ENERGY DATA AND AUTOMATION ARCHITECTURE REPORT (DRAFT)

Deliverable D3.2

Co-funded by the European Union

6GSNS

The TARGET-X project has received funding from the Smart Networks and Services Joint Undertaking (SNS JU) under the European Union's Horizon Europe research and innovation programme under Grant Agreement No: 101096614

GRANT AGREEMENT	101096614
PROJECT TITLE	Trial Platform foR 5G EvoluTion – Cross-Industry On Large Scale
PROJECT ACRONYM	TARGET-X
PROJECT WEBSITE	www.target-x.eu
PROJECT IDENTIFIER	https://doi.org/10.3030/101096614
PROGRAMME	HORIZON-JU-SNS-2022-STREAM-D-01-01 — SNS Large Scale Trials and Pilots (LST&Ps) with Verticals
PROJECT START	01-01-2023
DURATION	30 Months
DELIVERABLE TYPE	Deliverable
CONTRIBUTING WORK PACKAGES	WP3
DISSEMINATION LEVEL	Public
DUE DATE	M12
ACTUAL SUBMISSION DATE	15.12.2023
RESPONSIBLE ORGANIZATION	RWTH-ACS
EDITOR(S)	Manuel Pitz
VERSION	1.0
STATUS:	Final
SHORT ABSTRACT	This deliverable provides an overview of the planned hardware and software architecture. Software wise storage, visualization and deployment is discussed. On the hardware side the development of synchronization, 5G modem carrier and analog acquisition is given.
KEY WORDS	Energy, 5G, Digital Transformation, Real-Time
CONTRIBUTOR(S)	Manuel Pitz, Benish Khan, Vincent Bareiß

Disclaimer

Executive Summary

Within the TARGET-X project, the integration of 5G in the energy, construction, manufacturing, and automotive verticals is evaluated. The goal is to identify already available features and possible features for 6G that can benefit these verticals. The energy vertical targets mainly the topics of monitoring and energy awareness. The developed software and hardware are not only used for grid-specific use cases, but also in other verticals to evaluate the energy consumption of a specific process.

This deliverable provides an overview of the hardware and software architecture for the energy vertical. This includes the edge cloud as well as the field devices. On the edge cloud side the architecture includes services for provisioning, supervision, control, storage and visualization. Furthermore, supporting services like access control and a configuration database are needed. On the field side, two different types of hardware are described. For the monitoring of voltages and currents, the 5G edgePMU is presented, and for energy awareness, an outdoor capable measurement box is proposed. This box includes a 5G edgePMU at its core, but also additional metering and safety hardware needed for compliance. A detailed description of different components like 5G edgePMU, measurement devices, and edge infrastructure is given in the following chapters. The first promising results of a component-level evaluation for hardware and software components is provided. This deliverable is the first revision and thus mainly provides information on the design choices and the indicated next steps in the development process. The second revision will provide a final overview of the platform in terms of hardware and software.

Table of Contents

List of Figures

List of Acronyms and Abbreviations

5G	5th Generation
AC	Alternating Current
ADC	Analog to Digital Converter
CSV	Comma Separated Values
DC	Direct Current
GOOSE	Generic Object Oriented Substation Event
HDF5	Hierarchical Data Format5
HTTPS	Hypertext Transfer Protocol Secure
LV grid	Low Voltage grid
MIB	Management Information Base
MQTT	Message Queuing Telemetry Transport
NAT	Network address translation
PCB	Printed Circuit Board
PDC	Professional Darts Corporation
PLL	Phase Lock Loop
PMU	Phasor Measurement Unit
PPS	Pulse Per Second
QoS	Quality of Service
REST API	REpresentational State Transfer Application Programming Interface
RMS	Root Mean Square
SCADA	Supervisory Control and Data Acquisition
SD card	Secure Digital Card
SNMP	Simple Network Management Protocol
SSH	Secure Shell
TCP	Transmission Control Protocol
UI	User Interface
USB	Universal Serial Bus
UUID	Universal Unique Identifier
VPN	Virtual Private Network
ZFS	Zettabyte File System

1 Introduction

Within TARGET-X project, the 5G energy vertical is strongly focused on monitoring and energy awareness. This is especially important for the low voltage grid, where until now monitoring is not too common. Especially with the increase in volatile energy sources which are often deployed in the low voltage grid, a better understanding is key for a stable operation in the future. To achieve that goal, a major component needed is a 5G-enabled software and hardware architecture. This architecture has to be viewed holistically, including the field devices as well as the edge cloud infrastructure. Furthermore, it is important to include the actual communication components in this process. This will allow for an architecture that can be utilized across the different verticals and will increase energy awareness and grid monitoring capabilities in the fields of construction, manufacturing and energy. The gathered data is planned to be stored beyond the project duration to allow for future use within an academic environment.

This deliverable provides an overview of the planned software architecture and 5G hardware development for the energy use case. This specific deliverable is the first revision (Draft) of the architecture overview. An updated revision will be provided later in the project. First, the overall architecture is described. The following chapters then go into detail about the measurement hardware and different architecture components. Besides the draft of the architecture, some reasoning and information about planned work on different components, e.g. the broker, is given. Finally, the different subchapters contain preliminary measurements of already developed hardware and architecture components.

1.1 Document structure

This document is structured into four sections. Section 1 provides a brief Introduction. Section 2 provides an overview of the planned architecture. The detailed description of the different components is split into two blocks. The first block (Section 3) focuses on the field device, in this case the 5G edgePMU, which can measure voltage and current with high sampling rates and be synchronized to a GPS clock. Here, the hardware, as well as the software side, are discussed. The second block (Section 4) provides insights into the edge cloud software architecture. This includes communication, deployment, supervision, and storage. Finally, the document presents the conclusion in Section 5.

1.2 Relation to other activities

This deliverable is a first draft of the energy data architecture that will be developed and deployed within the TARGET-X project. This architecture as well as the 5G enabled energy measurement devices will be deployed within different verticals. Thus, there is a strong connection to WP5 (Construction) that is focused on construction and deconstruction processes and their energy footprint. Furthermore, the energy footprint of different manufacturing use cases will be studied. This creates a strong connection to WP2 (Manufacturing) where the developed measurement device will also be deployed. For the definition of use cases and definition of requirements, collaboration with WP1 (Methodological assessment framework) is done. Finally, collaboration with WP6 (Technology evolution beyond 5G) is done for the evaluation of the 5G features used within the 5G enables measurement box.

2 Architecture overview

This chapter will provide an overview of the planned architecture. The main objective of this architecture is to streamline the deployment and access to measurement data that is acquired in the field. A special focus is set on auto deployment to decrease the need for manual configuration and control actions, which is an obstetrical when deploying a higher number of devices. Furthermore, the data storage and sharing question is tackled by splitting the storage into a long-term and a short-term storage, as well as always storing the metadata bundled with the actual measurements. This will allow for reuse of the acquired data in follow-up projects and enable new data-driven use cases. It will provide insights into how the different components interact. The specific subcomponents will then be described in detail in the following chapters. As shown in Figure 1 the architecture is split into three major components. The top components are the services running within the edge cloud. Then the edge cloud is connected to the field devices via the 5G network through a VPN.

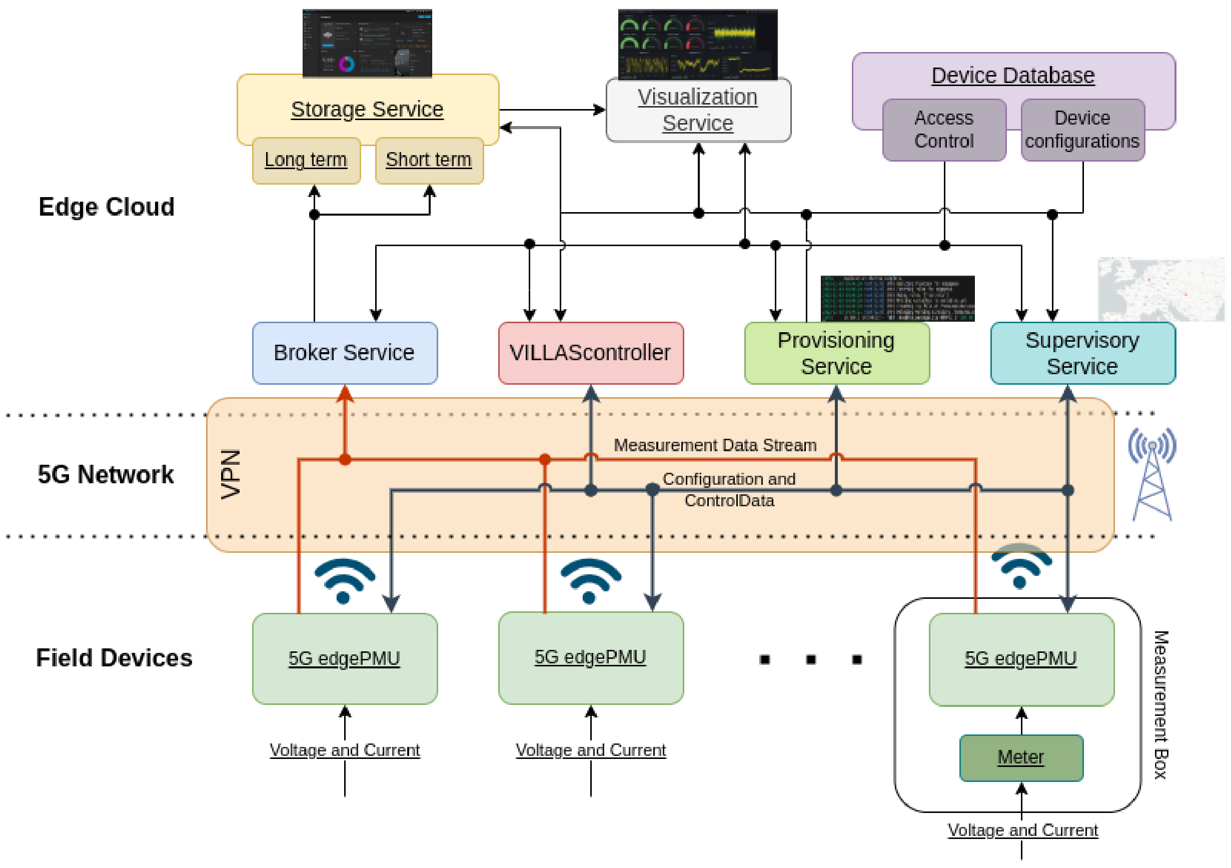


Figure 1: General Architecture of the Edge Cloud and Field side for the energy vertical.

The 5G edgePMU device is based on a 5G-enabled Raspberry Pi that is equipped with data acquisition hardware. The edgePMU is controlled (Section 3.3) and configured (Section 4.1) using the edge-cloud architecture. Furthermore, it streams the measurement results to the edge-cloud infrastructure using a Broker Service such as Message Queuing Telemetry Transport (MQTT) (Section 4.3). To allow for a secure connection, the edgePMU is connected to the cloud infrastructure via a Virtual Private Network (VPN) (Section 4.2). This secures the basic communication and allows for reaching the device even behind firewalls or Network Address Translation (NAT) systems. Since multiple devices are connected to the same Broker Service, the different device data streams have to be isolated from each other. The access control is handled by the Access Control part of the Database Service. A third major component is the Provisioning Service that is controlling the configuration of field devices (Section 4.1). To allow for a scalable infrastructure, the number of manual interventions for configuration and operation needs to be minimized. This is especially true for the configuration of the field devices. Apart from their device names and credentials, the devices could be installed with different configurations. Some examples are different 5G modems, GPS receivers or different data acquisition systems. To allow for simple and automated deployment, Ansible [1] is used. More details on the planned deployment stack are given in Section 4.1. A fourth core component is the Storage Service. The envisioned Storage Service can efficiently cope with long-term storage of either phasors, metering data or raw samples from multiple edgePMU devices (Section 4.5). The major challenge is the long-term storage. To reduce the risk of major data corruption events, a file-based approach is under investigation. The files should contain not only the actual measurements but also metadata, e.g., deployed sensor types, grid information, position information and more. The storage should be split into two types. Long-term storage and short-term storage for fast visualization. The last major component of the architecture is the Visualization Service (Section 4.6). This component should be accessible by the end users to allow access to short-term and long-term storage. It should also be possible to grant specific users access rights to specific datasets.

3 5G edgePMU

The edgePMU is a Raspberry Pi based hardware that utilizes open-source software to acquire measurements via the Universal Serial Bus (USB) based Analog to Digital Converter (ADC) device. The software used to implement the interface to the ADC and for the handling of measurements is called VILLASnode [7]. Within this project, it is planned to further develop the 5G version of the edgePMU. This will allow 5G based fast power measurements as well as long-term 3-phase energy measurement within one box. The box will be connected in line with the device under test. For that, the hardware, as well as the software stack, have to be updated. This is described in the following subchapters. First the hardware updates will be discussed, and in a second subsection, the software stack and its extension. An introduction to the edgePMU is given in Deliverable 3.1 [2].

3.1 Hardware

This subsection provides information about the planned hardware developments. Here the major topics are the input stage, that connects the edgePMU to the measured grid, the development of the 5G connectivity and the time synchronization.

3.1.1 Input Stage

The existing hardware only allows for the measurement of low voltage inputs, specifically for up to ±10 V. It is possible to measure on seven channels in parallel. These input levels are not directly compatible with most use cases, one of them is measuring the low voltage grid directly. Therefore, the edgePMU will be extended with a Low Voltage (LV) grid compatible input stage that allows for direct measurement of 230 V Alternating Current (AC) grid. Furthermore, the input stage will provide inputs for current transducers so that the device can measure both voltage and current, not needing third-party signal conditioning hardware. A first mockup of the planned hardware is shown in Figure 2. The shown version was already built as a prototype, and the first tests showed promising results. For the Direct Current (DC) case, the error for a ±10 V input configuration lies within 200 µV as shown in Figure 3. It is planned to develop a second revision based on the first prototype.

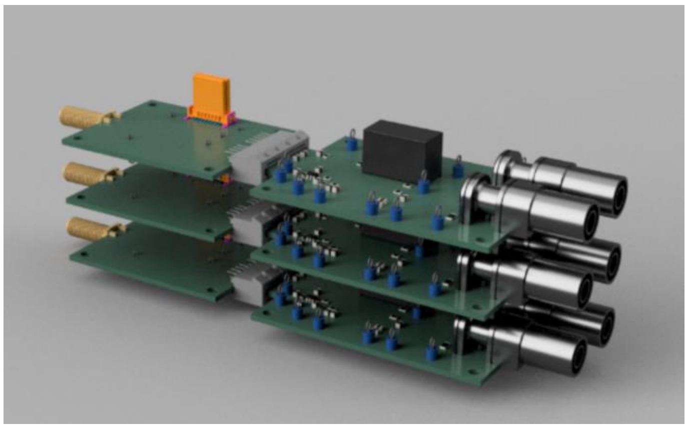


Figure 2: Three phase modular and isolated signal adaptation with inputs on the right side and analog output on the left side.

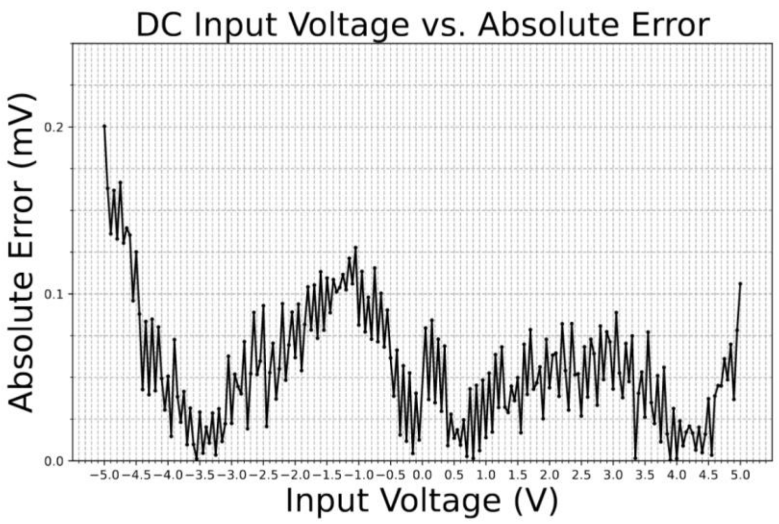


Figure 3: Absolute error for DC voltage measurement with an error of less than 200 uV within a full-scale input range of ±5 V.

3.1.2 5G Connectivity

Even so, the current version of the 5G edgePMU already supports 5G connectivity bundled in a Raspberry Pi hat, further integration is needed. The used 5G Module SimCom SIM8200 [4] is a M2 USB based module. A picture of the module can be seen in Figure 4. It is planned to integrate the module on a custom Printed Circuit Board (PCB) to allow for a smaller device footprint as well as being able to utilize the Pulse Per Second (PPS) output provided by the integrated GPS module. The PPS signal will be used for synchronized measurements. The resulting hardware will provide a simple USB connection and a GPS-based time synchronization output.

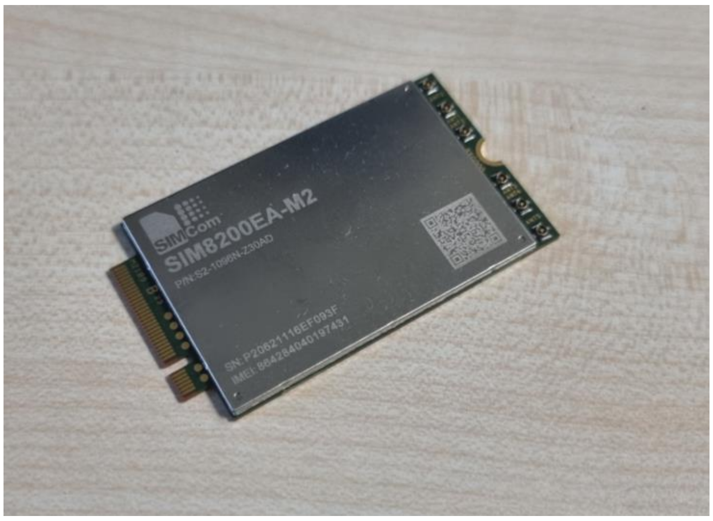


Figure 4: SimCom SIM8200EA-M2 module for 5G connectivity with onboard antenna connectors.

3.1.3 Time Synchronization

Time synchronization is critical for every PMU device. Up to now, the synchronization for the 5G edgePMU has been achieved by sampling the PPS signal. This allows for a simple time synchronization that can be achieved based on software running in a non-real time operating system environment (Figure 5 top). The downside of this type of synchronization is that the precision is directly related to the sampling rate. For a sampling rate of 10 kSamples/s, a time precision of 100 µs can be achieved. Based on this approach, achieving a time precision of higher than 100 µs on a Raspberry Pi is challenging due to the high data throughput needed at higher sampling rates. Further errors are introduced by the clock precision of the actual sampling clock. The resulting error can be observed in Figure 6 (the orange phase estimation result). A better solution is a hybrid approach for sampling. In that approach, the PPS signal is used as a time reference for a Phase Lock Loop (PLL). The PLL then generates a secondary PPS signal as well as a sampling clock for the acquisition device (Figure 5 bottom). This allows for much more precise timings without losing the easy to use USB interface and Raspberry Pi approach for the device. Based on a development board for the chosen PLL chip, a first test with the 5G edgePMU was already done. A comparison is shown in Figure 6. For the direct PPS sampling approach (orange) the phase is jumping periodically, whereas in the case of the hybrid sampling, the phase is estimated much more precisely. Based on these findings, it is planned to integrate the PLL chip with the 5G edgePMU. For that, a custom printed circuit board is needed.

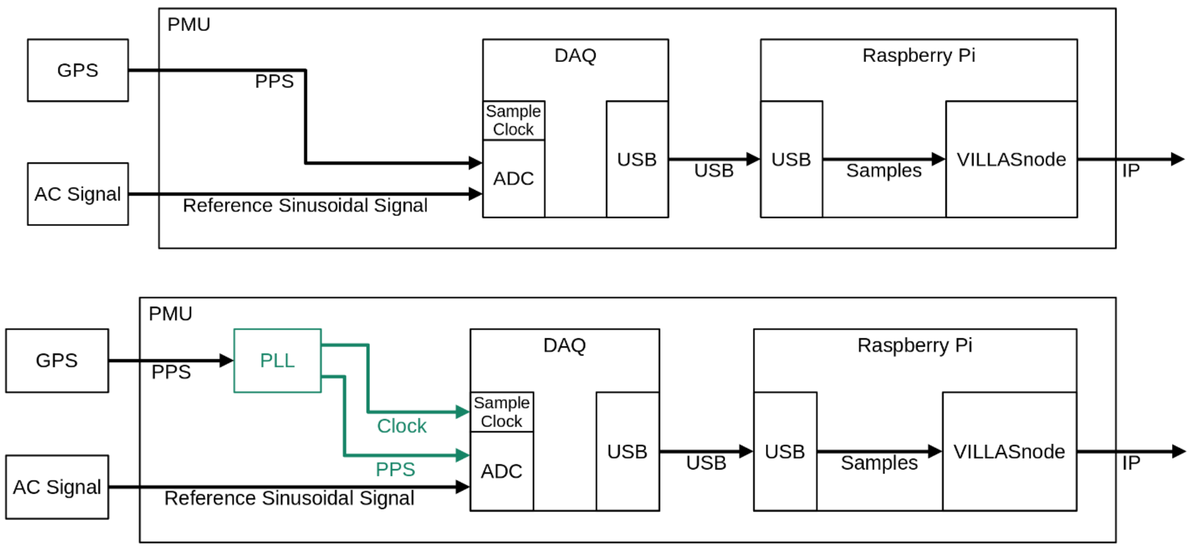


Figure 5: PPS based synchronization (Top) Hybrid synchronization (Bottom), usable for a PMU device. In this case, this approach is adapted for the 5G edgePMU.

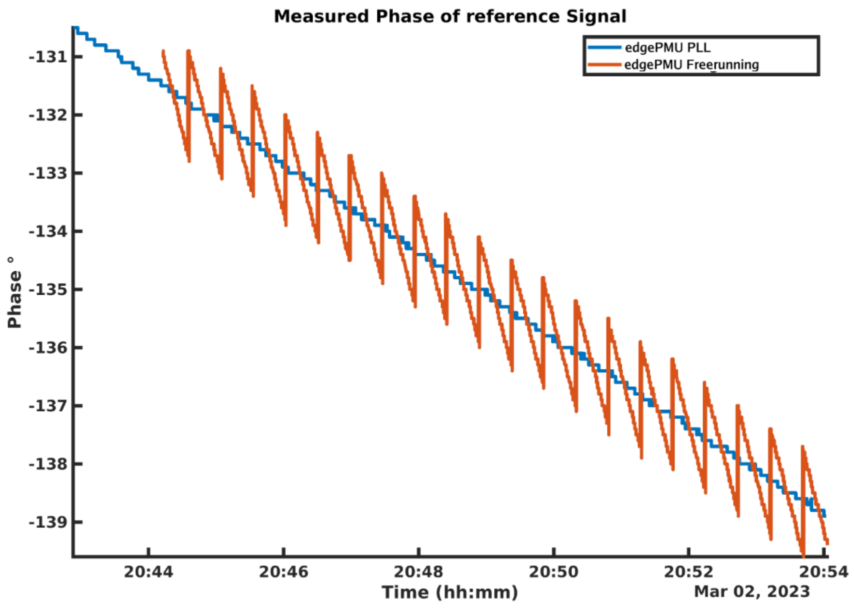


Figure 6: Comparison PPS synchronization (edegPMU Freerunning) vs. Hybrid time synchronization (edgePMU PLL).

3.2 Measurement Box

For the deployment of the above-mentioned hardware in the construction testbed, a weatherproof enclosure is needed. Furthermore, the device should be built in such a way that no electrician is needed for deployment. Therefore, it was decided to plan an outdoor-capable measurement box that can be connected in line with the device under test. In the case of the construction site, that could, for example, be a crane. More information about the construction site is given in D5.1 [3]. A first mockup of the device is shown in Figure 7. The internal setup of the device is shown in Figure 8.

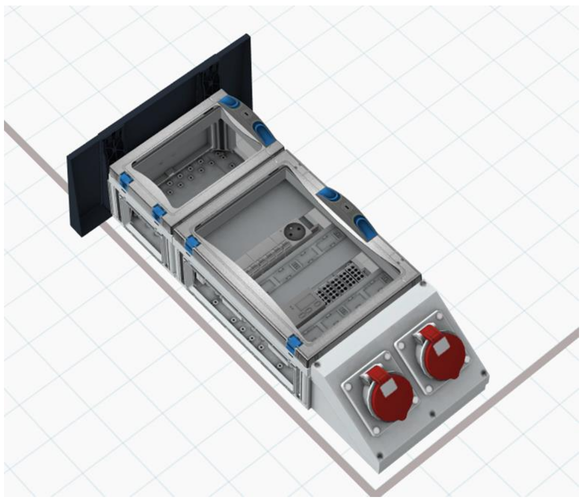


Figure 7: Mockup of measurement box for outdoor inline measurement of voltage current and energy.

The architecture of the measurement box comprises three-phase components like plugs, fuses and metering equipment, as well as low-voltage equipment like the Raspberry Pi, 5G Modem and ADC (Analog-to-Digital Converter). In the Figure 8, all connections and components marked in blue have to be able to handle three-phase grid voltages and up to 32 A of current. The connection is done via outdoor capable three-phase 32 A connectors referred to as CCE 32A in the figure. All black connections are low-voltage connections and are part of communication or analog acquisition. The connections marked green are for current sensors and in yellow are digital connections.

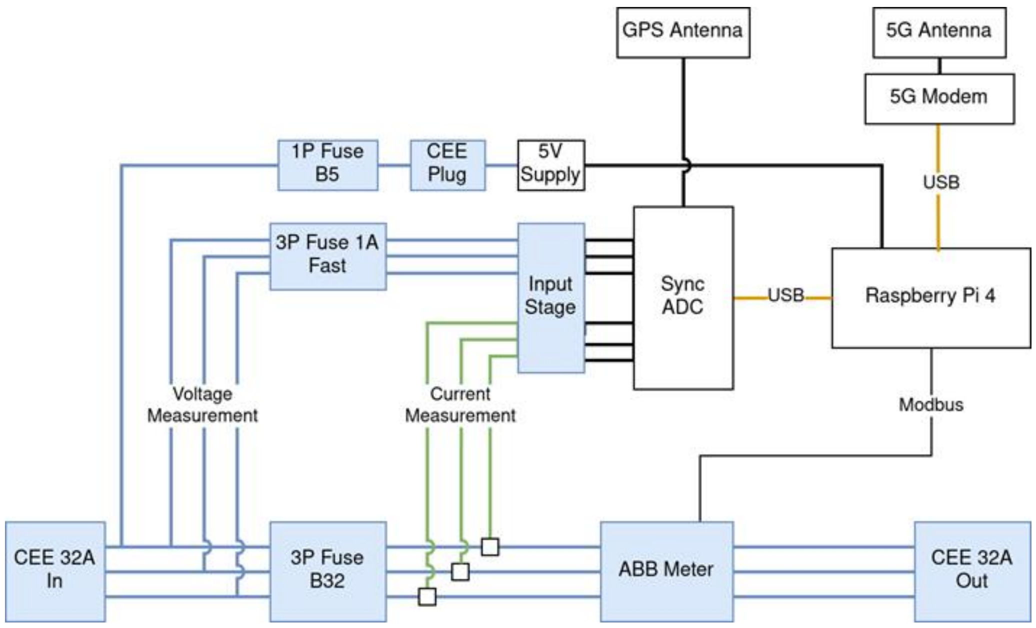


Figure 8: Architecture measurement box with the power components in blue, the low voltage components in black and the USB connections in yellow.

3.2.1 Metering Device

For energy metering, the Raspberry Pi is not well suited. Mainly, the Raspberry Pi is an operating system-based device. This means it can only handle metering while running, and metering can be interrupted due to software or hardware errors. Additionally, the Raspberry Pi would depend on a numerical integration of the energy. Especially for energy measurement, the impact of an integrative error is most important. Therefore, specialized metering hardware will be deployed. This hardware will, on the one hand, provide validated energy metering and, on the other hand, can be used for validation of the high sampling rate acquisition running on the Raspberry Pi.

For the metering, two devices were validated. The Schneider iem3155 and the ABB B23 metering devices. Both devices were evaluated against an ZES Zimmer LMG671 power analyzer. The test setup can be seen in Figure 9. On the top, the two metering devices are connected with the ZES Zimmer reference device to a Chroma AC/DC load. The Chroma 63804 (bottom) is used to provide a set power consumption. For the test, different power set points were validated. The range was limited by the used power supply and started at 5 W up to 780 W. The resulting measurements can be seen in Figure 10. It can be seen that the Schneider metering device has a higher relative error in the lower power range. Furthermore, the absolute error is also higher over most of the tested range. Therefore, the ABB device is selected for the metering application.

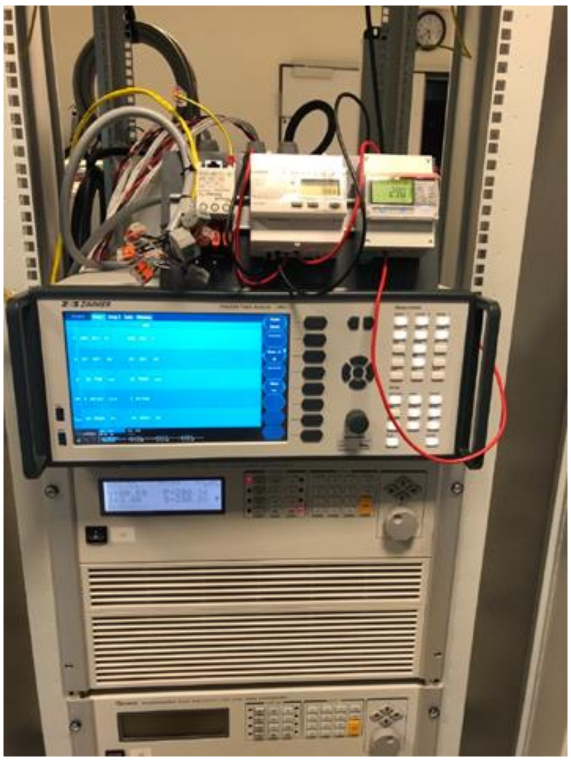


Figure 9: Energy meter validation with the two metering devices at the top, the Zimmer power analyzer in the middle and the Chroma AC/DC load on the bottom.

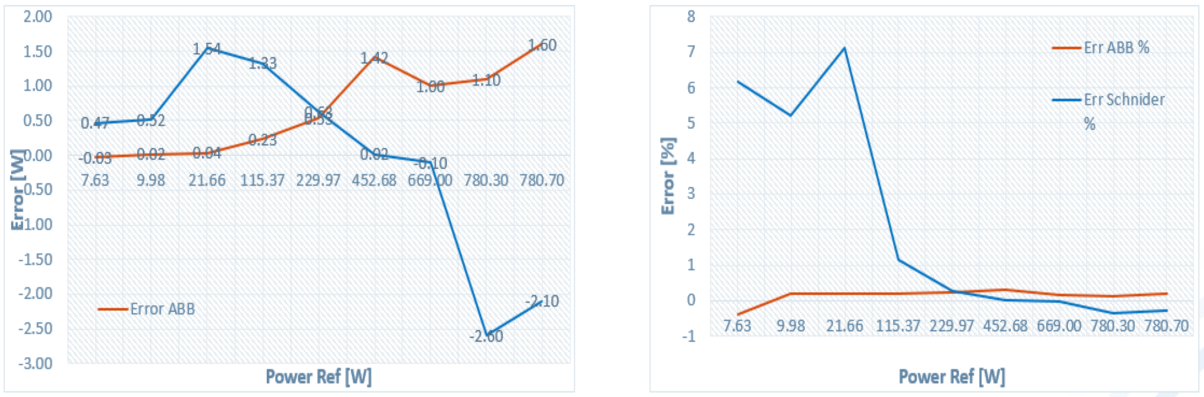


Figure 10: Measurement results for ABB and Schneider power meter with the absolute measurement error (left) and the relative measurement error (right).

3.3 VILLASframework

This chapter will introduce the different components used from the VILLASframework [9]. The VILLASframework is an open-source software set with different components that allows for data exchange between different data sources as well as the control of such nodes with a web-based UI.

3.3.1 VILLASnode

VILLASnode [5] is the core software component of the 5G edgePMU as depicted in the architecture overview in Figure 1. It allows for data translation based on so-called nodes and for data manipulation based on so-called hooks. For this project, both nodes and hooks are needed. An input node for communication with the USB data acquisition device is used and then after processing an output node is needed to forward the data to an upstream service like a database or a secondary VILLASnode as shown in Figure 11. The processing is done within a hook. An example for that could be Root Mean Square (RMS) calculation for voltages and currents or phasor estimation. For upstream communication, different types of nodes are available. The edgePMU currently allows for use of MQTT, IEC60870 [22] and IEC61850 [23] Generic Object Oriented Substation Event (GOOSE) and more. In addition, the implementation of the PMU protocol C37.118.2 [23] has started. This will allow for a PMU compliant communication and makes the edgePMU compatible with all standard compliant Supervisory Control and Data Acquisition (SCADA) and Phasor Data Concentrator systems (PDC).

For phasor estimation, an interpolated discrete Fourier Transform [22] algorithm is used. This algorithm was implemented within the edgeFLEX project [6], where the edgePMU was deployed in the distribution grid, to acquire measurements for voltage control service. The algorithm will now be extended with error compensation to allow for higher estimation precision in non-trivial cases like transients and off nominal frequency use cases.

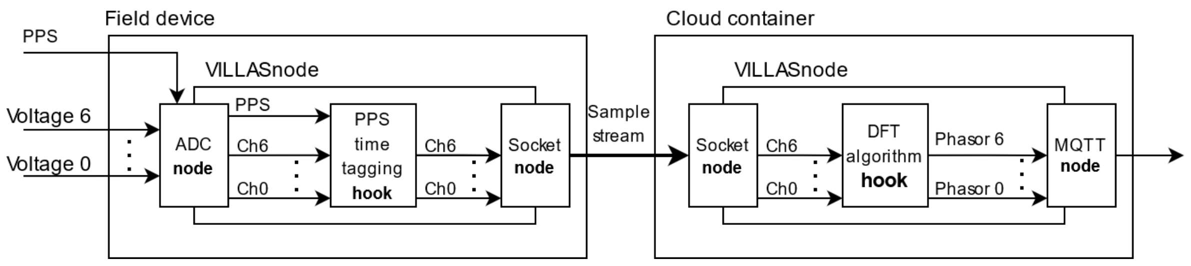


Figure 11: VILLASnode block diagram for the edgePMU with the field device on the left and the edge cloud service on the right.

3.3.2 VILLAScontroller

VILLASnode not only allows for a configuration file-based control, but can also be controlled with its own REpresentational State Transfer Application Programming Interface (REST API). This includes for example start/stop commands or configuration file selection. The tool used for this control is called VILLAScontroller [7]. This Python-based software utilizes a rabbitMQ broker [10] to exchange control data and acquire statistical information. Within this project, it is planned to utilize the

VILLAScontroller to remotely control the 5G edgePMU. To do so, the software needs to be extended and integrated into the architecture.

3.3.3 VILLASweb

VILLASweb is a web-based control application for VILLASnode. It includes remote control and visualization features. Different VILLASnode instances can be controlled directly via VILLASweb. The interface utilizes a rabbitMQ interface. The VILLASnodes can transmit statistics to the VILLASweb application as well as samples.

4 Edge Infrastructure

This chapter will introduce the different components of the edge infrastructure. First, the provisioning system is explained. Then, the VPN and considered broker software as well as the network supervision are discussed. Finally, the storage infrastructure and visualization are described.

4.1 Provisioning Service

The provisioning system is responsible for installing and updating the various software packages and drivers that are running on the external measurement devices (such as the edgePMU). It further provides the operator with a method of registering new measurement devices in the device database.

Provisioning occurs in two steps. In a first step, an image is generated to be written to an edgePMU Raspberry Pi. This image contains all configurations and software needed for the second provisioning step.

In the second provisioning step, the edgePMU is configured for its measurement operation via Ansible [1]. The edgePMU periodically sends out a request to the Ansible provisioning server, which then constructs an Ansible update package, consisting of a playbook (see Section 4.1.2) and corresponding variables. This package is transmitted to the edgePMU and run locally with the Ansible-playbook command.

Figure 12 shows an overview of the main components that are involved in deploying the software configuration from the management server to the measurement devices. The image generation is discussed in Section 4.1.1. The edgePMU web frontend, which is currently not implemented, will function as a user interface for this step. The edgePMU database, holds meta data about all clients, which are explained in Section 4.1.3. Finally, the Ansible provisioning server is detailed in Section 4.1.2.

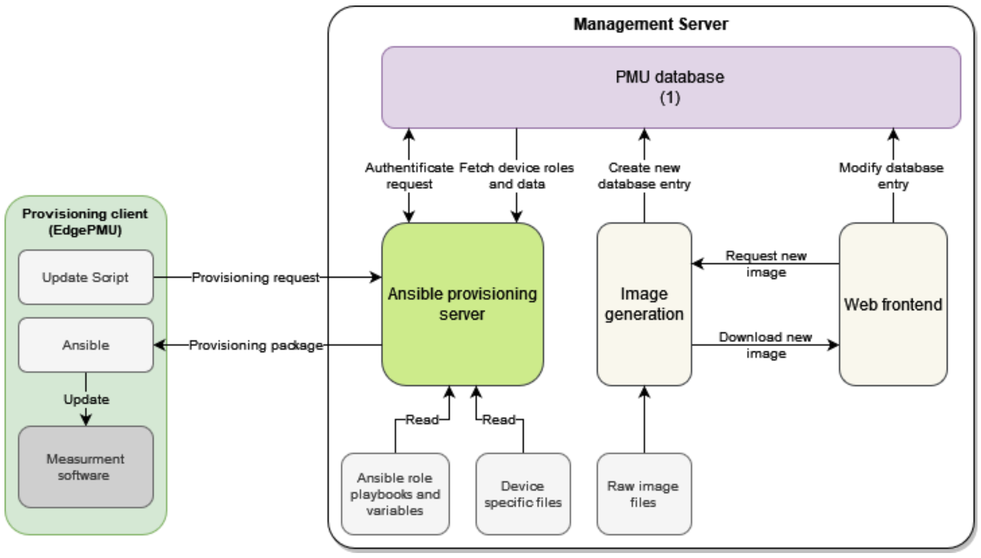


Figure 12: Provisioning Architecture with the edge cloud service for the database and provisioning on the right and the filed device on the left.

4.1.1 Image Generation

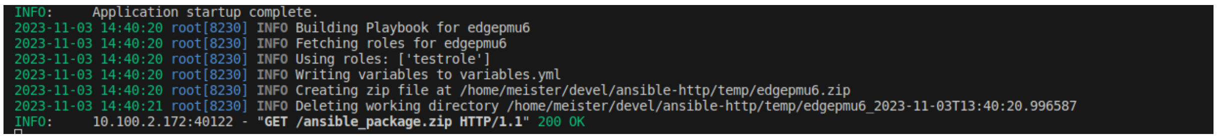
For the initial provisioning, a boot image for the edgePMU is created and written to an SD card. This image contains a pre-generated secret which is known only to the management server and the edgePMU. It further contains all software and configuration files to connect to the VPN (see Section 4.2) and perform further provisioning steps. After completing the initial provisioning, the edgePMU can provide basic functionality that is common to all devices, such as remote access via Secure Shell (SSH), a pre-defined hostname, supervision and Ansible.

4.1.2 Ansible Provisioning Server

To update the configuration of the field device the Ansible provisioning server receives a request from the provisioning client. The provisioning client, which is an external untrusted source, connects via Hypertext Transfer Protocol Secure (HTTPS). This request contains the edgePMUs Universal Unique Identifier (UUID) and the pre-defined secret from initial provisioning. After authenticating the request with the database, the provisioning server fetches the relevant configuration variables and roles from the database to construct an Ansible provisioning package.

Each Ansible provisioning package is constructed from roles. These roles are a set of tasks and variables defined in .yml files on the management servers. To amend this, it is possible to attach a file, i.e. a configuration file, to a role which is then also transmitted to the edgePMU. Finally, to enable the configuration to be device-specific, variables can be defined on a per edgePMU level. Role-specific variables as well as edgePMU specific variables are then combined and transmitted within the provisioning package.

After gathering all files, the data is encrypted using the edgePMUs secret which was generated during initial provisioning. The encrypted data is then compressed and transmitted to the client.

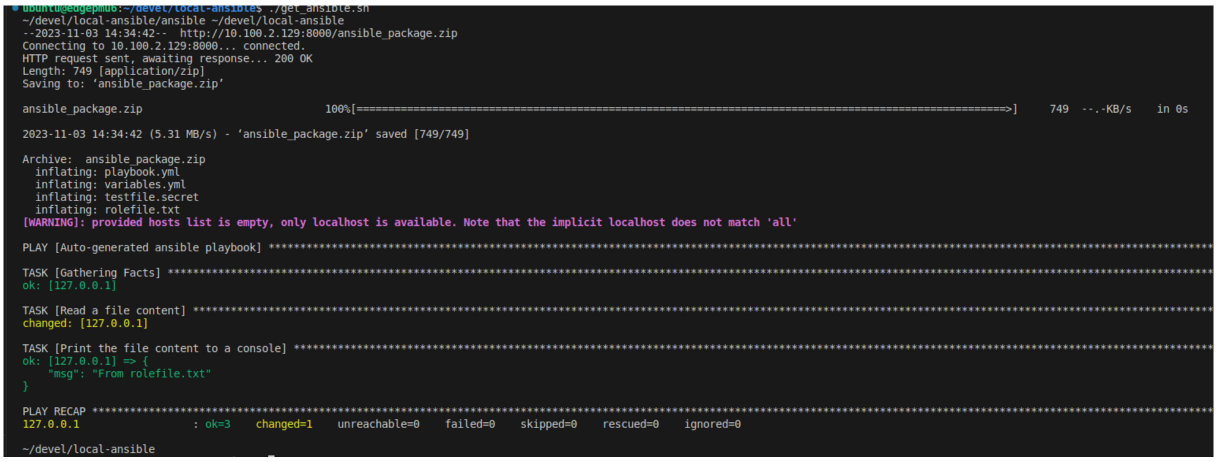


```
INFO:     Application startup complete.
2023-11-03 14:40:20 root[8230] INFO Building Playbook for edgepmu6
2023-11-03 14:40:20 root[8230] INFO Fetching roles for edgepmu6
2023-11-03 14:40:20 root[8230] INFO Using roles: ['testrole']
2023-11-03 14:40:20 root[8230] INFO Writing variables to variables.yml
2023-11-03 14:40:20 root[8230] INFO Creating zip file at /home/meister/devel/ansible-http/temp/edgepmu6.zip
2023-11-03 14:40:21 root[8230] INFO Deleting working directory /home/meister/devel/ansible-http/temp/edgepmu6_2023-11-03T13:40:20.996587
INFO:     10.100.2.172:40122 - "GET /ansible_package.zip HTTP/1.1" 200 OK
```

Figure 13: Proof of concept screenshot from provisioning server, with an 5G edgePMU request.

4.1.3 Provisioning client

The provisioning client requests a new Ansible provisioning package at regular time intervals. After receiving the provisioning package (see Section 4.1.2), the provisioning client unzips the file. It then runs the Ansible playbook using the Ansible-playbook command. This command downloads or updates the software and executes commands for installation and configuration (see Figure 14).



```
ubuntu@edgepmu6:~/devel/local-ansible$ ./get_ansible.sh
~/devel/local-ansible/ansible ~/devel/local-ansible
--2023-11-03 14:34:42--  http://10.100.2.129:8000/ansible_package.zip
Connecting to 10.100.2.129:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 749 [application/zip]
Saving to: ‘ansible_package.zip’

ansible_package.zip                        100%[==================================================================================================>]     749  --.-KB/s    in 0s

2023-11-03 14:34:42 (5.31 MB/s) - ‘ansible_package.zip’ saved [749/749]

Archive:  ansible_package.zip
  inflating: playbook.yml
  inflating: variables.yml
  inflating: testfile.secret
  inflating: rolefile.txt
[WARNING]: provided hosts list is empty, only localhost is available. Note that the implicit localhost does not match 'all'

PLAY [Auto-generated ansible playbook] ***************************************************************************************************************************

TASK [Gathering Facts] *******************************************************************************************************************************************
ok: [127.0.0.1]

TASK [Read a file content] ***************************************************************************************************************************************
changed: [127.0.0.1]

TASK [Print the file content to a console] ***********************************************************************************************************************
ok: [127.0.0.1] => {
    "msg": "From rolefile.txt"
}

PLAY RECAP *******************************************************************************************************************************************************
127.0.0.1                  : ok=3    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

~/devel/local-ansible
```

Figure 14: Execution of the Ansible playbook on a 5G edgePMU.

4.2 Virtual Private Network

Using a Virtual Private Network (VPN) is a common technique to secure communication and allow for access to devices that are running in the field and cannot be reached via a public IPv4 or Ipv6 address. In the case of the 5G edgePMU, the VPN is used to connect the 5G edgePMU securely to the edge cloud infrastructure and allows for remote control of the field device. This supports manual intervention in case of software errors or for testing during deployment. In the current version of the software stack, OpenVPN [11] is used as a VPN solution. OpenVPN is an established VPN solution and clients for most operating systems, including Linux, Windows, macOS as well as Android and iOS are available. The main disadvantage is that OpenVPN is running as a user space application and thus is not the most performant solution. Furthermore, the configurations can be complex and are not fully downwards compatible. Therefore, it is planned to evaluate more recent VPN implementations like WireGuard [12] WireGuard stands out as it is implemented as a kernel module and thus much more performant than 22 OpenVPN. Even so, WireGuard is relativity new (initial release 2015), it is already well established and implementations for Linux, Windows or Android and iOS are available.

4.3 Broker (MQTT/Kafka)

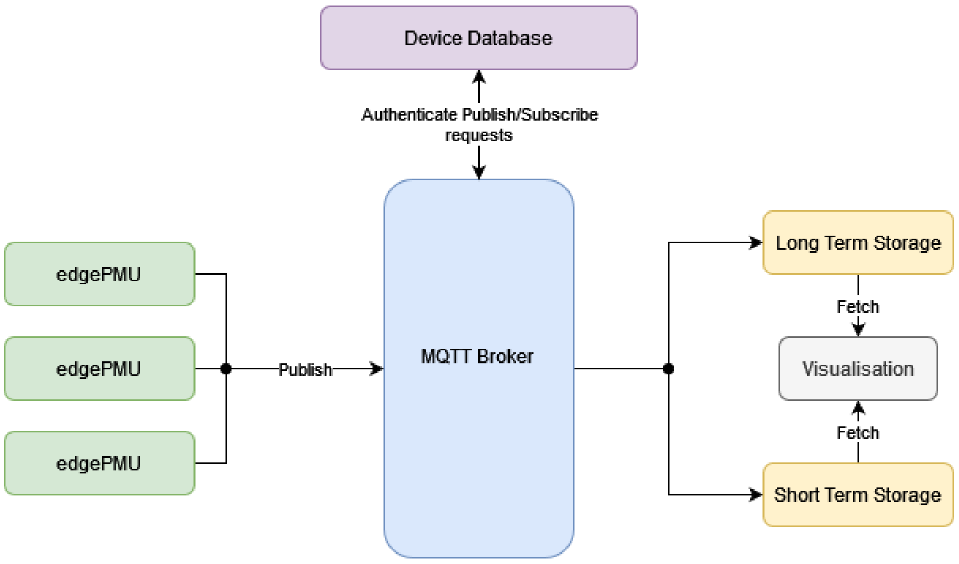


Figure 15: High-level data transmission architecture with field devices on the left and edge cloud services with broker, database and storage services on the right.

The broker is a server that controls the data exchange/distribution between PMUs and other devices. In our case, of the 5G edgePMU as shown in Figure 15, the broker is used to provide a secure data flow from the 5G edgePMUs to the data consumers, i.e. data flow to the long-term and short-term data storage and device database. For more information on the storage solution, see Section 4.5. Currently, both MQTT Broker and Kafka are considered to be used. Both design options will be examined in more detail in the following subsections.

4.3.1 Design option Broker 1: MQTT Broker

MQTT is a lightweight messaging protocol that is designed for limited bandwidth, high latency and low reliability networks [13], [14]. For real-time monitoring of edge devices like PMUs, it is important to have low-latency messaging by the protocol which can be fulfilled by MQTT. MQTT brokers are simple to set up and to use while handling numerous devices, which makes it a good choice for data transmission from edge devices to a storage or device database. However, MQTT brokers typically do not retain messages for an extended period, which may be a disadvantage if we need data history in some cases for analysis.

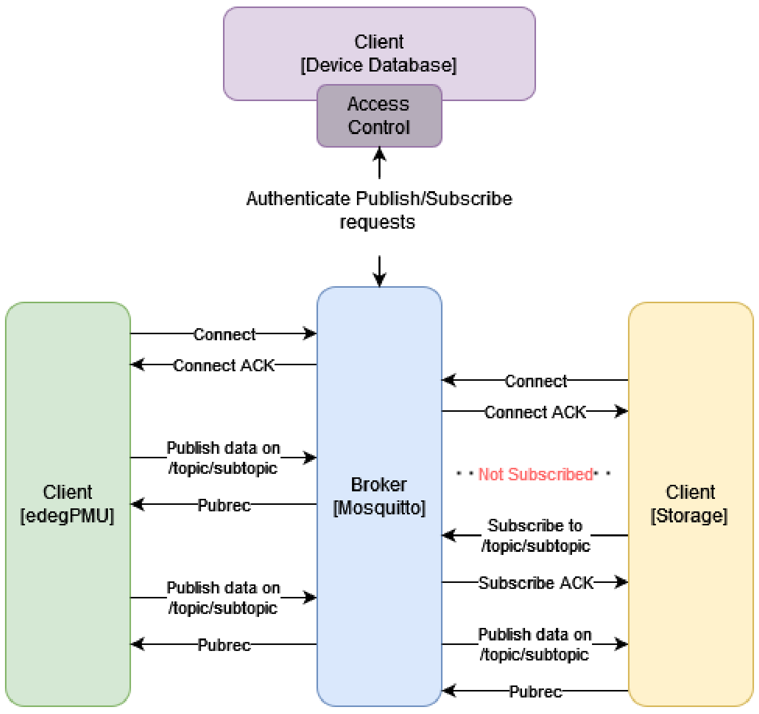


Figure 16: Overview of Mosquitto broker architecture for edgePMU data transmission case with the different states of communication.

In this case, the MQTT protocol can be used for communication between edgePMU devices and data storage or device database. MQTT brokers follow the Publish/Subscribe Architecture, where the Publisher/Client publishes the messages on a topic, and the Subscriber/Client receives those messages by subscribing to that topic [12]. The Publisher, 5G edgePMU will publish data to the MQTT broker on a topic, and the Subscribers like storage and device database will receive this data by subscribing to the specific topic as shown in Figure 14. In our case, of 5G edgePMU data transmissions, Mosquitto is used as MQTT broker as shown in Figure 15. The edgePMU, storage and device database as clients create a connection with the Mosquitto Broker. After creating a connection, the Mosquitto replies with a connection acknowledgement (Connect ACK) to each of them. Then, as per requirement, Mosquitto sends a data publishing request (Pubrec) to the Publisher edgePMU. In result, edgePMU publishes data to Mosquitto on a topic. On the other side, Subscribers like storage and device database as a client subscribe to the specific topic on Mosquitto. So, in return, the Subscribers get the subscription acknowledgment (Subscribe ACK) from Mosquitto. Further, as per need, Subscribers request for edgePMU data publishing (Pubrec) from Mosquitto. And as an outcome, Mosquitto publish the required data from edgePMU with that specific topic to the Subscribers.

4.3.2 Design option Broker 2: Kafka

Kafka is an open-source distributed data store and streaming platform, optimized for ingesting and processing large amounts of streaming data in real-time. Kafka [16] is based on a number of servers called brokers and clients communicating through a TCP network protocol. Kafka is designed as an event streaming platform, capable of handling high-throughput, fault-tolerant data streams. It is well-suited for scenarios where data must be ingested, processed, and retained over time. Key-value messages which come from many random processes known as producers are stored by Kafka. Here the data can be divided into different partitions within various topics. Kafka processes data in both publish-subscribe and log-based methods. PMUs can publish data to Kafka topics, and consumers can subscribe to these topics. Within Kafka data can be retained for a specified duration. Kafka provides built-in data transformation and processing capabilities through Kafka Streams and Kafka Connect, allowing the manipulation of data as it flows through the system. However, Kafka can be more complex to set up and manage as compared to MQTT, and it typically requires more resources.

4.4 Infrastructure Supervision

Enabled by the distributed nature of the infrastructure, an efficient monitoring of the components is key to detecting faults and ensuring stable operation. This includes keeping track of the health status of the network and all connected clients. It further remains important to keep metadata about the connected clients, such as location, contact person or organization.

The Simple Network Management Protocol (SNMP) is a widely used protocol to supervise the state of multiple agents from a single central manager by polling. The manager periodically sends a request which is executed by the agent and a value is returned. The returned value is then logged by the manager in the Management Information Base (MIB) to be displayed in a graphical user interface. These requests can be amended by custom data, such as the agent's location.

For the supervision of the edge-cloud infrastructure and the deployed edgePMUs, Observium has been selected. Observium [18] is mostly automatic SNMP data visualization tool, based on open-source software. Figure 17 shows the dashboard with an overview of the device locations. Features of this software include:

- Automated detection of available SNMP device features
- Visualization of historic data for the different features
- Alerting in case of errors (e.g. resource allocation maximum)
- Alerting in case of unreachable device
- Device grouping
- Geographical representation of devices

To add a device in the supervision infrastructure, the SNMP role can be assigned to the device during provisioning. This will install the SNMP daemon on the device and configure the device to respond as an agent to requests from Observium. To automatically register a device with Observium further development is needed.

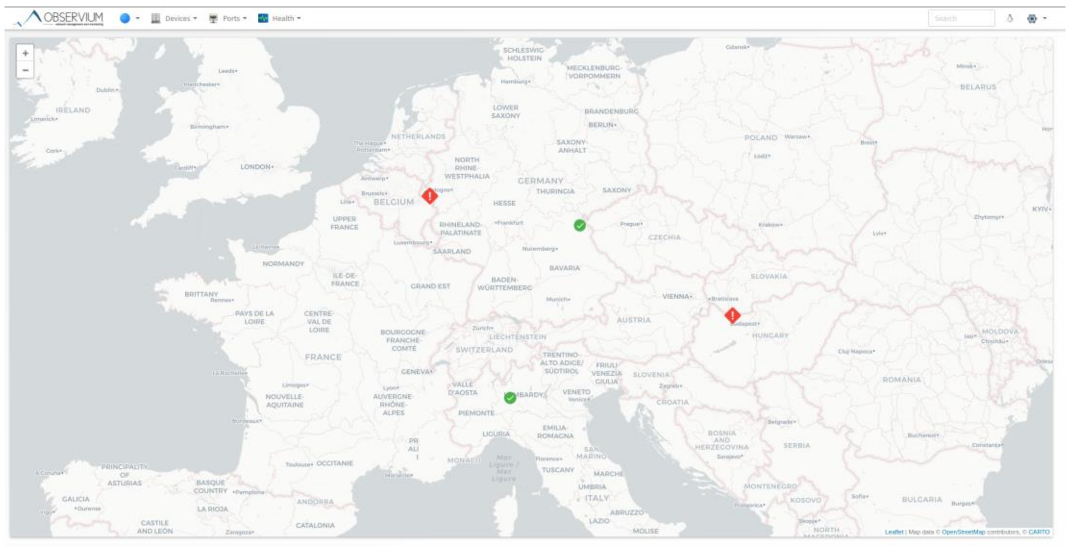


Figure 17: Observium Interface showing the different devices on a map of Europe.

4.5 Storage

The storage solution is a key element of making energy data available to the project partners but also allows for later reuse, for example in machine learning applications. To achieve the data storage, a storage server was procured. It was decided to buy a DELL PowerEdge R550 server equipped with 8 X 16 TB hard drives. As operating system TrueNAS Scale [19] was chosen. This enables a RAIDz2 [20] with a usable storage capacity of 82 TB. It is expected that the actual storage capacity is higher since Zettabyte File System (ZFS) allows for native LZ4 compression, and it is expected that the energy data will allow for a high compression ratio. A screenshot of the installed TrueNAS Scale is given in Figure 18.

The above-mentioned storage server will be utilized for short as well as long-term storage. To allow for future extension, an S3 storage solution will be evaluated. This will allow for upscaling in case the available storage is not sufficient. The two topics of short-term and long-term storage will be further discussed in the following subchapters.

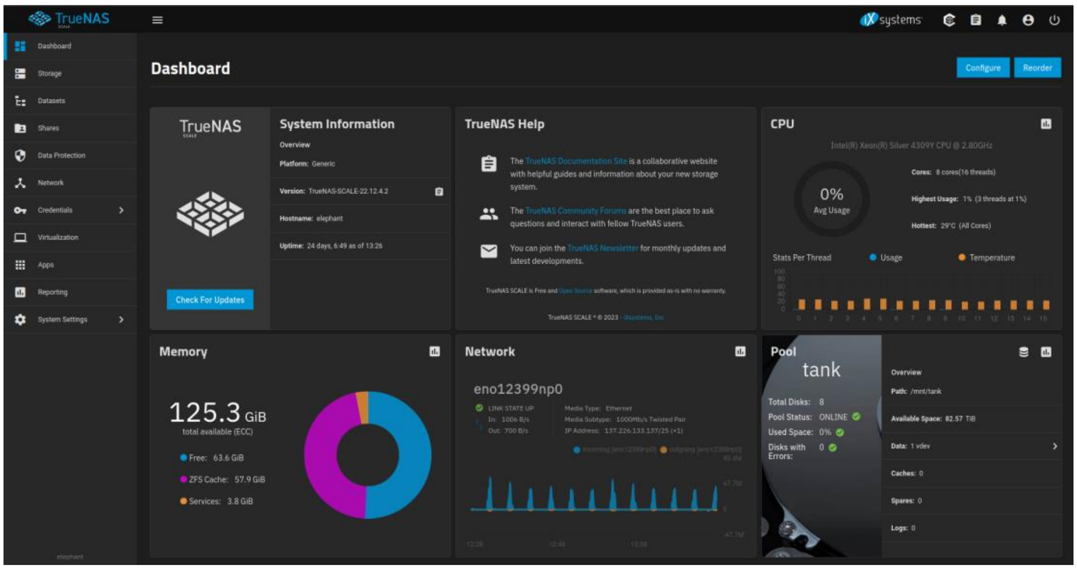


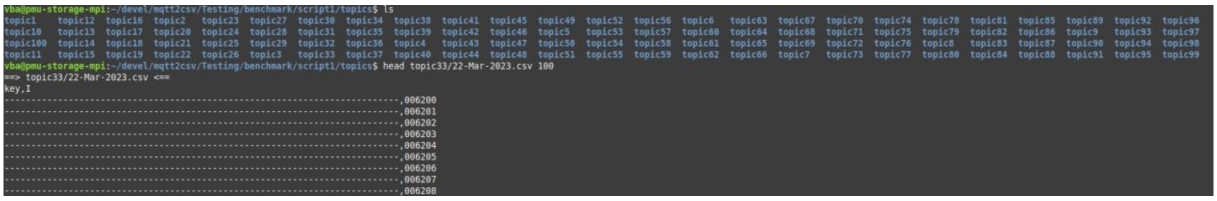
Figure 18: Screenshot of TrueNAS Scale installation on storage server.

4.5.1 Short-Term Storage

Short-term storage allows the user direct access to the data via a web User Interface (UI). It is planned to provide access to data for the past weeks or months. This is highly dependent on the chosen database infrastructure. For the current software stack, InfluxDB was utilized. It was seen that scalability for this approach is limited by the performance of InfluxDB. Therefore, it is planned to evaluate other time series databases for the short-term storage. The major requirements are fast access to data and the capability to automatically remove data after a certain timespan.

4.5.2 Long-Term Storage

The long-term storage server, functions as an archive for previous measurement data as well as an easy access point for further processing of the acquired data. All operations of the long-term storage server take place decoupled from the management server's operations. For this reason, the long-term storage is located on an entirely different server than the main management tasks. While running, all configured broker topics are continuously logged into the file system. This has been demonstrated with a proof of concept, logging 100 topics, each with a data-rate of 100 kBps (see Figure 19), with further improvements possible.



```
vba@pmu-storage-mpi:~/devel/mqtt2csv/Testing/benchmark/script1/topics$ ls
topic1    topic12  topic16  topic2   topic23  topic27  topic30  topic34  topic38  topic41  topic45  topic49  topic52  topic56  topic6   topic63  topic67  topic70  topic74  topic78  topic81  topic85  topic89  topic92  topic96
topic10   topic13  topic17  topic20  topic24  topic28  topic31  topic35  topic39  topic42  topic46  topic5   topic53  topic57  topic60  topic64  topic68  topic71  topic75  topic79  topic82  topic86  topic9   topic93  topic97
topic100  topic14  topic18  topic21  topic25  topic29  topic32  topic36  topic4   topic43  topic47  topic50  topic54  topic58  topic61  topic65  topic69  topic72  topic76  topic8   topic83  topic87  topic90  topic94  topic98
topic11   topic15  topic19  topic22  topic26  topic3   topic33  topic37  topic40  topic44  topic48  topic51  topic55  topic59  topic62  topic66  topic7   topic73  topic77  topic80  topic84  topic88  topic91  topic95  topic99
vba@pmu-storage-mpi:~/devel/mqtt2csv/Testing/benchmark/script1/topics$ head topic33/22-Mar-2023.csv 100
==> topic33/22-Mar-2023.csv <==
key,I
------------------------------------------------------------------------,006200
------------------------------------------------------------------------,006201
------------------------------------------------------------------------,006202
------------------------------------------------------------------------,006203
------------------------------------------------------------------------,006204
------------------------------------------------------------------------,006205
------------------------------------------------------------------------,006206
------------------------------------------------------------------------,006207
------------------------------------------------------------------------,006208
```

Figure 19: Long term storage proof of concept showing 100 topics and the running benchmarking application.

The active log files are rotated daily with old log files being compressed to reduce disk usage. Currently, two file formats are being explored for the final storage.

CSV: Comma Separated Values are a textual representation of a table format. Each column is separated by a separator character, usually a comma, and rows are separated by a new line. This simple representation makes CSV files easy to parse and human-readable. For this reason, CSV files require more disk space than a binary representation but also allow for easy processing.

HDF5: The Hierarchical Data Format has been developed to store large amounts of data in a file system like container. Values are stored in data frames in a binary format and accessed through a library. This decreases the file size drastically, while increasing the effort to extract the data for subsequent users. In addition, HDF5 can store metadata alongside the actual data frames.

4.6 Visualization

The visualization is the final puzzle piece that allows the user to interact with the proposed architecture. In the above-mentioned architecture, three different types of visualization are needed. The visualization of historical measurement data, configuration and control information and real-time measurement data. The three aspects will be discussed in the following paragraphs.

Historical Measurement Data

For visualizing historic data, Grafana is a well-established choice. For the current revision of the edgePMU architecture, Grafana is already used. In the updated deployment, this Grafana interface will be extended with an access policy to only allow for access to specific datasets. Additionally, it is planned to set up a template like approach for the dashboards. This would allow for easy generation of dashboards for different field devices. An example of the Grafana UI is shown in Figure 20. This is an example already showing PMU measurements, but all plots had to be configured manually. This is not feasible when deploying multiple devices in the field.

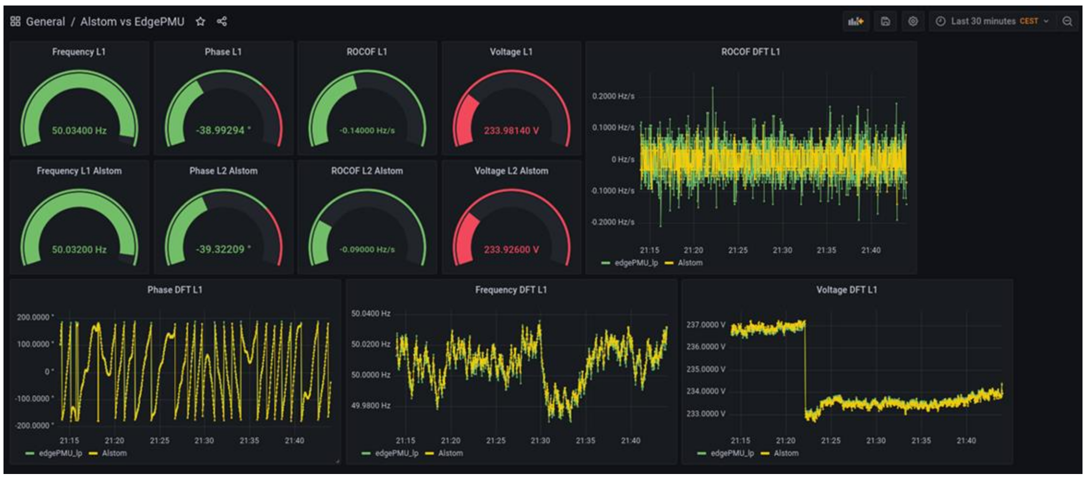


Figure 20: Example Grafana UI showing PMU measurements, currently with a manually configured interface.

Visualization of Configuration and Control

This aspect of the visualization is not yet implemented but will provide a management interface for the databases that control the deployment of the edgePMU device. This includes setting the configuration for the field devices and registering new devices.

Visualization of Real-Time Measurements

Grafana is well suited for the visualization of data points from a Database. Due to the underlying concept, Grafana relies on loading all data points of a visualization every time the visualization is updated. This is especially inefficient when visualizing large amounts of real-time data. Furthermore, the main use case for Grafana is reading data, not sending an action to a client application. It is possible to enable such features, but the integration could raise issues. Due to these reasons, VILLASweb is developed. VILLASweb [8] is an open-source project that is part of the VILLASframework. It allows for web socket-based streaming of data and provides seamless integration with VILLASnode. Thus, it is well suited for this architecture. Further development will be done to extend VILLASweb within this project.

5 Conclusions

In conclusion, this deliverable provides insights about the planned architecture. The different aspects of the edgePMU hardware development have been described. Further development is needed to transfer the prototype developments to an integrated solution. The developed hardware prototypes already show promising results.

For the edge cloud software platform, first prototype software has been developed and described. Most of the components are currently in a proof-of-concept state and will be integrated within the project duration. The major components of the architecture are the automated deployment and management of the field devices, the storage and supervision infrastructure and finally the visualization. The proposed architecture provides a holistic approach to the general question of measurement acquisition and archival. This is the first version of two deliverables, and thus it is labeled as a draft that will be updated in the upcoming deliverable D3.3.

6 References

[1] https://www.ansible.com/
[2] https://target-x.eu/wp-content/uploads/2023/06/TARGET-X-D3.1-V1.0-Pilot-implementation-plan.pdf
[3] https://target-x.eu/wp-content/uploads/2023/10/TARGET-X_D5.1_Roadmap-for-the-5G-6G-empowered-deconstruction-robotic-platform.pdf
[4] https://www.simcom.com/product/SIM8200EA_M2.html
[5] https://villas.fein-aachen.org/docs/node/
[6] https://www.edgeflex-h2020.eu/
[7] https://villas.fein-aachen.org/docs/controller/
[8] https://fein-aachen.org/projects/villas-web/
[9] https://villas.fein-aachen.org/docs/
[10] https://www.rabbitmq.com/
[11] https://openvpn.net/
[12] https://www.wireguard.com/
[13] Soni, Dipa, and Ashwin Makwana. "A survey on mqtt: a protocol of internet of things (iot)." *International conference on telecommunication, power analysis and computing techniques (ICTPACT-2017)*. Vol. 20. 2017.
[14] P. Brogan et al., "MQTT Architecture for Stream Analytics of PMU Data," 2021 32nd Irish Signals and Systems Conference (ISSC), Athlone, Ireland, 2021, pp. 1-7, doi: 10.1109/ISSC52156.2021.9467849.
[15] https://www.hivemq.com/blog/mqtt-essentials-part2-publish-subscribe/
[16] https://kafka.apache.org/documentation/
[17] A. Hugo, B. Morin and K. Svantorp, "Bridging MQTT and Kafka to support C-ITS: a feasibility study," 2020 21st IEEE International Conference on Mobile Data Management (MDM), Versailles, France, 2020, pp. 371-376, doi: 10.1109/MDM48529.2020.00080.
[18] https://www.observium.org/
[19] https://www.truenas.com/
[20] https://www.raidz-calculator.com/raidz-types-reference.aspx
[21] https://grafana.com/
[22] https://ieeexplore.ieee.org/document/7980868
[23] "IEEE Standard for Synchrophasor Data Transfer for Power Systems," in *IEEE Std C37.118.2-2011 (Revision of IEEE Std C37.118-2005)* , vol., no., pp.1-53, 28 Dec. 2011, doi: 10.1109/IEEESTD.2011.6111222.
[24] https://iec61850.dvl.iec.ch/