D5.1 Initial deployment and Configuration of enablers at 6G sites

Document Summary Information

Project Identifier	HORIZON-JU-SNS-2022. Project 101096954		
Project name	6G-BRICKS: Building Reusable testbed Infrastructures for validating Cloud-to-device breaKthrough technologieS		
Acronym	6G-BRICKS		
Start Date	January 1, 2023	**End Date**	December 31, 2025
Project URL	www.6g-bricks.eu		
Deliverable	D5.1 – Initial deployment and Configuration of enablers at 6G sites		
Work Package	5		
Contractual due date	01/03/2023	**Actual submission date**	
Type		**Dissemination Level**	PU - Public
Lead Beneficiary	KUL		
Responsible Author	Franco Minucci (KUL)		
Contributors	Jan Fromme (NI), Karim Boutiba (EUR), Abdelkader Mekrache (EUR), Charalambos Papadakis (ICOM), Marinela Mertiri (ICOM), Nikolaos Psaromanolakis (ICOM), Arifur Raman (ISW)		
Peer reviewer(s)	Carolina Fernández (i2CAT)		

Funded by the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

Disclaimer

The content of this document reflects only the author's view. Neither the European Commission nor the INEA are responsible for any use that may be made of the information it contains.

While the information contained in the documents is believed to be accurate, the authors(s) or any other participant in the 6G-BRICKS consortium make no warranty of any kind with regard to this material including, but not limited to the implied warranties of merchantability and fitness for a particular purpose.

Neither the 6G-BRICKS Consortium nor any of its members, their officers, employees or agents shall be responsible or liable in negligence or otherwise howsoever in respect of any inaccuracy or omission herein.

Without derogating from the generality of the foregoing neither the 6G-BRICKS Consortium nor any of its members, their officers, employees or agents shall be liable for any direct or indirect or consequential loss or damage caused by or arising from any information advice or inaccuracy or omission herein.

Copyright message

© 6G-BRICKS Consortium. This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both. Reproduction is authorised provided the source is acknowledged.

Table of Contents

1 Executive Summary 8
2 Introduction 9
3 Deployment and Configuration of enablers at the KUL facility 11
3.1 KUL testbed overview 11
3.2 Dataset Collection at KUL Site 13
3.2.1 Data collection API 14
3.2.2 Hardware and Software Requirements 15
3.2.3 Software Architecture 15
3.2.4 Potential extensions to the framework and future work 17
4 6G-BRICKS Enablers to be deployed at KUL Testbed 18
4.1 Open-RU platform deployment 18
4.2 Cell-Free Massive MIMO enablers 19
4.2.1 Deployment of Cell-free mMIMO enabled RAN 19
4.2.2 Deployment of Cell-free mMIMO Near-RT RIC 20
4.3 Deployment of the Cognitive Compute Continuum enabler 21
5 Deployment and Configuration of 6G-BRICKS enablers at EUR facility 28
5.1 EUR testbed overview 28
5.2 RIS control framework deployment and configuration 29
5.3 DMO's fault management deployment and configuration 31
5.3.1 Abnormal Traffic Detector (ATD) 32
5.3.2 LLM-enabled Trial Fault explanator (LLM-TFE) 33
6 Conclusions 35
7 References 36

List of Figures

Figure 1: Remote access configuration for KUL Testbed. 11

Figure 2: On the left, the picture shows the massive MIMO KUL setup, the MOCAP system, and the cartesian positioners. On the right, it shows the rack with the new controller and two2 X410 USRPs with "SuperIQ" capabilities, USRPs x410and the HP Enterprise Edge server, which is part of the compute cluster and committed to 6G-BRICKS 12

Figure 3: Accurate positioning infrastructure for Joint Communication and Sensing Research 13

Figure 4: General architecture of the data recording framework 14

Figure 5: Example recording setup with one transmitter and one receiver (both USRP X410) 15

Figure 6: Software architecture of the RF Data Recording Framework 15

Figure 7: Example configuration file in YAML and JSON format 16

Figure 8: Basic building blocks of a "Category A" Open-RU 18

Figure 9: Basic architecture of the Open-RU with cabled RU-DU synchronization 18

Figure 10: Deployment schema of the O-DU and O-CU components 20

Figure 11 Deployment schema of Cell-free mMIMO Near-RT RIC on a docker container 21

© 6G-BRICKS 2023

Figure 12: 6GBRICKS enablers deployment at EUR Facility 28
Figure 13: DMO fault management deployment at the EUR facility 31
Figure 14: OAI CN NWDAF deployment to support ATD. 32

List of Tables

Table 1 – Adherence to GA Deliverable & Tasks Descriptions 9
Table 2 - System requirements for RAN deployment 20

Glossary of terms and abbreviations used

Abbreviation / Term	Description
3GPP	3rd Generation Partnership Project
5G-PPP	5G Infrastructure Public Private Partnership
AI	Artificial Intelligence
API	Application programming interface
BSS	Business Support System
CCC	Cognitive Cloud Computing
CCSA	China Communications Standards Association
CDSO	Cross-Domain Service Orchestrator
CEC	Cloud-Edge Continuum
CF	Cell Free
COTS	Commercial Off the Shelf
CPU	Central Processing Unit
CSI	Channel State Information
CU	Centralized Unit
DE	Decision Engine
DMO	Domain Manager Orchestrator
DRL	Distributed Reinforcement Learning
DU	Distributed Unit
ECCM	Edge-Cloud Continuum Manager
eMBB	Enhanced Mobile Broadband
ETSI	European Telecommunications Standards Institute
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
GPSDO	GPS Disciplined Oscillator
GPIO	General Purpose Input Output
ICT	Information and communications technology
IETF	Internet Engineering Task Force
IFFT	Inverse Fast Fourier Transform

© 6G-BRICKS 2023

IPC	Inter Process Communication
I/Q	In-phase and Quadrature
JTAG	Joint Test Action Group
JSON	JavaScript Object Notation
KPI	Key performance Indicator
MANO	Management and Orchestration
MEC	Multi-Access Edge Computing
MHU	Mast Head unit
MDAS	Management Data Analytics Service
MIMO	Multiple-Input and Multiple-Output
ML	Machine Learning
MPM	Module Peripheral Manager
MR	Machine Reasoning
mMIMO	Massive MIMO
mMTC	Massive Machine-Type Communications
M-SDO	Multiple Standards Developing Organization
MOCAP	Motion Capture
NFV	Network Function Virtualisation
NSA	Non-Standalone
NWDAF	Network data analytics function
OAI	OpenAirInterface
OFDM	Orthogonal Frequency-Division Multiplexing
OSS	Operations Support System
OTA	Over The Air
PaaS	Platform as a Service
PPS	Pulse per Second
PTP	Precision Time Protocol
QSFP28	Quad Small Form Factor Pluggable 28Gbps
6G REF-RAN	Real-time Flexible Radio Access Network
RAN	Radio Access Network
REST	Representational State Transfer

© 6G-BRICKS 2023

RF	Radio Frequency
RFNoC	RF Network on Chip
RIC	RAN Intelligent Controller
RIS	Reconfigurable Intelligent Surface
RRM	Radio Resource Management
RTL	Register Transfer Language
RU	Radio Unit
SA	Standalone
SDI	Software Defined Infrastructure
SDN	Software-defined networking
SDR	Software Defined Radio
SE	Spectral Efficiency
SLA	Service Level Agreement
SLO	Service Level Objective
SSH	Secure Shell Protocol
TMF	Tele Management Forum
UE	User Equipment
URLLC	Ultra-Reliable Low-Latency Communications
USRP	Universal Software Radio Peripheral
VIM	Virtualized Infrastructure Manager
VNF	Virtual Network Function
XAI	Explainable AI

© 6G-BRICKS 2023

1 Executive Summary

This document provides a comprehensive overview of the initial deployment of enablers on the Eurecom (EUR) and KU Leuven (KUL) 6G test facilities, detailing their configurations and deployment processes up to low-level details.

First, the KUL facility is described; emphasizing its infrastructure for physical layer testing and the upgrade to a modular and containerized architecture. This deliverable describes in great detail the available infrastructure, focusing on details not yet described in other deliverables, such as the motion capture system, which is essential to the use of this facility. This section also provides basic rules for the deployment of future enablers. The previous infrastructure has been upgraded to a more modular and containerized architecture to accommodate the needs for 6G-BRICKS and future projects. A dedicated subsection focuses on the data collection API, by National Instruments, detailing its architecture, requirements, and how to effectively use it to collect measurements in the KUL testbed.

Besides this, a comprehensive list of enablers deployed (or in deployment) at the KUL facility have its functionality and deployment details detailed; including the Open-RU platform, Cell-Free Massive Multiple-Input and Multiple-Output (MIMO) enablers, and the π-Edge Platform. The OpenRU is an open-source implementation of an O-RAN Radio Unit (RU) and it is provided by National Instruments. The Open-RU platform includes both software and hardware,in the form of Field Programmable Gate Array (FPGA) RTL code, sources. The Cell-Free Massive MIMO DU and RU is provided by IS Wireless while the Compute Continuum is provided by Intracom.

Finally, the EUR facility is documented in detail, highlighting the configuration and setup of the Reconfigurable Intelligent Surface (RIS) framework provided by CEA Leti and EUR. The RIS hardware is provided by CEA Leti and the software is a collaboration between CEA and EUR. We outline the deployment process and configuration parameters essential for performing experiments with the RIS. The control framework of the RIS is thoroughly described. The deployment process and configuration parameters for RIS experiments are outlined, with a detailed description of the RIS control framework. Enablers deployed at the EUR facility, such as the RIS control framework and Domain Manager Orchestrator (DMO)'s fault management systems (including the Abnormal Traffic Detector and Large Language Model (LLM)-enabled Trial Fault Explanator), are enumerated with their functionality and deployment requirements.

All enablers mentioned in the document are accompanied by a detailed description of their functionality and all the relevant deployment and configuration requirements.

© 6G-BRICKS 2023

2 Introduction

This document serves as a comprehensive overview of the initial deployment of enablers on the Eurecom (EUR) and KUL 6G test facilities and their configurations. For each facility, we describe the basic configuration settings of the facility, how they need to be adjusted to accommodate the deployment of each enabler. We also provide detailed insights into the deployment and configuration processes of current enablers and basic rules on how to prepare future enablers for deployment. More specifically, the deliverable structure is as follows:

- In Section 3, we describe the KUL facility, providing an in-depth description of the available hardware and software. Being primarily a physical layer test facility, the KUL testbed provides more infrastructure for pure RF measurements.
- In Section 4, we provide a comprehensive list of enablers deployed (or in deployment) at the KUL facility, including the Open-RU platform, Cell-Free Massive MIMO enablers, and the Compute Continuum.
- Section 5, focuses on the EUR facility and describes the configuration and setup of the Reconfigurable Intelligent Surface (RIS) framework.

All enablers mentioned in the document are accompanied by a description of their functionality and all the details required for their deployment, including Docker files, scripts, YAML or JSON files, and codebooks where appropriate.

Table 1 – Adherence to GA Deliverable & Tasks Descriptions

6G-BRICKS GA Component Title	6G-BRICKS GA Component Outline	Respective Document Chapter(s)	Justification
DELIVERABLE			
D5.1 Initial deployment and Configuration of enablers at 6G sites			
TASKS			
Task 5.1: Deployment, configuration and interfacing of enablers at the KUL site	Bricks identification: decide which bricks are most promising based on Tasks 3.2 and 3.3 Integrating the bricks with the KUL infrastructure Collect relevant datasets covering multiple frequency bands, blocking scenarios, and locations of the antennas.	Chapter 2: Introduction Chapter 3: Dataset collection at KUL Chapter 4: Deployment of enablers at KUL	Delivers a detailed description of dataset collection, deployment and configuration of enablers at KUL in chapters 3 and 4 respectively.
Task 5.2: 6G enablers from T3.1 and T3.4 will be	First, the RIS reflector will be tested and validated with OAI-gNB	Chapter 5: Deployment of enablers at EUR	Delivers a detailed description of enablers deployment and configuration at EUR in chapter 5

integrated to the EUR facility.	Second, the RIS Controller is integrated as a xApp in the RIC Near-time controller Third, the RIS control algorithm developed in T3.4 to validate the control algorithm (including the positioning xApp) is tested and validated		

3 Deployment and Configuration of enablers at the KUL facility

3.1 KUL testbed overview

The infrastructure of the KUL testbed (**Error! Reference source not found.**), as detailed in D2.2 consists of (1) a Compute Cluster, to perform computation and control of experiments, (2) the Access Server, which is the gateway between the testbed and the internet, and (3) the Radio Infrastructure, which consists of a set of Universal Software Radio Peripherals (USRPs), which can operate as Radio Units (RUs). The connectivity between the controller and the RUs is provided by a dedicated high-speed 100GbE fiber link. The testbed is shown in Figure 2.

The Testbed Controller node, which is part of the Cluster, not only provides the core operational aspects of the testbed but also the software and hardware for Machine Learning (ML), and data processing. Users can perform number crunching and Artificial Intelligence (AI) training using an NVIDIA A100 accelerator card, in conjunction with Tensor Flow or PyTorch. Data sets can be stored locally or on open portals such as IEEE Dataport, and are accessible to the public, ensuring transparency and availability. The AI-as-a-Service component can either be used to perform AI tasks online, like running AI algorithms as part of an experiment in conjunction with the radio hardware, or offline to analyze the collected data without moving them.

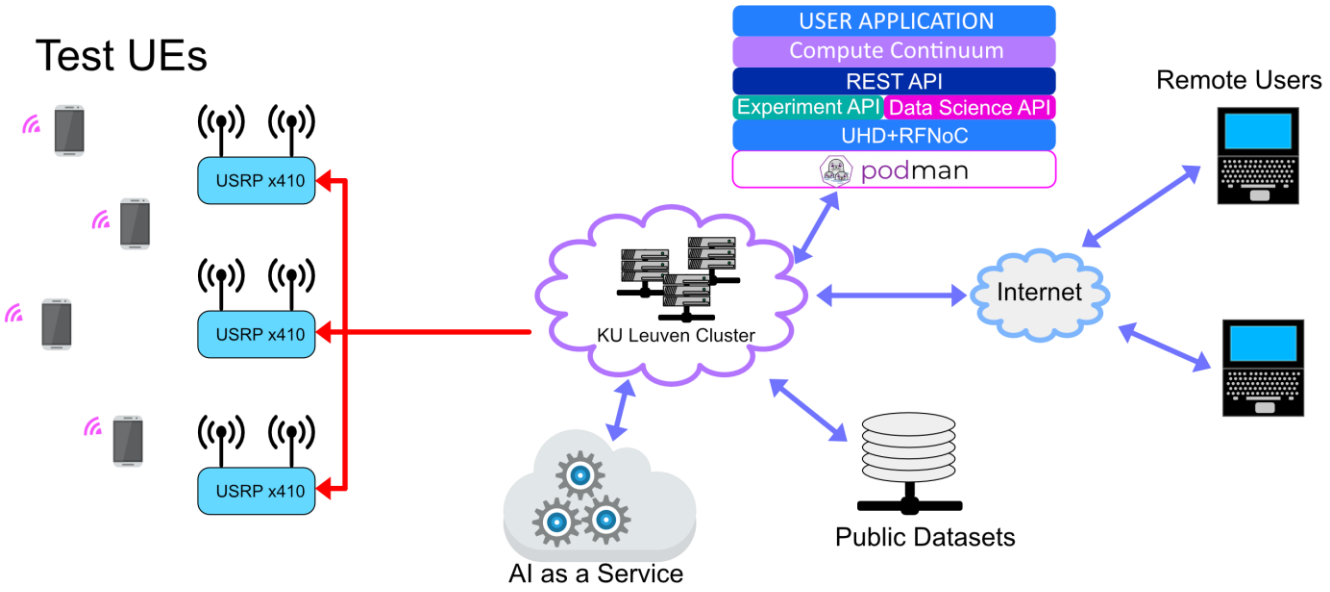

Figure 1: Remote access configuration for KUL Testbed.

The enablers (also called “bricks”) developed in WP3, except for OpenRU, should be deployed on the KUL testbed primarily as containers relying on the Podman infrastructure. This is so because OpenRU, and future similar bricks, include components that require direct access to the hardware, such as FPGA bitstreams and OS drivers. These enablers cannot be containerized because they run directly on the USRP hardware and not on the Testbed Controller. For enablers (or “bricks”) to be deployed and integrated into the KUL platform, they must follow a predefined structure and set of rules:

- Should have its logic self-contained in a single folder.
- They have a single-entry point (a.k.a. a single executable file).

© 6G-BRICKS 2023

- The folder includes a human-readable description or a documentation file in a text format.
- Each brick includes a machine-readable file (or set of files) with the metadata concerning the brick. The metadata are the name of the executable, command line parameters, input and output data, and information regarding the language/runtime needed for it to run.
- There is no specific rule regarding the language in which bricks are written, if the implementer guarantees a way to make them interoperable with other bricks. This can be achieved by having each brick offer a REST (Representational State Transfer) API or offering a specific port for Inter Process Communication (IPC).
- Each brick should be tested and provide a set of unit tests for each function it provides.

Figure 2: On the left, the picture shows the massive MIMO KUL setup, the MOCAP system, and the cartesian positioners. On the right, it shows the rack with the new controller and two2 X410 USRPs with "SuperIQ" capabilities, USRPs x410and the HP Enterprise Edge server, which is part of the compute cluster and committed to 6G-BRICKS

Experimenters can access the testbed via SSH and perform remote experimentation, which consists of four steps:

1. Generation of a container with the required dependencies and working script: this container will have access to a specific folder to store the data files and a "scratch" folder on the local hard drive. The "scratch" folder is not backed up. A basic container is provided to each user as a starting point.
2. Copy of the required files to run the experiment.
3. Execution of the experiment. This might require to physically go to the lab to place antennas in the correct position, prepare the positioner, configure the Motion Capture (MOCAP) system, and any other task which requires manual intervention.
4. Collection of results from the folder and execution of data analysis.

Experiments can be packaged within containers generated by Podman, which is based on Fedora Linux and includes:

a. USRP Hardware Drivers.
b. Software to control the positioner.
c. Python 3.12 including:
    a. PyTorch.
    b. Tensorflow (with CUDA).
    c. UHD (to control USRPs from Python).

© 6G-BRICKS 2023

3.2 Dataset Collection at KUL Site

The testbed infrastructure is mostly focused on three areas of research:

- Distribution of antenna elements and computing power.
- Integration of communication and sensing.
- Native use of AI and data-driven models.

Depending on the use case, data can be collected using different components of the testbed infrastructure each providing different kind of data.

The KUL Distributed Massive MIMO testbed, developed since 2013, has undergone several upgrades allowing for distributed antenna elements [1], extended data logging [2], a positioning system [3], and over-the-air sync for measurements with a drone-mounted UE [4]. [5] provides an overview of its capabilities and dataset, with code available from IEEE Code Ocean [6] for user scheduling methods and MIMO precoders. The testbed's data logging is open for 6G-BRICKS experimenters to obtain new datasets. Additional information is available on the KUL website [7], including hardware and software overviews and older datasets with processing code.

The 6G Real-time Flexible Radio Access Network (6G REF-RAN)[1] testbed is focused on future 6G Networks. It is also mostly dedicated to collecting channel state information and In-phase and Quadrature (I/Q) samples of Radio Frequency (RF) signals indoors. It is tightly coupled with the cartesian robot positioner (Figure 3) and the motion capture system, and it can run AI-in-the-loop experiments. The bandwidth of each radio is 400MHz. However, the antenna array is smaller than for the massive MIMO infrastructure and the hardware is meant to operate indoors.

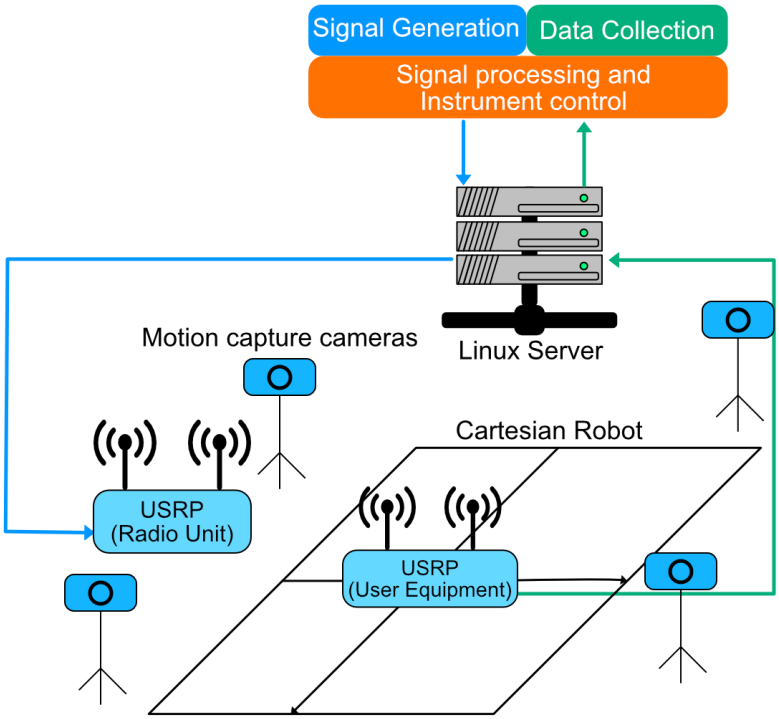

Figure 3: Accurate positioning infrastructure for Joint Communication and Sensing Research

The MOCAP system and the cartesian robot (**Error! Reference source not found.**) are used to collect positional

[1] https://research.kuleuven.be/portal/en/project/3E230023

information coupled with the RF samples. The MOCAP system is synchronized with the same reference clock used by the radios, and it can be used to label positional data. Labelled data is useful for ML algorithms regarding joint communication and sensing and to verify the accuracy of angle of arrival algorithms. The motion capture software can be interfaced with Python, Matlab and C++. The hardware, software, and API to control the motion capture system is provided by Qualisys.

3.2.1 Data collection API

Appropriate datasets are crucial during algorithmic development. Traditional deterministic algorithms need to be validated; while for AI/ML-based models, extensive data is needed to train the algorithms according to the required performance specifications. These data sets can be generated synthetically via simulation, which is usually a good first step to confirm proper functionality. However, real-world data is required to test the algorithms under realistic conditions. For example, theoretical channel models, as being typically used in simulation environments, miss critical characteristics of a real wireless channel and hardware impairments. Relevant datasets in the context of the 6G-BRICKS project should also cover multiple frequency bands, blocking scenarios, a flexible number of receivers and transmitters, and different locations of the antennas, which is not easily achievable via simulation.

To capture these realistic datasets for studying the performance of various multi-band channel estimation, clustering, synchronization and distributed processing methods, a data recording framework based on the USRP Software Defined Radio (SDR) platform will be used and extended towards the specific use cases. A subset of the acquired datasets will be publicly available for researchers outside of the 6G-BRICKS consortium. The public datasets will be shared on platforms such as IEEE Dataport, the KUL Research Data Repository[2], or other similar platforms.

The general architecture of the RF data recording framework can be seen in Figure 4. A central controlling instance (“Linux Server”) is handling the orchestration of the respective recording scenario which can be configured flexibly via configuration files. Pre-generated Tx waveforms will be radiated from one or multiple transmitting units and received by one or multiple receivers. The captured data is then stored in a dataset library for further utilization.

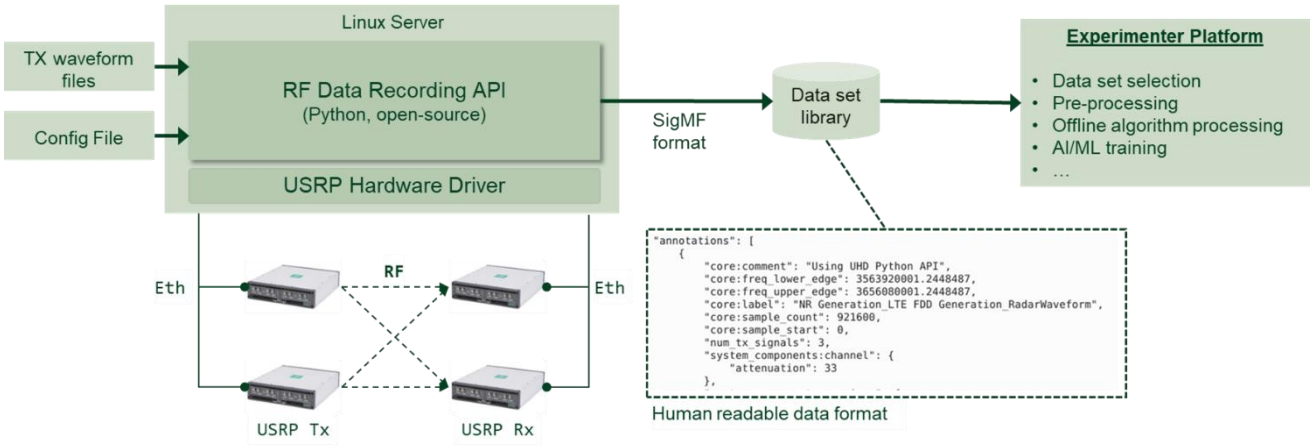

Figure 4: General architecture of the data recording framework

[2] https://www.kuleuven.be/rdm/en/rdr

3.2.2 Hardware and Software Requirements

NI's USRP X410 [10] and NI's USRP X310 [9] models are supported by the RF data recording framework, while the number of transmitters and receivers can be configured flexibly. The devices are connected via 10G Ethernet connections to a single or multiple Linux-based host computers, depending on the respective recording scenario. For realistic scenarios, the respective signal can be transmitted over the air via antennas or in a cabled manner, potentially utilizing channel emulators. A cabled example recording setup with one transmitter and one receiver is depicted in Figure 5.

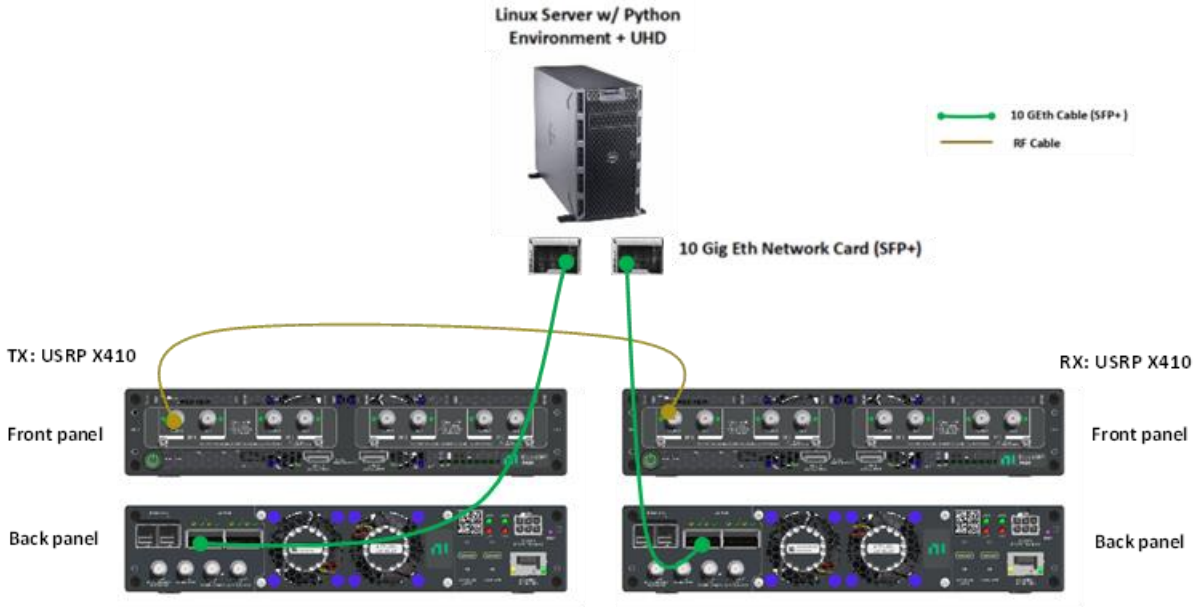

Figure 5: Example recording setup with one transmitter and one receiver (both USRP X410)

3.2.3 Software Architecture

Figure 6 shows the general software architecture of the data recording framework.

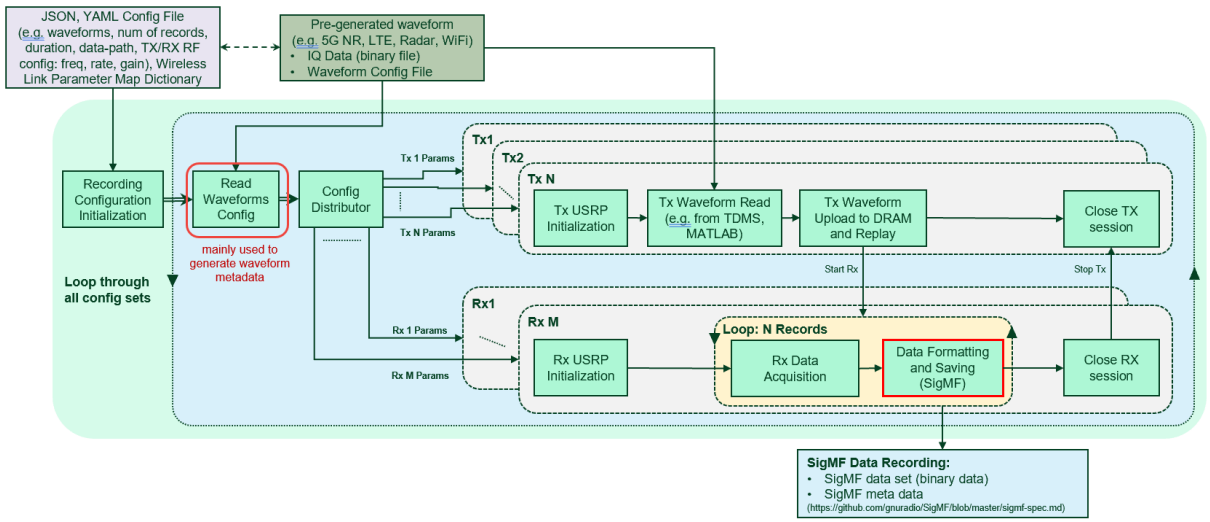

Figure 6: Software architecture of the RF Data Recording Framework

As a first step, all the parameters that define a respective recording scenario are read-in from a configuration file. The recording scenario is then decomposed into multiple configuration sets that will subsequently be executed. Configuration sets are created by calculating the cross-product over all parameter ranges and could differ e.g. in the operating frequency, the waveform to be transmitted, or other parameters. The transmitters and receivers are configured and operated via the USRP hardware driver from within the main scenario execution loop that also stores the captured I/Q samples in SigMF [13] format to the dataset library.

In the following, the inputs and outputs to and from the RF Data Recording framework are described.

Configuration File

The configuration file in JSON or YAML format is the main resource for defining the respective recording scenario. System parameters like operating frequency, number of USRP devices, number of records, and more are defined. An example configuration file is depicted in Figure 7 to demonstrate the general concept.

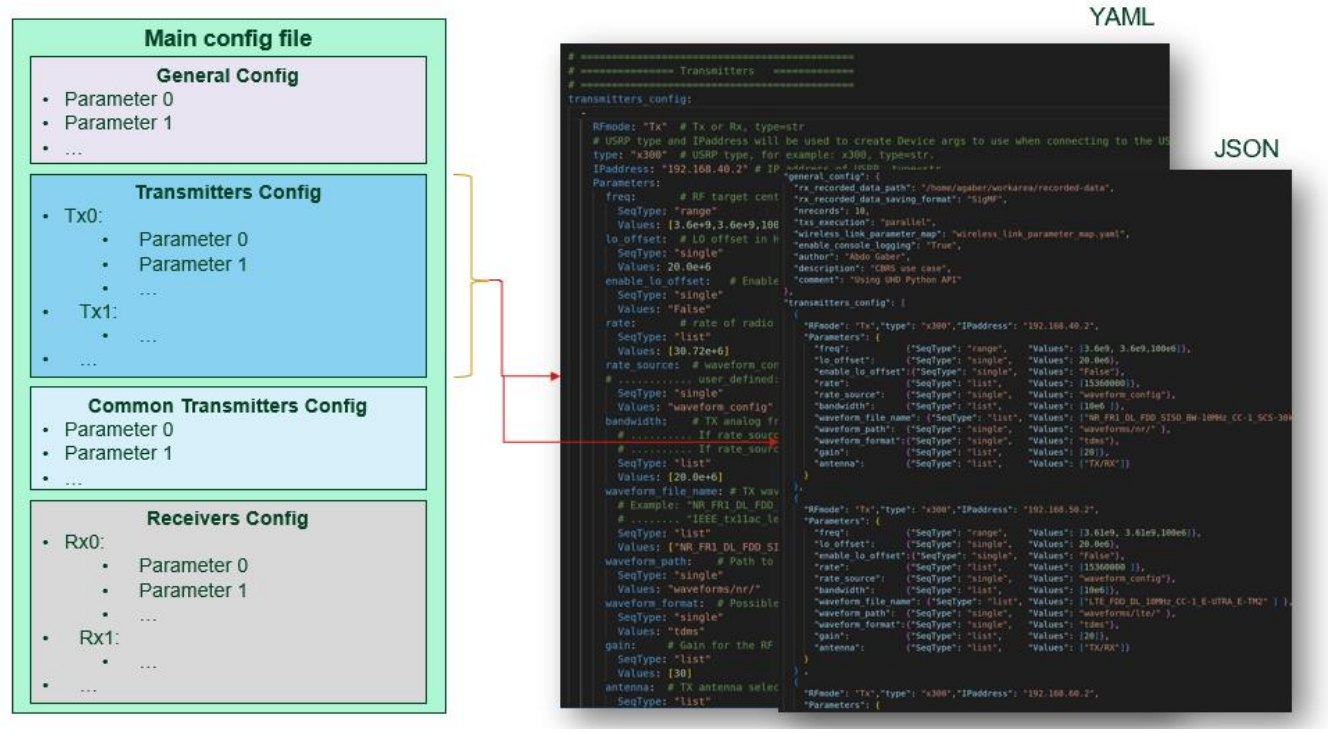

Figure 7: Example configuration file in YAML and JSON format

Each configuration file consists of the following sections:

- General configuration: file paths, number of records, description of the scenario, …
- Transmitter configuration: frequency, gain, sampling rate, antenna ports, …
- Common Transmitter configuration: clock reference, waveform replay data configuration
- Receiver configuration: same as transmitter configuration

A more detailed description of the parameters and sample configuration files can be found in [8].

Waveforms

While a few examples of 5G NR, LTE, radar, and Wi-Fi waveforms are provided together with the data recording framework, users can also create compliant waveforms that fit the respective recording scenario by using

external tooling like MATLAB, the RFmx Waveform Creator [11], or others. The following data formats are currently supported:

- tdms (e.g., created by the NI RFmx Waveform Creator)
- mat (e.g., created by MATLAB)

Support for other data formats could be added as needed. Each waveform should consist of a (i) binary data file and a (ii) human-readable waveform description file. Both files should have the same file name.

Wireless Link Parameter Map

Multiple waveform creation tools and playback waveform formats are supported. However, the structure and naming conventions of the output format need to be consistent, independent of the user's input waveform. The Wireless Link Parameter Map file in YAML format is therefore used to define a mapping between the input parameter names and the respective output parameter names of the SigMF meta data. For example, the user could define to map the input waveform parameter "Bandwidth (Hz)" to the SigMF meta data parameter "bandwidth". The data recording framework will then automatically copy the bandwidth value from the waveform description information to the recorded SigMF meta data file.

Results

The recorded datasets are stored in the Signal Metadata Format (SigMF) format. Besides the binary data, it additionally includes human-readable metadata which is crucial for contextual understanding and automation purposes.

Dedicated SigMF packages are available for e.g. Python to ease the interaction with SigMF files, thus enabling further post-processing of the data sets.

3.2.4 Potential extensions to the framework and future work

The 6G-BRICKS project demands for extended functionality to be added to the Data Recording Framework to comply with the requirements given by the Cell-Free massive Multiple-Input and Multiple-Output (CF mMIMO) scenario and the architecture of the higher layers of the testbed. Thus, additional work is required to e.g. add support for a specific type of mmWave frontend or to be able to read-in and process user defined waveforms and waveform formats beyond what is currently offered. It is also needed to add additional post-processing or data conversion functionality. Finally, the viability to capture data from complex CF scenarios as well as the integration with the higher management layers of the testbed needs to be ensured which might require adaptions to the architecture of the framework.

© 6G-BRICKS 2023

4 6G-BRICKS Enablers to be deployed at KUL Testbed

4.1 Open-RU platform deployment

The Open-RU is based on the USRP X410 software-defined radio by NI that provides the digital and analog RF processing blocks as well as analog-to-digital conversion functionality. Following the 7.2 functional split defined by the O-RAN alliance, additional OFDM specific processing like FFT/IFFT and cyclic prefix handling is implemented as well. An overview of the basic building blocks and the functional split between RU and DU is depicted in Figure 8.

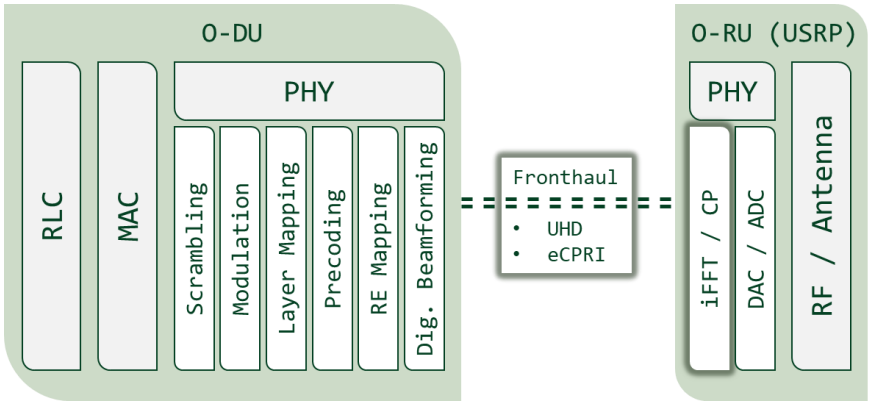

Figure 8: Basic building blocks of a "Category A" Open-RU

While an O-RAN compliant, eCPRI-based front haul between RU and DU would be beneficial for the interoperability between O-RAN compliant components, it also increases the integration effort on the RU as well as DU side and decreases the flexibility. As the 6G-BRICKS testbeds should allow for fast prototyping and easy customization of user specific scenarios, a simplified interface based on the USRP Hardware Driver (UHD) is provided as a first step. User data and control commands are exchanged over the respective Ethernet interfaces between RU and DU while tight synchronization is ensured via an external timing reference. Figure 9 depicts the basic concept.

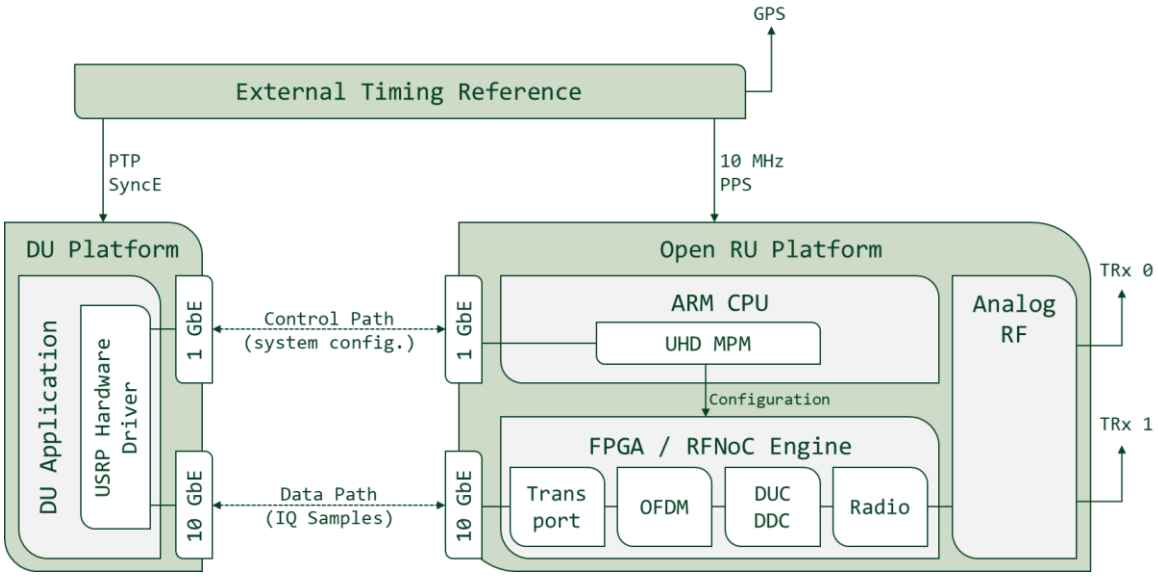

Figure 9: Basic architecture of the Open-RU with cabled RU-DU synchronization

The fronthaul between Open RU and the DU platform consists of two separate physical interfaces. Control commands are sent over the 1GbE connection to the Open RU where they are handled by the Module Peripheral Manager (MPM) running on the USRP's CPU. The MPM controls basic system parameters like center frequency, subcarrier spacing, FFT sizes, output power, and others.

The actual user data in the form of frequency domain I/Q samples are transmitted over the high-speed 10GbE connection and are received and processed directly by the signal processing algorithms running on the FPGA. Basic functionality like Ethernet transport handling, OFDM processing, digital up- and down-conversion, and the control of the analog radio unit is encapsulated in modular RFNoC blocks [12]. RFNoC is an open-source framework designed to enhance the capabilities of USRP devices by enabling the deployment of user-defined blocks to further extend the signal processing chain running on the FPGA. Additional features like precoding or beamforming could thus be made available in the future. The framework also supports the dynamic reconfiguration of the signal processing chain without the need for hardware changes which is particularly useful in dynamic and adaptive radio systems. Two transmit and receive chains are currently supported as depicted in Figure 9. More chains will be available in the future.

The Open RU software components are **installed in the KUL Testbed Controller** and are accessible by all experiment scripts and by the DU software components. The software drivers are part of the base infrastructure and do not depend on any container technology. The drivers are installed in the permanent storage of the Testbed Controller under a fixed path.

The Open RU firmware, being implemented within 6G-BRICKS, is **deployed directly at the USRP X410 peripherals**, which are part of the KUL Radio Infrastructure. The firmware consists of an FPGA image and embedded firmware running on the ARM core of the Xilinx RF SoC, which constitutes the heart of the USRP. The USRPs are connected to the Testbed Controller with two separate interfaces: a 1GB Ethernet for command and control and a 100Gb Ethernet for streaming of I/Q samples.

Synchronization

An external timing reference is used to synchronize the DU platform with the Open RU. This GPS-disciplined device, which is part of the KUL testbed, supports on one hand the PTP / SyncE protocol to ensure a synchronization of the DU via the DU's network interface controller. On the other hand, it outputs 10 MHz / PPS reference signals that can be connected directly to the USRP to synchronize one or multiple Open RUs. While this cabled setup acts as a baseline, more advanced synchronization techniques utilizing e.g. OTA synchronization algorithms might also be investigated.

4.2 Cell-Free Massive MIMO enablers

4.2.1 Deployment of Cell-free mMIMO enabled RAN

The CF mMIMO enabled RAN solution can be deployed as O-DU, O-CU or combined O-DU/O-CU in a few as a Docker Compose image. The CNF can be deployed on a Docker container with Docker Compose. In the case of a single machine deployment (combined O-DU/O-CU), the user has one YAML file for entity management and network configuration. In the case of several machines (O-DU, O-CU or separate O-CU-CP and O-CU-UP), the user must configure several YAML files and properly configure the networks.
The RAN components can be deployed on different infrastructure (computers, servers, cloud). The Hardware requirements depend on the network configuration and performance of the used processor. The deployed RAN in the KUL testbed consists of O-DU, O-CU, Near-RT RIC and the relevant xApp(s). The O-DU, O-CU, Near-RT RIC and xApp are deployed as containers within a single server. While O-DU, O-CU, Near-RT RIC are interconnected with standard interfaces (F1 and E2), xApp is attached to RIC via a proprietary interface. The software consisting of Near-RT RIC, O-DU, and O-CU is deployed in the KUL testbed as containers running on the Testbed Controller.

© 6G-BRICKS 2023

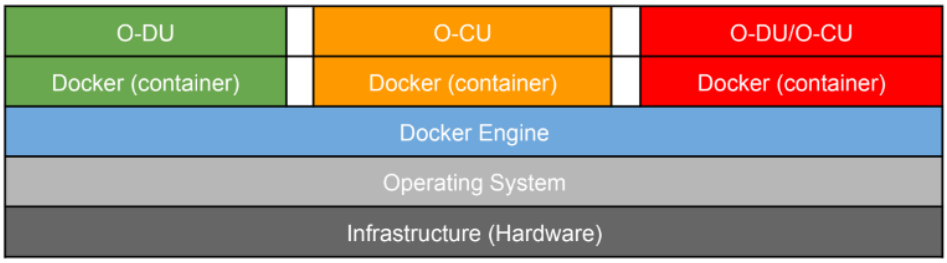

Figure 10: Deployment schema of the O-DU and O-CU components

The whole ecosystem is connected to the external components in the following way:

- O-DU communicates with USRP over the Fronthaul interface
- the NI's USRP x410 used as an O-RU
- The O-CU communicates with Core Network over standard defined N2/N3 interfaces
- The xApp includes an external interface that exposes OpenAPI-compliant interfaces, which allows standard means to access RAN related metrics

To deploy the RAN at the KUL, several requirements are listed in below in the case of channel bandwidth = 20MHz, MIMO 2x2, Max 256QAM in DL

Table 2 - System requirements for RAN deployment

Bandwidth	CNF	Processor (example)	vCPU	Memory	Storage
20 MHz	O-DU only	PC: minimum 9th Gen Intel x64 Core i7 9700K Server: minimum xeon gen2	8	8G	100GB + logs
20 MHz	O-CU only	PC: minimum 9th Gen Intel x64 Core i7 9700K Server: minimum xeon gen2	6	8G	100GB + logs
20 MHz	O-DU/O-CU	PC: minimum 9th Gen Intel x64 Core i7 9700K Server: minimum xeon gen2	14	16G	200GB + logs

Requirements for Operating System (and Kernel): Ubuntu Server 20.04 (with Low Latency Kernel: 5.11.0-46-lowlatency or newest) and Ubuntu Server 22.04 (with Low Latency Kernel: 5.15.0-67-lowlatency or newest).

4.2.2 Deployment of Cell-free mMIMO Near-RT RIC

The CF mMIMO enabled Near-RT RIC is the O-RAN compliant solution that connects to the O-DUs and the O-CUs over E2 interface can be deployed with Docker Compose. In this deliverable, we describe the basic deployment of the solution based on Cloud-native Network Function (CNF) deployment on a Docker container with Docker compose. The Near-RT RIC as a softwzare along with trajectory prediction xApp and the multi-band clustering xApp can be deployed on different infrastructure (computers, servers, and the cloud). To deploy the Near-RT RIC several requirements are listed in bellow:

- BIOS To run the L1 PHY application, you should configure the BIOS to Low latency/Real Time mode. If the BIOS setup utility provides a profile designed for Low latency, select this option to lock in a predetermined set of settings known by the system vendor to optimize latency.
- Hardware requirements (Processor, vCPU, Memory, Storage)
    - PC: minimum 9th Gen Intel x64 Core i7 9700K Server: minimum xeon gen2, vCPU (8), memory (8 GB), and storage (100 GB + Logs)
- Software requirements (Operating system, docker and docker compose)
    - Operating system: Ubuntu Server 18.04.6 LTS x86_64. Ubuntu Server 20.04.2 LTS x86_64
    - Docker and Docker-compose versions: Docker version 20.10.17, build 100c701 or newer. Docker-compose version 1.29.2, build 5becea4c or newer. Docker-compose configuration file version 3.8.

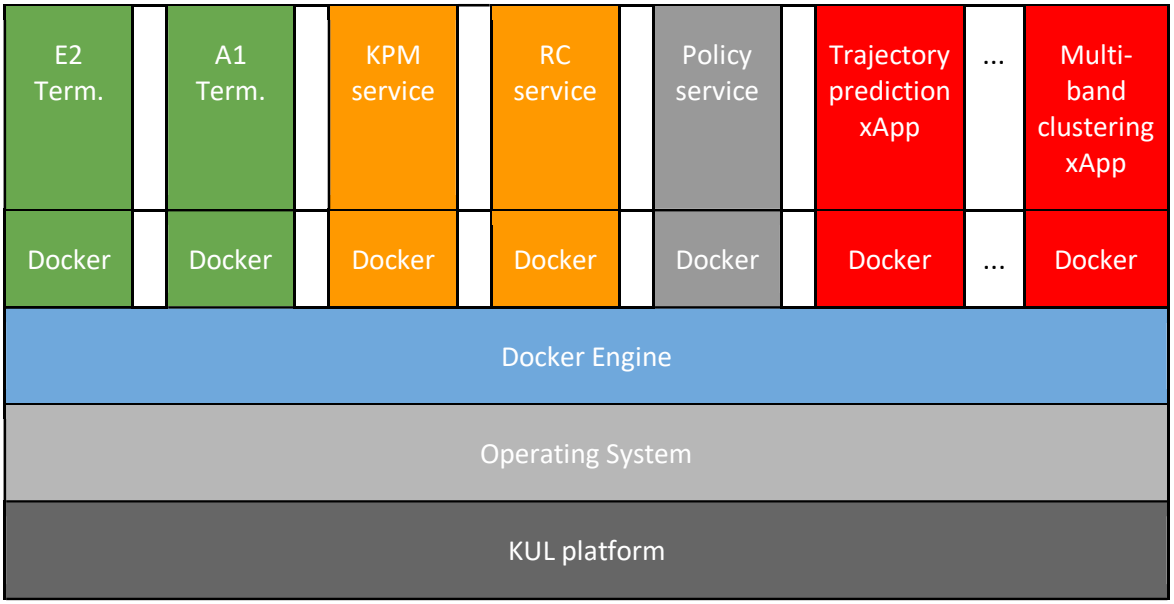

Figure 11 Deployment schema of Cell-free mMIMO Near-RT RIC on a docker container

Near-RT RIC deployment and configuration:

The Near-RT RIC can be a CNF deployment on a Docker container with Docker Compose. In the case of single machine deployment, the user has one YAML file for entity management. The deployment consists of the following steps:

- Pulling Near-RT RIC docker images from DockerHub.
- Configuration of the Near-RT RIC deployment using docker-compose.yaml file.

4.3 Deployment of the Cognitive Compute Continuum enabler

The 6G-BRICKS Cognitive Compute continuum enabler is offered through the usage of the π-Edge Platform, provided by ICOM, which is accessed by the higher-level components of 6G-BRICKS (e.g. the Experimentation Platform and the Edge-Cloud Continuum Manager). The server running the π-Edge continuum has the following characteristics:

- Core i5-8500

© 6G-BRICKS 2023

- 16GB of RAM
- 2TB SSD (1 SATA + 1 M.2)
- 2x 1Gb Ethernet interface

The computer runs Ubuntu Server 22.04 LTS, and it is placed on the external network of KUL. The Cognitive Compute Continuum is comprised of two components:

- the π-Edge Platform, which manages an Edge-Cloud, and,
- the Edge-Cloud Continuum Manager (ECCM), which is a central component and manages several Edge-Clouds (i.e. several π-Edge Platforms).

From these two components, the π-Edge Platform will manage the KUL compute cluster. Additionally, a wireless X-Haul transport solution, provided by ICOM, will also be installed in the testbed. This solution is comprised of a radio device and a white box switch accompanied by a Network Operating System, acting both as a radio link (connecting far edge devices) and as a compute node (managed by the π-Edge Platform). Moreover,

Configuration of π-Edge Platform

First of all, a Kubernetes cluster with at least 2 nodes is used. For that purpose, microk8s is installed to initiate our cluster. The Master node is KUL's server.

Certain prerequisites were met to prepare the cluster appropriately:

1. Each node of the cluster has a **location** label to make sure that it is recognized from the platform during its initialization.
2. There exist **monitoring** namespace with both Prometheus and Grafana enabled, to ensure the proper operation of all monitoring functionalities of the platform.

From the software perspective, the π-Edge Platform consists of three software components, (1) two distinct databases, where each offers different functionalities to the platform and (2) the main controller:

- **MongoDB:** This component is a usual MongoDB database, which is used to store all the important information that will need to be handled from the platform's APIs. In this database, we can track the registered Service Functions, PaaS services and nodes. The Deployment (Listing 1), Service (Listing 2) and Persistent Volume Claim's (Listing 3) yaml files are depicted in the following listings.

```
apiVersion: apps/v1
kind: Deployment
metadata:
    creationTimestamp: null
  labels:
    piedge.service: mongopiedge
  name: mongopiedge
spec:
  replicas: 1
  selector:
    matchLabels:
      piedge.service:: mongopiedge
  strategy:
    type: Recreate
  template:
    metadata:
      creationTimestamp: null
      labels:
        piedge.service:: mongopiedge
```

```
    spec:
      containers:
        - image: mongo
          name: mongopiedge
          ports:
            - containerPort: 27017
          resources: {}
          volumeMounts:
            - mountPath: /data/db
              name: mongo-db
      restartPolicy: Always
      volumes:
        - name: mongo-db
          persistentVolumeClaim:
            claimName: mongo-db
```

Listing 1: MongoDB Deployment YAML file

```
apiVersion: v1
kind: Service
metadata:
    creationTimestamp: null
  labels:
    piedge.service: mongopiedge
  name: mongopiedge
spec:
  type: NodePort
  ports:
    - name: "27017"
      nodePort: 32411
      port: 27017
      targetPort: 27017
  selector:
    piedge.service: mongopiedge
```

Listing 2: MongoDB Service YAML file

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  creationTimestamp: null
  labels:
    piedge.service: mongo-db
  name: mongo-db
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Mi
status: {}
```

Listing 3: MongoDB Persistent Volume Claim YAML file

© 6G-BRICKS 2023

- **Postgres:** This is a PostgreSQL database, used to keep track of users of the platform and their roles. Every user that exists in the database will be able to obtain an authorization token which will give them access to all the functionalities. All files needed for the deployment and configuration of this specific component are presented in Listings 4 to 8 for Deployment, Service, ConfigMap, PersistentVolume and PersistentVolumeClaim resources, respectively.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: postgres  # Sets Deployment name
spec:
  replicas: 1
  selector:
    matchLabels:
      app: postgres
  template:
    metadata:
      labels:
        app: postgres
    spec:
      containers:
        - name: postgres
          image: postgres:10.1 # Sets Image
          imagePullPolicy: "IfNotPresent"
          ports:
            - containerPort: 5432  # Exposes container port
          envFrom:
            - configMapRef:
                name: postgres-config
          volumeMounts:
            - mountPath: /var/lib/postgresql/data
              name: postgredb
      volumes:
        - name: postgredb
          persistentVolumeClaim:
            claimName: postgres-pv-claim-pi-edge
```

Listing 4: Postgres Deployment YAML file

```
apiVersion: v1
kind: Service
metadata:
  name: postgres # Sets service name
  labels:
    app: postgres # Labels and Selectors
spec:
  type: NodePort # Sets service type
  ports:
    - port: 5432 # Sets port to run the postgres application
  selector:
    app: postgres
```

Listing 5: Postgres Service YAML file

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: postgres-config
  labels:
    app: postgres
data:
  POSTGRES_DB: postgresdb
  POSTGRES_USER: <YOUR_USERNAME>
  POSTGRES_PASSWORD: <YOUR_PASSWORD>
```

Listing 6: Postgres Configuration YAML file

```
kind: PersistentVolume
apiVersion: v1
metadata:
  name: postgres-pv-volume-pi-edge # Sets PV's name
  labels:
    type: local # Sets PV's type to local
    app: postgres
spec:
  storageClassName: manual
  capacity:
    storage: 200Mi # Sets PV Volume
  accessModes:
    - ReadWriteMany
  hostPath:
    path: "/mnt/data/data"
```

Listing 7: Postgres Persistent Volume YAML file

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: postgres-pv-claim-pi-edge # Sets name of PVC
  labels:
    app: postgres
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteMany  # Sets read and write access
  resources:
    requests:
      storage: 200Mi # Sets volume size
```

Listing 8: Postgres Persistent Volume Claim YAML file

- **Piedgecontroller:** The π-Edge platform's primary controller handles all functionalities, including API calls and database writes. Proper setup of environment parameters is crucial for its operation and the entire platform. Listings 9 and 10 present the files needed for the deployment and configuration of the π-Edge Controller through its Deployment and Service resources.

© 6G-BRICKS 2023

```
apiVersion: apps/v1
kind: Deployment
metadata:

  creationTimestamp: null
  labels:
    piedge.service: piedgecontroller
  name: piedgecontroller
spec:
  replicas: 1
  selector:
    matchLabels:
      piedge.service: piedgecontroller
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:

        piedge.service: piedgecontroller
    spec:
      containers:
        - env:
            - name: KUBERNETES_MASTER_HOSTNAME
              value: <YOUR_HOSTNAME>
            - name: KUBERNETES_MASTER_PASSWORD
              value: <YOUR_HOSTNAME_PASSWORD>
            - name: KUBERNETES_MASTER_IP
              value: <YOUR_HOSTNAME_IP>
            - name: KUBERNETES_MASTER_TOKEN
              value: <YOUR_K8S_MASTER_TOKEN>
            - name: EMP_STORAGE_URI
              value: mongodb://mongopiedge:27017
            - name: MONITORING_IP
              value: <YOUR_MONITORING_IP>
            - name: MONITORING_NODE_PORT
              value: <YOUR_MONITORING_NODE_PORT>
            - name: SYSTEM_DATABASE_PASSWORD
              value: <YOUR_POSTGRES_PASSWORD>
            - name: SYSTEM_DATABASE_HOST
              value: "postgres"
            - name: MONITORING_SERVICE_PASSWORD
              value: <YOUR_MONITORING_SERVICE_PASSWORD>
            - name: MEC_PLATFORM_IP
              value: <YOUR_MEC_PLATFORM_IP>
            - name: MEC_PLATFORM_PORT
              value: "32500"
            - name: MONITORING_IP_PROXY
              value: <YOUR_MONITORING_IP_PROXY>
            - name: MONITORING_PORT_PROXY
              value: <YOUR_MONITORING_PORT_PROXY>

          image: nikospsar/pi_edge:2.0.0
```

```
          name: piedgecontroller
          ports:
            - containerPort: 8080
          resources: {}
          imagePullPolicy: Always
      restartPolicy: Always
```

Listing 9: piedgecontroller Deployment YAML file

```
apiVersion: v1
kind: Service
metadata:

  creationTimestamp: null
  labels:
    piedge.service: piedgecontroller
  name: piedgecontroller
spec:
  type: NodePort
  ports:
    - name: "8080"
      nodePort: 32415
      port: 8080
      targetPort: 8080
  selector:
    piedge.service: piedgecontroller
```

Listing 10: piedge controller Service YAML file

After preparing the cluster with the already mentioned steps and deploying all the yaml files, the π-Edge platform is fully functional. A GET request on the following endpoint is able to initialize all the nodes defined in the KUL cluster and fully integrate the platform with the cluster which is deployed on:

http://<HOSTNAME_IP>:8080/piedge-connector/2.0.0/initNodes

5 Deployment and Configuration of 6G-BRICKS enablers at EUR facility

5.1 EUR testbed overview

Figure 12 depicts the deployment of the RIS-related 6GBRICKS enablers at the EUR testbed. These enablers are: (i) the RIS control framework provided by EUR and (ii) DMO provided by EUR.

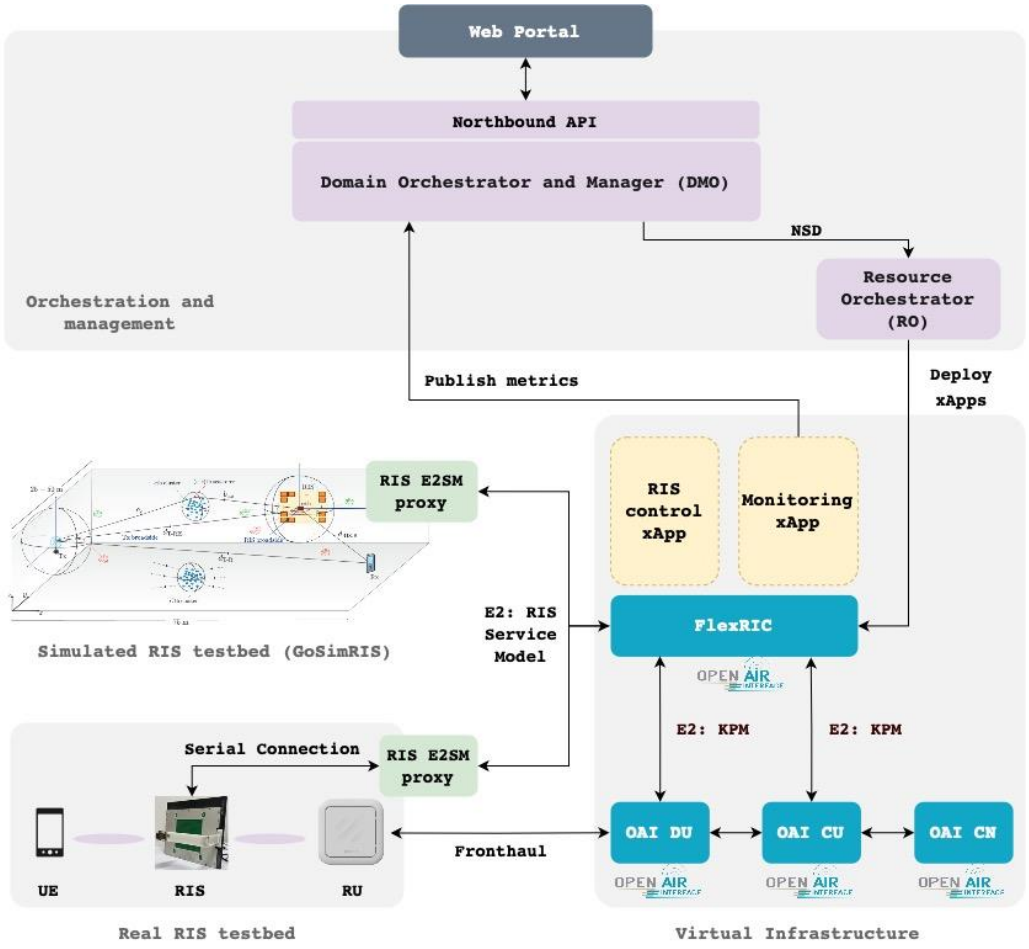

Figure 12: 6GBRICKS enablers deployment at EUR Facility

One of the main components of the EUR testbed is the FR2 setup. FR2 stands for Frequency Range 2 (FR2) which includes frequency bands from **24.25 GHz to 52.6 GHz.** FR2 is currently provided using Open Air Interface (OAI) Distributed Unit (DU), OAI Centralized unit (CU) and OAI 5G Core Network (CN). USRP X410 with Interdigital Masthead Unit and LiteON FlexFI product are used as Radio Units.

The InterDigital MHU provides a single-beam per MHU using a 64-element array. A single USRP X410 can control 2 MHU to allow for two concurrent beams for OAI baseband processing. The MHU is capable of 200 MHz channel bandwidth although current testing with OAI is limited to single 100 MHz channels. It provides 64 independent beams in three dimensions with a horizontal opening of 90 degrees.

The LiteON FR2 O-RU provides two independent concurrent beams on the same 64-element (8x8) array with 49 dBm EIRP output power. It uses the O-RAN 7.2 fronthaul interface and operates as a Category-A O-RU providing 120 degree horizontal and vertical beamwidths. The unit operates in band n257 (26500MHz -29500MHz) with up to 400 MHz operating bandwidth. Beam selection is handled by the O-RAN 7.2 C-plane API.

© 6G-BRICKS 2023

Another crucial aspect of the EUR facility is the novel orchestration and management framework that separates service and resource management, as depicted in Figure 12. These responsibilities are managed by the DMO for service management, and the Resource Orchestrator (RO) for resource management, respectively. The DMO is tasked with managing vertical services, providing monitoring information, and handling fault management. When deploying or updating vertical services, the DMO transmits the resource descriptor, named Network Service Descriptor (NSD), to the RO. This NSD defines the set of applications within the network service, specifies how they are exposed, and allocates the resources for each application. One main component of the DMO is the fault management component, which detects anomalies within the network. This component can directly request the RO to update configurations when anomalies are detected or predicted, or it can communicate anomalies with the vertical portal.

In the subsequent subsections, we explore the deployment and configuration of the RIS control framework, along with the configuration and deployment of the DMO's fault management related components.

5.2 RIS control framework deployment and configuration

The RIS control framework is composed of 3 components: (i) the RIS control xApp; (ii) the RIS E2SM; and (iii) the E2SM proxy as depicted in Figure 12 and also detailed in D2.2.

To initiate the deployment of the RIS control framework, our initial step involved the deployment of FlexRIC within the EUR testbed, leveraging Kubernetes as virtual infrastructure. To facilitate this deployment, we crafted a Helm chart tailored specifically for the FlexRIC application. A Helm chart serves as a package format designed for definition, installation, and management of Kubernetes applications. Among its core components, the deployment file. Listing 11 below depicts the deployment file of FlexRIC.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ include "oai-flexric.fullname" . }}
  labels:
    {{- include "oai-flexric.labels" . | nindent 4 }}
spec:
  replicas: {{ .Values.replicaCount }}
  selector:
    matchLabels:
      {{- include "oai-flexric.selectorLabels" . | nindent 6 }}
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        {{- include "oai-flexric.labels" . | nindent 8 }}
```

© 6G-BRICKS 2023

```
        app: oai-flexric
    spec:
      containers:
        - name: oai-flexric
          image: "{{ .Values.containerImage.repository }}:{{ .Values.containerImage.tag |
default .Chart.AppVersion }}"
          imagePullPolicy: {{ .Values.containerImage.pullPolicy }}
          securityContext:
            privileged: true
          command: [ "/bin/bash", "-c" ]
          args:
            - |
              # Copy files from /usr/local/lib/flexric/ to shared volume path
              cp -r /usr/local/lib/flexric/* {{ .Values.volume.sharedVolume.path }}
              # Replace NEAR_RIC_IP in flexric.conf with POD_IP
   sed -i "s/NEAR_RIC_IP = .*/NEAR_RIC_IP = $POD_IP /" /usr/local/etc/flexric/flexric.conf
              # Execute the nearRT-RIC example
              exec stdbuf -o0 /flexric/build/examples/ric/nearRT-RIC
          env:
            - name: POD_IP
              valueFrom:
                fieldRef:
                  fieldPath: status.podIP
          volumeMounts:
            - name: shared-volume
              mountPath: {{ .Values.volume.sharedVolume.path }}
      volumes:
        - name: shared-volume
          hostPath:
            path: {{ .Values.volume.sharedVolume.path }}
      restartPolicy: {{ .Values.restartPolicy }}
```

Listing 11: FlexRIC deployment file

Using FlexRIC Helm chart, the FlexRIC controller and the RIS E2SM are deployed within the testbed (Using the command *helm install).*

In the configuration, we specify two ports: (i) 36421 to connect FlexRIC with OAI DU/CU; and (ii) 36422 to connect FlexRIC with xApps. The xApp, after the deployment via the resource orchestrator, should be configured with this port + FlexRIC IP address (10.96.3.133 as specified in the deployment file) to be able to connect to FlexRIC.

© 6G-BRICKS 2023

The RIS E2SM is already integrated and available in the FlexRIC code base, from which the oai-flexric:develop

```
5694DA92B256D59A36ADA92A41A86A7A9CA657958E86A716958D18D12E12ED1AC7B44F92E4BE1B86F11ED3A47B84B8472472
52545ADB7A5254DA96A4ADAB69EA7A1A96A71585ADE7A515858C9BC13653C49B47B0ED9B66FF99E67B0ED1E613C19CC3E632
535B4AEB4B535A5ADAB2ACAB28EA3A8EA7A5A5C5E1EBA1E54584DB03B24FC4D927C0E43B26D798EC1B03D86713F18EC0E712
554BAAAB4D554B6ACABA92A4BE1A86AEA3A9E15571EAAAE15593D923826DCBD93648D0300C00300C07219C67E37982C867C6
5C8AAB17554C8BAAAAC99374D71DC663AAAAA23155D4EAA2359B71F34AC0CB53F6FCD2340D00701C873F9FC5E303A1CF4DE6
4A8B59577A6A8AAB1655D575555D575594EAA2A9ADD565E2A1B032F6FED0D032D4FCBF3FCFFFF3FCFE3F178C0707BF9F8C0E
6D0D59292A6909595655D6B7AADE975595657069A868657079B43004FCFFB03004001B27CBFFE3D8E400100C0EFF3F100C1E
654CBDA9B4254D592B6ACAB2AA8EA3A9E86571581E6A7E3159BFF40400DFBFF6C4810000000000000042139FFEF700101FFE
6245ACD18532E4A9AB36CD3341CC739CEA6A1B8C52473A518936FFF48000B6FDFCB52C0902418060385E3F7F9E00061FFF9C
34A74CB58B36A76CD335E978D72D6B5CC739DA9CE25E31DA1C003DB4BC24BC3D3F6D0F0340D701C0F079FC7C3E183E1E7C00
3922DE6DD21D22DCA789E2F8BE2F8B62DA3788748779B7886C0B3F0F4C0D0B3F4FCBF30CC13C4330CFE3F1FCE07031F0FCE0
1A613E58E41B639E6F92F59D69765F86F9B6C9A41B25BC49A402F4CFD8330274CF73F43D8F69F27C1FCDF31D80CC27F31F80
0CF06F30790CF13E1BC4FB1EC3B4EF13E4FC4F306D0CF90F300CF09F30B60CF23D27C8F72DC378DF23D87C8F309E0CF60F30
01F8CFE43301B8CFB3F83E4F96D1BC2FCEF32E40CC1BF32F4025923DA4D825936D9F61FA6E96B9AF49F679C6D8271A7C8658
073F0F8C0E073F8FC7F30CC23C8330CFD3F2FCD0B032F0FCD03611ED9EE12E11EC5B46D1F47D1F4791E53B44B84BB67B449C
003E787C187C3E3F9E070380EB02C0F0B6FCBC3D243D2DBC00385B8C7A47395B9CE33AD6B4EB1E97ACCB36E56CD1AD32E52C
39FFF8600079FEFC7A1C060182409034AD3FBF6D00012FFF6C918A5CE24A31D8565739CE3382CCB36CD595274CA18B35A246
7FF80800EF7FF9C842000000000000000081236FFDFB00202FFD9A8C7E56781A8EA61795C571554D5356D49AB2A42D95BD32A6
783008FCFF70300800271BC7FFD3E4D800200C0DFF3F200C2D9E0EA61615960EA6A9AAE97B55ED6BAA6A9A909654949AB0B6
7031F9FDE0E030E8FC7F3FCFFFF3FCFD3F2B4C0B0B7F6F4C0D8547A6ABB595455729AAEABAAAAEABAA68D551565EEA9AD152
67B2F385C0C7A3F9FCE3380E00B02C4B3F6FCAD30352CF8ED9AD45572BAA8C455555C663B8EB2EC99355555132AAE8D5513A
63E613419EC7E63984E0300C00300C0B126C9BD3B641C49BC9AA8755578EAA8795C57561587D25495D5356D2AAB2D555D2AA
48E703718FC8E61BC0D83719EB64DC2703E49B23F24DC0DB21A2A785D787A3A5A5E5715C5714D5354D5B5A5ACAD2D752DACA
4C67C33983C8678B70DE6799FF66D9B70DE2D923CA6C83D931A1A8A5E7B5A1A8E569585E5796D5B525695B2A4A5EDB5A2A4A
4E24E21D21DE25CB788F61D87D2749F22DE358B748748B18B1A968E56171A9EA65395E5615825495B56C59AB6A4D495B296A
```

Listing 12: RIS codebook configuration

Docker image has been built and pushed to the EUR facility local image registry.

The E2SM proxy is deployed in a Raspberry Pi and connected to the RIS via a Serial cable. The E2SM proxy is compiled with the *make* command and runs as a process inside the Raspberry Pi. It is configured with the following RIS parameters: number of elements, dimensions, codebook file location; where the latter is the most important configuration. Listing 12 depicts the codebook configuration of the CEA RIS. Each row represents the state of the RIS elements for a given beam angle (total number of beams is 25).

5.3 DMO's fault management deployment and configuration

Figure 13 depicts the deployment of the DMO-related fault management components at the EUR testbed. These components are: (i) the Abnormal Traffic Detector (ATD); and (ii) the LLM-enabled Trial Fault Explanator (LLM-TFE). The rest of the section explains the functionalities of these components.

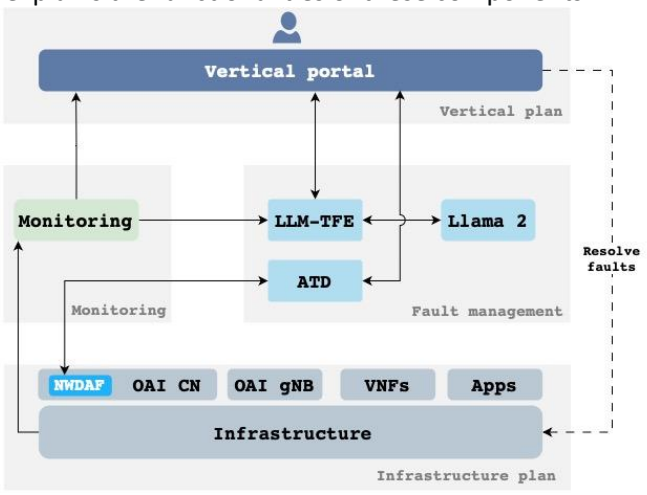

Figure 13: DMO fault management deployment at the EUR facility

5.3.1 Abnormal Traffic Detector (ATD)

The ATD is deployed to detect abnormal traffic among a specified list of the User Equipments (UEs), designated by vertical users through the vertical portal. Verticals can subscribe to the DMO's abnormal traffic detection functionality, receiving notifications when abnormal traffic is identified. The notification is generated by the DMO's ATD, which, upon deployment, is configured with OAI NWDAF's endpoint and a probability threshold (T) as environment variables. Here, the OAI Core Network (CN) deployment also includes NWDAF components. This enables the use of NWDAF's 'oai-nwdaf-engine-ads,' responsible for calculating probabilities of abnormal traffic. The deployment components of OAI NWDAF into the OAI CN are illustrated in Figure 14.

Upon receiving a subscription request from vertical users, the ATD uses the NWDAF endpoint to subscribe to NWDAF, obtaining abnormal traffic probabilities for the designated UEs. The subscription request is initiated using the 'abnormal_behavior' event ID and 'unexpected_large_rate_flow' exception ID, following 3GPP standards (TS 29.520). This enables the ATD to receive periodic abnormal traffic probabilities. Subsequently, the ATD compares these probabilities to the configured threshold (T). If the probabilities exceed the threshold, the ATD notifies vertical users of the UE experiencing abnormal traffic.

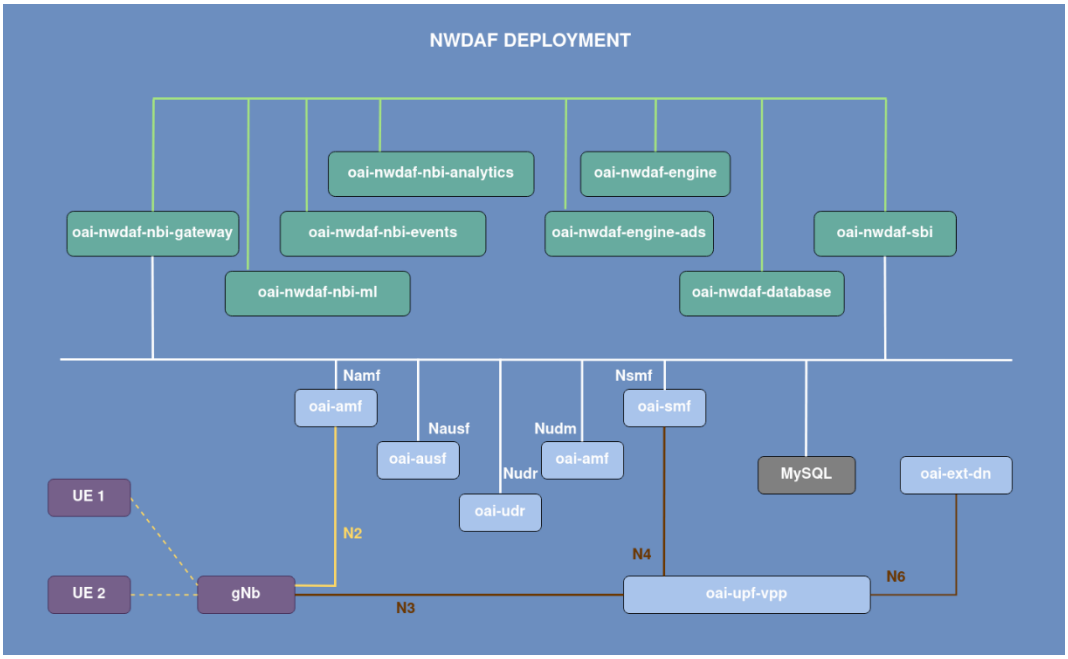

Figure 14: OAI CN NWDAF deployment to support ATD.

The Helm chart utilized for deploying the EUR facility has been extended to support the deployment and configuration of the ATD. As mentioned earlier, this configuration involves the use of environment variables to set the NWDAF endpoint and the probability threshold (T). This capability enables the ATD to communicate effectively with NWDAF. Furthermore, the ATD exposes APIs that allow vertical users to subscribe to the abnormal traffic probabilities functionality directly from the vertical portal. The body of the request (Listing 13) should contain the list of UEs to be monitored. In addition, the docker file used to build the ATD image is depicted in Listing 14 below.

```
{
  "notificationURI":                                    "http://{ATD-URI}/notification",
  "eventSubscriptions":                                                              [
```

```
    {
      "event":                                                    "ABNORMAL_BEHAVIOUR",
      "excepRequs":                                                                [
        {
          "excepId":                                          "UNEXPECTED_LARGE_RATE_FLOW"
        }
      ],
      "notificationMethod":                                                "PERIODIC",
      "repetitionPeriod": 2
        "tgtUe":                                                                   {
           "supis":                                                ["imsi-208950000000031"]
        }
    }
  ]
}
```

Listing 13: NWDAF Abnormal behavior request body.

```
python:3.6.9-slim

WORKDIR /atd

COPY . /atd

RUN pip install --no-cache-dir -r requirements.txt

EXPOSE 8000

CMD ["python", "run.py"]
```

Listing 14: ATD docker file.

5.3.2 LLM-enabled Trial Fault explanator (LLM-TFE)

Another crucial element in the fault management of the 6G-BRICKS DMO is the LLM-TFE. This component is responsible for predicting latency Service Level Agreement (SLA) violations for a given application within the infrastructure. Verticals can request this component to monitor a specific application, receiving explanations and recommendations when latency-related anomalies are predicted.

The LLM-TFE is configured with the monitoring endpoint to enable the collection of CPU and RAM usage information, along with associated limits. Leveraging these metrics, the LLM-TFE employs an AI model based on XGBoost to predict latency SLA violations and utilizes SHAP to identify the main features contributing to the prediction. Subsequently, it queries an LLM (e.g., Llama 2) to generate human-understandable text derived from the prediction and SHAP values. This text offers a clear human language explanation of the root causes and suggests actions to resolve anomalies to the vertical users. The Llama 2 LLM is deployed on a single NVIDIA A100 Tensor Core GPU and exposes OpenAI-compatible endpoints to be queried using HTTP. These endpoints are used by the LLM-TFE to generate the human language text and then send it to the vertical portal.

The Helm charts for the EUR facility have been further enhanced to incorporate the deployment of both the LLM-TFE, and the Llama 2 LLM. As previously mentioned, the LLM-TFE is configured with the monitoring component endpoint and the Llama 2 LLM endpoint. The deployment of Llama 2 LLM is facilitated using 'text-generation-webui' [14], that is configured only with the LLM HuggingFace link. In this context, we are utilizing the GPTQ version of instruct Llama2, i.e., TheBloke/Upstage-Llama-2-70B-instruct-v2-GPTQ [15], which is optimized to run

on a single A100 GPU. Listings 15 and 16 depict the docker files of both the LLM-TFE and 'text-generation-webui', respectively.

```
FROM registry.access.redhat.com/ubi8/python-38
RUN pip install --upgrade pip
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY . .
CMD ["python", "run.py"]
```

Listing 15: LLM-TFE docker file.

```
FROM ubuntu:22.04

WORKDIR /builder

ARG TORCH_CUDA_ARCH_LIST="${TORCH_CUDA_ARCH_LIST:-3.5;5.0;6.0;6.1;7.0;7.5;8.0;8.6+PTX}"

ARG BUILD_EXTENSIONS="${BUILD_EXTENSIONS:-}"

ARG APP_UID="${APP_UID:-6972}"

ARG APP_GID="${APP_GID:-6972}"

RUN --mount=type=cache,target=/var/cache/apt,sharing=locked,rw \
    apt update && \
    apt install --no-install-recommends -y git vim build-essential python3-dev pip bash curl && \
    rm -rf /var/lib/apt/lists/*

WORKDIR /home/app/

RUN git clone https://github.com/oobabooga/text-generation-webui.git

WORKDIR /home/app/text-generation-webui

RUN GPU_CHOICE=A USE_CUDA118=FALSE LAUNCH_AFTER_INSTALL=FALSE INSTALL_EXTENSIONS=TRUE
./start_linux.sh --verbose

COPY CMD_FLAGS.txt /home/app/text-generation-webui/

EXPOSE ${CONTAINER_PORT:-7860} ${CONTAINER_API_PORT:-5000} ${CONTAINER_API_STREAM_PORT:-5005}

WORKDIR /home/app/text-generation-webui

CMD umask 0002 && export HOME=/home/app/text-generation-webui && ./start_linux.sh
```

Listing 16: 'text-generation-webui' docker file.

© 6G-BRICKS 2023

6 Conclusions

This document provides a comprehensive overview of the initial deployment of enablers on the EUR and KUL 6G test facilities. By detailing the configuration settings, deployment processes, and operational guidelines for each enabler, it serves as a valuable resource for researchers and practitioners utilizing these facilities for 6G experimentation and development.

The deployment of the remaining enablers developed during the project and the remaining tasks related to Work Package 5 will be described in detail in future deliverables.

© 6G-BRICKS 2023

7 References

[1] A. P. Guevara, S. De Bast and S. Pollin, "Weave and Conquer: A Measurement-based Analysis of Dense Antenna Deployments," ICC 2021 - IEEE International Conference on Communications, Montreal, QC, Canada, 2021, pp. 1-6, doi: 10.1109/ICC42927.2021.9500612.

[2] A. Sakhnini, S. De Bast, M. Guenach, A. Bourdoux, H. Sahli and S. Pollin, "Near-Field Coherent Radar Sensing Using a Massive MIMO Communication Testbed," in IEEE Transactions on Wireless Communications, vol. 21, no. 8, pp. 6256-6270, Aug. 2022, doi: 10.1109/TWC.2022.3148035.

[3] S. De Bast, E. Vinogradov and S. Pollin, "Expert-Knowledge-Based Data-Driven Approach for Distributed Localization in Cell-Free Massive MIMO Networks," in IEEE Access, vol. 10, pp. 56427-56439, 2022, doi: 10.1109/ACCESS.2022.3177837.

[4] A. Colpaert, Z. Cui, E. Vinogradov and S. Pollin, "3D Non-Stationary Channel Measurement and Analysis for MaMIMO-UAV Communications," in IEEE Transactions on Vehicular Technology, doi: 10.1109/TVT.2023.3340447.

[5] A. Colpaert, S. De Bast, R. Beerten, A. P. Guevara, Z. Cui and S. Pollin, "Massive MIMO Channel Measurement Data Set for Localization and Communication," in IEEE Communications Magazine, vol. 61, no. 9, pp. 114-120, September 2023, doi: 10.1109/MCOM.004.2200716.

[6] https://codeocean.com/capsule/0881566/tree/v1

[7] https://www.esat.kuleuven.be/wavecorearenberg/research/NetworkedSystems/infrastructure/mimo

[8] NI. (2024). NI RF Data Recording API on GitHub. Retrieved from https://github.com/genesys-neu/ni-rf-data-recording-api

[9] NI. (2024). NI USRP X310. Retrieved from https://www.ettus.com/all-products/x310-kit/

[10] NI. (2024). NI USRP X410. Retrieved from https://www.ettus.com/all-products/usrp-x410/

[11] NI. (2024). RFmx Waveform Creator User Manual. Retrieved from https://www.ni.com/docs/de-DE/bundle/rfmx-waveform-creator/page/user-manual-welcome.html

[12] NI Ettus. (2024). RFNoC (RF Network on Chip). Retrieved from https://www.ettus.com/sdr-software/rfnoc/

[13] SigMF. (n.d.). Signal Metadata Format (SigMF). Retrieved from https://github.com/sigmf/SigMF

[14] https://github.com/oobabooga/text-generation-webui

[15] https://huggingface.co/TheBloke/Upstage-Llama-2-70B-instruct-v2-GPTQ

© 6G-BRICKS 2023

Funded by the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.