
    configuration.access_token = token.access_token
    api_client = swagger_client.ApiClient(configuration=configuration)
    return api_client

def read_cellid() -> int:
    response = requests.get('http://0.0.0.0:8000/cellid',
                            headers=None,
                            data=None
                            )

    return response.json()

def get_host_of_the_nef_emulator() -> str:
    return "http://localhost:8888"
```

Figure 42 Request to the cell ID endpoint.

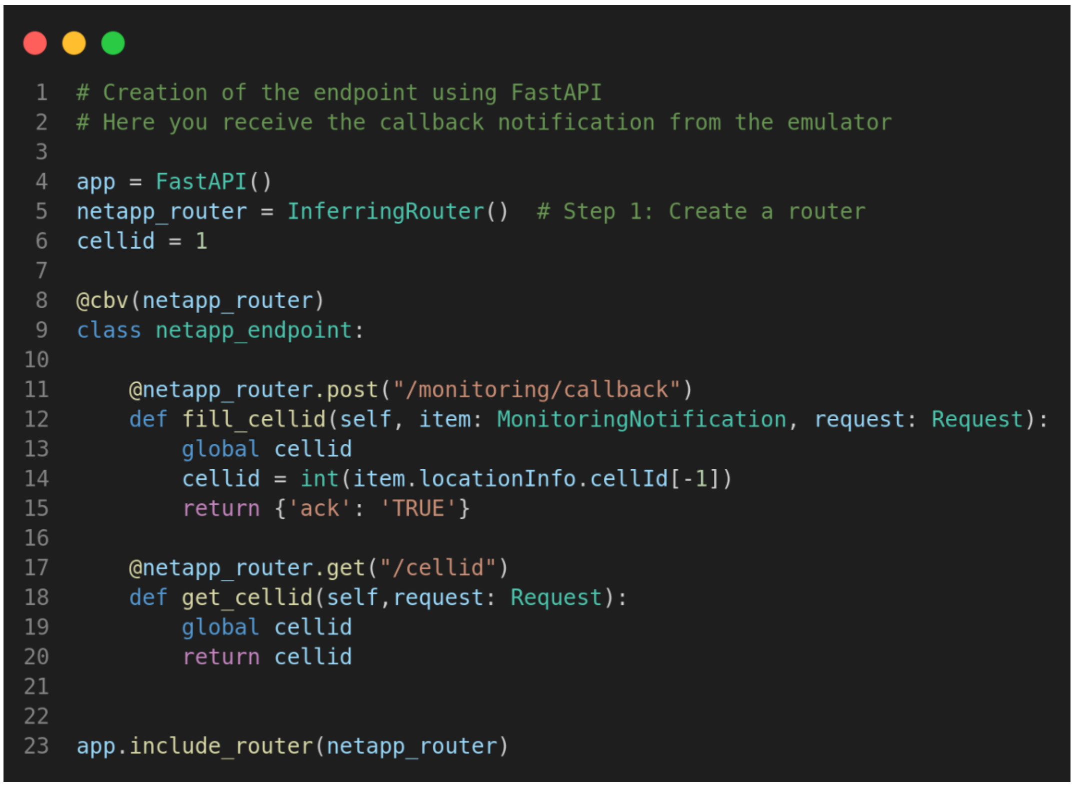
-
    - *endpoint.py:* Is in charge to validate to monitor and validate the data received from the Emulator. We instantiate different classes to monitor Location information, events, and notifications



```python
from fastapi import FastAPI, Request, Depends
from pydantic import BaseModel, IPvAnyAddress, AnyHttpUrl, Field
from typing import Optional
from enum import Enum
from fastapi_utils.cbv import cbv
from fastapi_utils.inferring_router import InferringRouter


class LocationInfo(BaseModel):
    cellId: Optional[str] = None
    enodeBId: Optional[str] = None


class MonitoringType(str, Enum):
    locationReporting = "LOCATION_REPORTING"
    lossOfConnectivity = "LOSS_OF_CONNECTIVITY"


class MonitoringEventReport(BaseModel):
    externalId: Optional[str] = Field("123456789@domain.com",
            description="Globally unique identifier ")
    monitoringType: MonitoringType
    locationInfo: Optional[LocationInfo] = None
    ipv4Addr: Optional[IPvAnyAddress] = Field(None, description="String identifying an Ipv4 address")


class MonitoringNotification(MonitoringEventReport):
    subscription: AnyHttpUrl

```

Figure 43 Instantiate different classes to monitor Location information, events

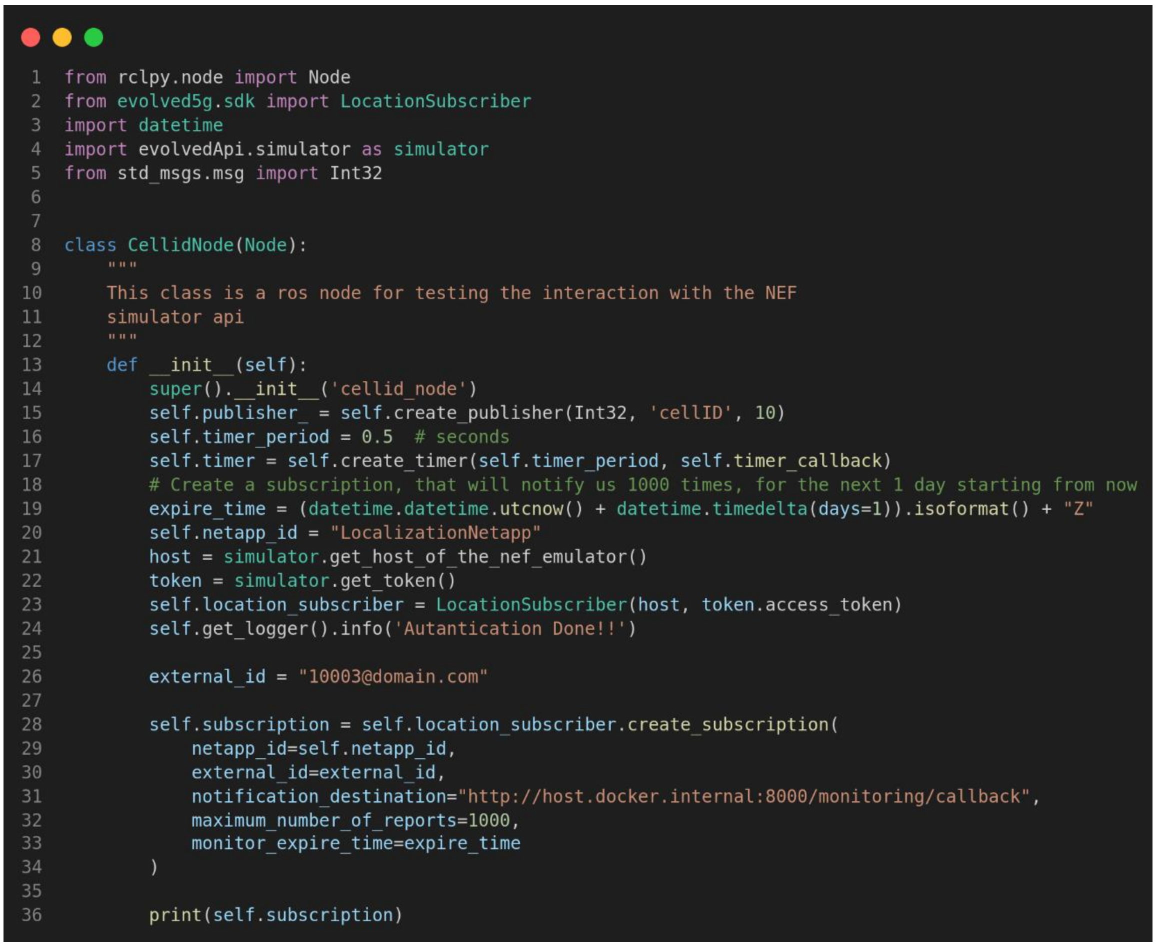
Then we create endpoints using FastAPI to monitor a call-back. The reason of using FastAPI is because is more fast than proposed libraries such as Flask. Since we are reading the localization at higher rate, we decided to choose that library. Then using REST API with the end points received we subscribe to the call-back and to the cell ID.



```python
# Creation of the endpoint using FastAPI
# Here you receive the callback notification from the emulator

app = FastAPI()
netapp_router = InferringRouter()  # Step 1: Create a router
cellid = 1

@cbv(netapp_router)
class netapp_endpoint:

    @netapp_router.post("/monitoring/callback")
    def fill_cellid(self, item: MonitoringNotification, request: Request):
        global cellid
        cellid = int(item.locationInfo.cellId[-1])
        return {'ack': 'TRUE'}

    @netapp_router.get("/cellid")
    def get_cellid(self,request: Request):
        global cellid
        return cellid


app.include_router(netapp_router)
```

Figure 44 Creation of endpoints using FastAPI to monitor a call-back

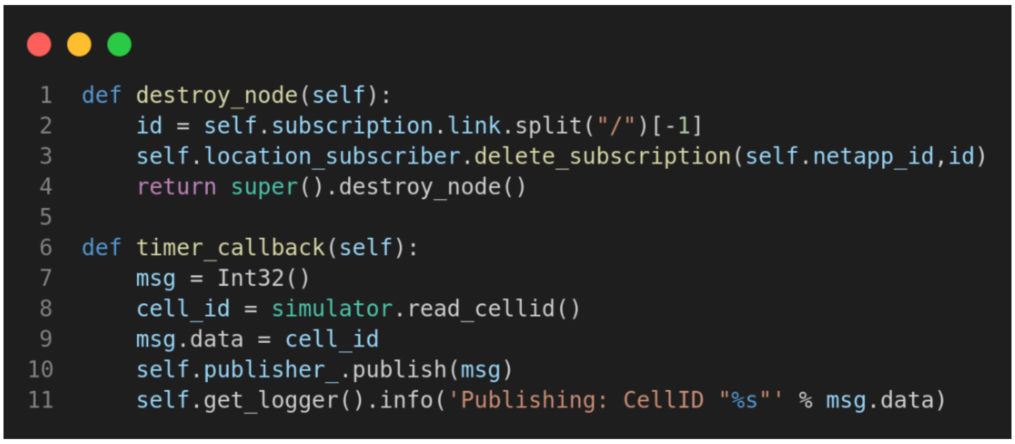
- Localization_netapp:
    - *cell_node.py:* Is in charge to read the Location data given from REST API calls and creates a ROS2 node to transfer this data using RTPS (Real-time Publisher Subscriber) Protocol which is used by an underlying middleware in ROS2 called DDS (Data Distribution Service)



```python
from rclpy.node import Node
from evolved5g.sdk import LocationSubscriber
import datetime
import evolvedApi.simulator as simulator
from std_msgs.msg import Int32


class CellidNode(Node):
    """
    This class is a ros node for testing the interaction with the NEF
    simulator api
    """
    def __init__(self):
        super().__init__('cellid_node')
        self.publisher_ = self.create_publisher(Int32, 'cellID', 10)
        self.timer_period = 0.5  # seconds
        self.timer = self.create_timer(self.timer_period, self.timer_callback)
        # Create a subscription, that will notify us 1000 times, for the next 1 day starting from now
        expire_time = (datetime.datetime.utcnow() + datetime.timedelta(days=1)).isoformat() + "Z"
        self.netapp_id = "LocalizationNetapp"
        host = simulator.get_host_of_the_nef_emulator()
        token = simulator.get_token()
        self.location_subscriber = LocationSubscriber(host, token.access_token)
        self.get_logger().info('Autantication Done!!')

        external_id = "10003@domain.com"

        self.subscription = self.location_subscriber.create_subscription(
            netapp_id=self.netapp_id,
            external_id=external_id,
            notification_destination="http://host.docker.internal:8000/monitoring/callback",
            maximum_number_of_reports=1000,
            monitor_expire_time=expire_time
        )

        print(self.subscription)
```

Figure 45 Code for cell_node.py

Once the constructor is defined, we create a method to destroy the node and to publish the cell ID in RTPS format at a certain rate.

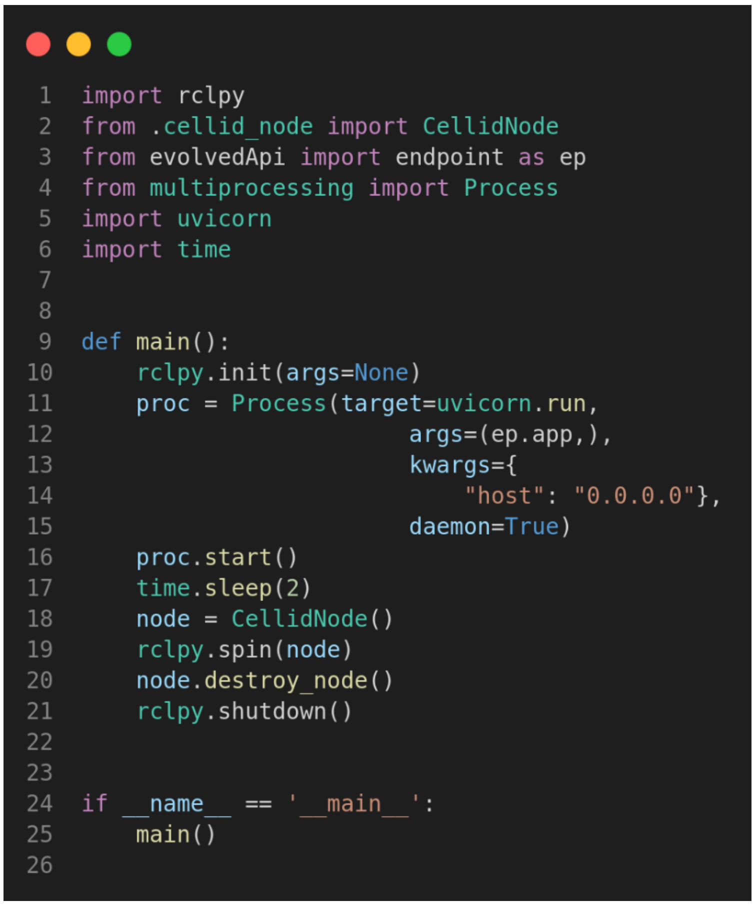


```python
def destroy_node(self):
    id = self.subscription.link.split("/")[-1]
    self.location_subscriber.delete_subscription(self.netapp_id,id)
    return super().destroy_node()

def timer_callback(self):
    msg = Int32()
    cell_id = simulator.read_cellid()
    msg.data = cell_id
    self.publisher_.publish(msg)
    self.get_logger().info('Publishing: CellID "%s"' % msg.data)
```

Figure 46 destroy the node and to publish the cell ID in RTPS

- *main.py:* Is in charge to instantiate the ROS2 node and the uvicorn server from FastAPI to make REST calls.

To ease the deployment, a docker file is created so as to virtualize the NetApp and be easily deployed in different infrastructure since it has all the dependencies required. Once then the Cell ID data is being published using RTPS we proceed to verify its functionality with the vApp.



```python
import rclpy
from .cellid_node import CellidNode
from evolvedApi import endpoint as ep
from multiprocessing import Process
import uvicorn
import time


def main():
    rclpy.init(args=None)
    proc = Process(target=uvicorn.run,
                   args=(ep.app,),
                   kwargs={
                       "host": "0.0.0.0"},
                   daemon=True)
    proc.start()
    time.sleep(2)
    node = CellidNode()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()


if __name__ == '__main__':
    main()
```

Figure 47 Code for the main.py

4 CONCLUSION AND NEXT STEPS FOR NETAPP DEVELOPMENT

The work presented in this deliverable describes in details the full-scale integration of the SDK leading to the Release A of the NetApps for the four pillars in the EVOLVED-5G context.
Task 4.2-4.5 are driving the current deliverable and one of the primary purposes of the specific taskscduring the lifetime of the project, is to tightly interact with Task 4.1, responsible for the design of the APIs, and at a higher level with WP2 and WP3, in order to consistently collect the relevant architectural information from the former while the latter will facilitate the development of Release B of the NetApps, by providing the updates on the specific tools supporting the development process.

In the first development cycle (M5-M14), the applications supporting the Industry 4.0 pillars were identified and an early set of NetApps per pillar were developed and presented in D4.1.
To that end several structuring sections from D4.1 have been updated. More specifically Section 2 provides details on the tools within the workspace environment that are utilised towards the development process. All information related to the Release A of the NetApps, including use cases' brief description, added value of the 5G, detailed architecture giving a breakdown of major components and their role, as well as the NetApps' specific development tools, are presented in section 3.
In the second iteration cycle (M15-M32) the full set of NetApps will be developed and the final prototypes (Release B) will be reported in the corresponding deliverables for each pillar, namely D4.4 "NetApps for Interactions of Employees and Machines", D4.5 "NetApps for FoF Operations", D4.6 "NetApps for Security Guarantees and Risk Analysis" and D4.7 "NetApps for Production Line Infrastructures", all due at M32 of the project.

5 REFERENCES

[1] “EVOLVED-5G, "D3.1 Implementations and integrations towards EVOLVED-5G framework realisation,",” [Online]. Available: https://evolved-5g.eu/wp-content/uploads/2022/01/EVOLVED-5G-D3.1-v1.0.pdf.

[2] “EVOLVED-5G, "Deliverable D2.1 "Overall Framework Design and Industry 4.0 requirements",” [Online]. Available: https://evolved-5g.eu/wp-content/uploads/2021/11/EVOLVED-5G-D2.1_v1.4.pdf..

[3] [Online]. Available: https://evolved5g-cli.readthedocs.io/en/latest/ .

[4] [Online]. Available: https://evolved-5g.eu/wp-content/uploads/2022/03/EVOLVED-5G-D4.1_v2.0-final.pdf .

[5] [Online]. Available: https://github.com/EVOLVED-5G/SDK-CLI .

[6] [Online]. Available: https://github.com/EVOLVED-5G/NEF_emulator.

[7] [Online]. Available: https://github.com/EVOLVED-5G/CAPIF_API_Services.

[8] [Online]. Available: https://www.sonarqube.org/..

[9] [Online]. Available: https://aquasecurity.github.io/trivy/v0.28.1/.

[10] [Online]. Available: https://robotframework.org/.

[11] [Online]. Available: https://www.jetbrains.com/pycharm/.

[12] [Online]. Available: https://pypi.org/project/jsonpickle/ .

[13] [Online]. Available: https://botman.io/.

[14] [Online]. Available: https://www.gmi-aero.com/en/anita-ez-2-zone-hot-bonder-3.html .

[15] [Online]. Available: https://www.embarcadero.com/.

[16] [Online]. Available: https://www.uco.es/investiga/grupos/ava/node/26.

[17] [Online]. Available: https://flask.palletsprojects.com .

[18] [Online]. Available: https://www.sqlalchemy.org/ .

[19] [Online]. Available: https://oauth.net/2/ .

[20] [Online]. Available: https://www.keycloak.org/.

[21] [Online]. Available: https://www.ros.org/.