



Demonstration of **5G** solutions for **SMART** energy **GRIDs** of the future

Deliverable D.4.1

Development and deployment of NetApps for the energy
vertical sector

Version 1.0 - Date 31/03/2023



D4.1 – Development and deployment of NetApps for the energy vertical sector

Document Information

Programme	Horizon 2020 Framework Programme – Information and Communication Technologies
Project acronym	Smart5Grid
Grant agreement number	101016912
Number of the Deliverable	D4.1
WP/Task related	WP4
Type (distribution level)	PU Public
Date of delivery	31-03-2023
Status and Version	Version 1.0
Number of pages	81 pages
Document Responsible	Theocharis Saoulidis (SID)
Author(s):	Achilleas Moukoulis (SID) Andreas Foteas (AXON) Antonios Sarigiannidis (SID) Dimitrios Margounakis (SID) Eleftherios Fountoukidis (SID) Elisavet Grigoriou (SID) Guillermo Gomez Chavez (ATOS) Hélio Simeão (UW) Ioannis Hatjigeorgiou (SID) Konstantinos Kyranou (SID) Kostas Chrysagis (SID) Paris Alexandros Karypidis (SID) Plamen Tonchev (SC) Sonia Castro Carrillo (ATOS) Theocharis Saoulidis (SID)

Thomai Karamitsou (SID)

Valentin Velev (SC)

Reviewers:

Hélio Simeão (UW)

Michalis Rantopoulos (OTE)

Revision History

Version	Date	Author/Reviewer	Notes
0.1	02/2022	Theocharis Saoulidis	Initial structure of content and feedback
0.2	05/2022	All authors	Structure change
0.3	10/2022	All authors	First round of contributions
0.5	01/2023	All authors	Second round of contributions
0.7	02/2023	All authors	Final round of contributions
0.9	03/2023	Hélio Simeão, Michalis Rantopoulou	Peer review
1.0	03/2023	Theocharis Saoulidis	Final version available

Executive summary

To address the Business-to-Business (B2B) opportunities of 5G, collaboration between Mobile Network Operators (MNOs), software providers and vertical customers is essential in fostering what currently is an evolving ecosystem. Particularly looking at the Energy domain, 5G networks represent a platform on which software solutions can run, delivering new functionalities to smart grids, with more computational resources at the edge, closer to the grid equipment.

This document focuses on the software provider side, by depicting the concept of the Network App. Network Apps can be seen as an abstraction that helps developers to overcome the complexity of developing applications that run in the 5G infrastructure, using a service-based architecture. Two models for the Network Apps were explored, a purely Cloud-Native (CN) application and an OSM-compatible Network App. These two options are relevant since currently MNOs are transitioning, at different speeds, from a 5G Non-Standalone (5G NSA) architecture towards a 5G Standalone (5G SA). With these two models we cover most of the current 5G public network scenarios.

A guide explaining how a Network application is developed is provided, with the several steps that were followed by the Smart5Grid partners during the development of the Network Apps for the 4 Use Cases (UC) of the project. A brief explanation about the interactions between the Network App and the Smart5Grid platform, is also provided, covering several stages from the development, packaging, verification and validation, deployment, operation and termination of a Network App.

Further resources and examples are provided in each specific UC sections, where the Network Apps deployed in each UC are described in detail, giving important information to interested developers on how all the theory of previous chapters was applied in live deployments.

Developers can and should use this document as an entry point to what is being done in Smart5Grid, particularly in the WP4 Technology Readiness Evolution for Network Applications Development and how to deliver software solutions that make use of 5G networks and which services Transmission System Operators (TSOs) and Distributed Network Operators (DSOs) are looking for.

Table of contents

Revision History.....	4
Executive summary	5
Table of contents.....	6
List of figures.....	8
List of tables	11
Notations, abbreviations and acronyms	12
1. Introduction	13
1.1. Scope of the document.....	13
1.2. Document Structure	13
2. Smart5Grid Open 5G Platform.....	14
2.1. Smart5Grid network application vision/definition.....	15
2.2. Smart5Grid Network Application lifecycle in Smart5Grid Platform.....	16
2.2.1. Network Application Design	17
2.2.2. Network Application Validation and Verification	17
2.2.3. Network Application Deployment.....	17
2.2.4. Network Application Operation.....	17
2.2.5. Network Application Termination	18
3. How to develop a Network Application. A Guide for developers	19
4. UC#1 Network Application(s)	31
4.1. Technical specification	31
4.2. Functional and Technical Requirements.....	32
4.3. Network Application Functionality	32
4.4. Integration & Deployment.....	33
5. UC#2 Network Application(s)	39
5.1. Technical specification	39
5.2. Functional and Technical Requirements.....	39
5.3. Network Application Functionality	40
5.4. Integration & Deployment.....	41
6. UC#3 Network Application(s)	43
6.1. Technical specification	43
6.2. Functional and Technical Requirements.....	43

6.2.1.	RTPM	43
6.2.2.	PM	44
6.3.	Network Application Functionality	45
6.3.1.	Common functionality of the services.....	45
6.3.2.	MQTT Broker service.....	50
6.3.3.	RTPM service	50
6.3.4.	Predictive Maintenance Enabler service	52
6.4.	Integration & Deployment.....	56
6.4.1.	Docker images of the services	56
6.4.2.	Helm charts of the services.....	57
6.4.3.	Network App descriptor.....	61
6.4.4.	Deployment of network application in the NetAppController	62
7.	UC#4 Network Application(s)	64
7.1.	Technical Specification.....	64
7.2.	Functional and Technical Requirements.....	64
7.2.1.	vPDC	64
7.2.2.	WAM Service	65
7.2.3.	Advisory Service	65
7.3.	Network Application Functionality	66
7.3.1.	Common functionality of the services.....	66
7.3.2.	vPDC service.....	68
7.3.3.	WAM service	68
7.3.4.	Advisory service.....	69
7.4.	Integration & Deployment.....	72
7.4.1.	Docker images of the services	72
7.4.2.	Helm charts of the services	74
7.4.3.	Network application descriptor	77
7.4.4.	Deployment of network application in the NetAppController	77
8.	Conclusions.....	79
9.	References.....	80

List of figures

Figure 1. Smart5Grid Open 5G platform High-level Architecture	14
Figure 2. Basic Network App representation	16
Figure 3. Network Application lifecycle developer's point of view	17
Figure 4. Network Application technologies and artefacts	19
Figure 5. Source code directory example for python application	22
Figure 6. Source code directory for Helm chart	23
Figure 7. Helm chart standardized fields	24
Figure 8. Telco Network App's VNF descriptor example (left) and VNF package structure (right)	25
Figure 9. Telco Network App's NS example (left) and package structure (right)	26
Figure 10. Telco Network Application example	28
Figure 11. Cloud-Native Network Application example	29
Figure 12 UC1 Network Application - High level architecture solution	31
Figure 13. UC1 Network Application - Folder tree	34
Figure 14. UC2 Network Application - Kubernetes services	41
Figure 15. UC2 Network Application - VNF Descriptor with helm chart package integration	42
Figure 16. UC3 Network Application Architecture	43
Figure 17. UC3 Network Application - Login Interface	47
Figure 18. UC3 Network Application - Service state	48
Figure 19. UC3 Network Application - List of devices (RTPM)	48
Figure 20. UC3 Network Application - Default IoT icon (RTPM)	49

Figure 21. UC3 Network Application - Device Status interface	50
Figure 22. UC3 Network Application - Wind turbine visualization (Energy production signals)	51
Figure 23. UC3 Network Application - Wind turbine visualization (Environment signals).....	51
Figure 24. UC3 Network Application - Wind turbine visualization (Technical signals)	52
Figure 25. UC3 Network Application - Select signals visualization.....	53
Figure 26. UC3 Network Application - Historical signal data visualization	54
Figure 27. UC3 Network Application - Export Data graphical interface	55
Figure 28. UC3 Network Application - Graph from exported data.....	55
Figure 29. UC3 Network Application - Alarm page user interface	56
Figure 30. UC3 Network Application - Folder and files output of helm command	58
Figure 31. UC3 Network Application – YAML(values) file edit.....	58
Figure 32. UC3 Network Application – YAML(deployment) file edit.....	60
Figure 33. UC3 Network Application – YAML(service) file edit.....	61
Figure 34. UC3 Network Application – Service Designer (Selecting of Network App).....	62
Figure 35. UC3 Network Application – Service Designer (Selecting deployment site)	63
Figure 36. UC3 Network Application - Services interface	63
Figure 37. UC4 Network Application Architecture	64
Figure 38. UC4 Network Application - List of devices per service interface	67
Figure 39. UC4 Network Application - Device properties per service interface	67
Figure 40. UC4 Network Application - Service state interface.....	68
Figure 41. UC4 Network Application - vDPC Statistics for each PMU device interface	68

Figure 42. UC4 Network Application - WAM service, PMUs data visualization interface	69
Figure 43. UC4 Network Application - Select signal interface	70
Figure 44. UC4 Network Application - Visualization of selected signal(s) interface	70
Figure 45. UC4 Network Application - Export data for selected signal(s) interface	71
Figure 46. UC4 Network Application - Alarms interface	71
Figure 47. UC4 Network Application - Alarms log	72
Figure 48. UC4 Network Application - Folder and files output of helm command	74
Figure 49. UC4 Network Application – YAML(values) file edit	75
Figure 50. UC4 Network Application – YAML(deployment) file edit	76
Figure 51. UC4 Network Application – YAML(service) file edit	76
Figure 52. UC4 Network Application – Service Designer (Selecting of Network App)	77
Figure 53. UC4 Network Application – Service Designer (Selecting deployment site)	78
Figure 54. UC4 Network Application - Services interface	78

List of tables

Table 1: Acronyms list.....	12
-----------------------------	----

Notations, abbreviations and acronyms

Item	Description
B2B	Business-to-Business
CLI	Command Line Interface
IM	Information Model
K8s	Kubernetes
M&O	Management & Orchestration
NAC	Network Application Controller
NFVO	Network Functions Virtualization Orchestration
NS	Network Service
OSR	Open Service Repository
PMUs	Phasor Measurement Units
SLO	Service Level Objectives
UC	Use Case
V&V	Validation & Verification
VNF	Virtual Network Function
WP	Work Package

Table 1: Acronyms list

1. Introduction

1.1. Scope of the document

As stated in the executive summary, D4.1 aims mostly at software providers/developers by providing an extensive guide on how to develop a Network Application by using specific steps. These steps are later mapped in each Use Case (UC) and the Network Applications that were developed to fit the needs and requirements of those UCs. That guide was followed from all of the developers of the project to proper adjust the smooth deployment.

The Smart5Grid Open 5G platform provides the software providers a way to be able to quickly attain their services which will result in faster and more reliable deployments. The testing and validation of their applications by accessing the Smart5Grid services, systems and tools on demand will be made under a controlled environment, with replicable conditions, and in accordance with specific requirements.

This deliverable covers the activities performed as part of Work Package (WP) 4 and specifically the tasks “Task 4.1 Development of use case specific NetApps”, “Task 4.2 NetApp deployment over the open experimentation platform”, “Task 4.3 NFV automatic testing & validation framework through continuous integration (V&V CYCLE)” and finally “Task 4.4 Technical Support and assistance to 3rd parties for NetApp Deployment”.

1.2. Document Structure

The deliverable is organized in the following manner:

Section 2 provides the generic overview of the architectural components and interfaces of the Smart5Grid platform with a focus on the development and deployment. This analysis offers the general idea of the different modules of the platform to assist the deployment of the Network Apps. In the same section, the lifecycle of the whole application is provided with specific general steps that a developer can easily understand. With this aim the section 3 is devoted to analyse the guide on how to develop a Network Application with detailed steps. Section 4, Section 5, Section 6 and Section 7 provide the information on the specific Network Applications in the different UCs of the project and how the followed and implemented the guide of Section 3.

The deliverable is concluded by the final section with details on the final remarks of the document.

2. Smart5Grid Open 5G Platform

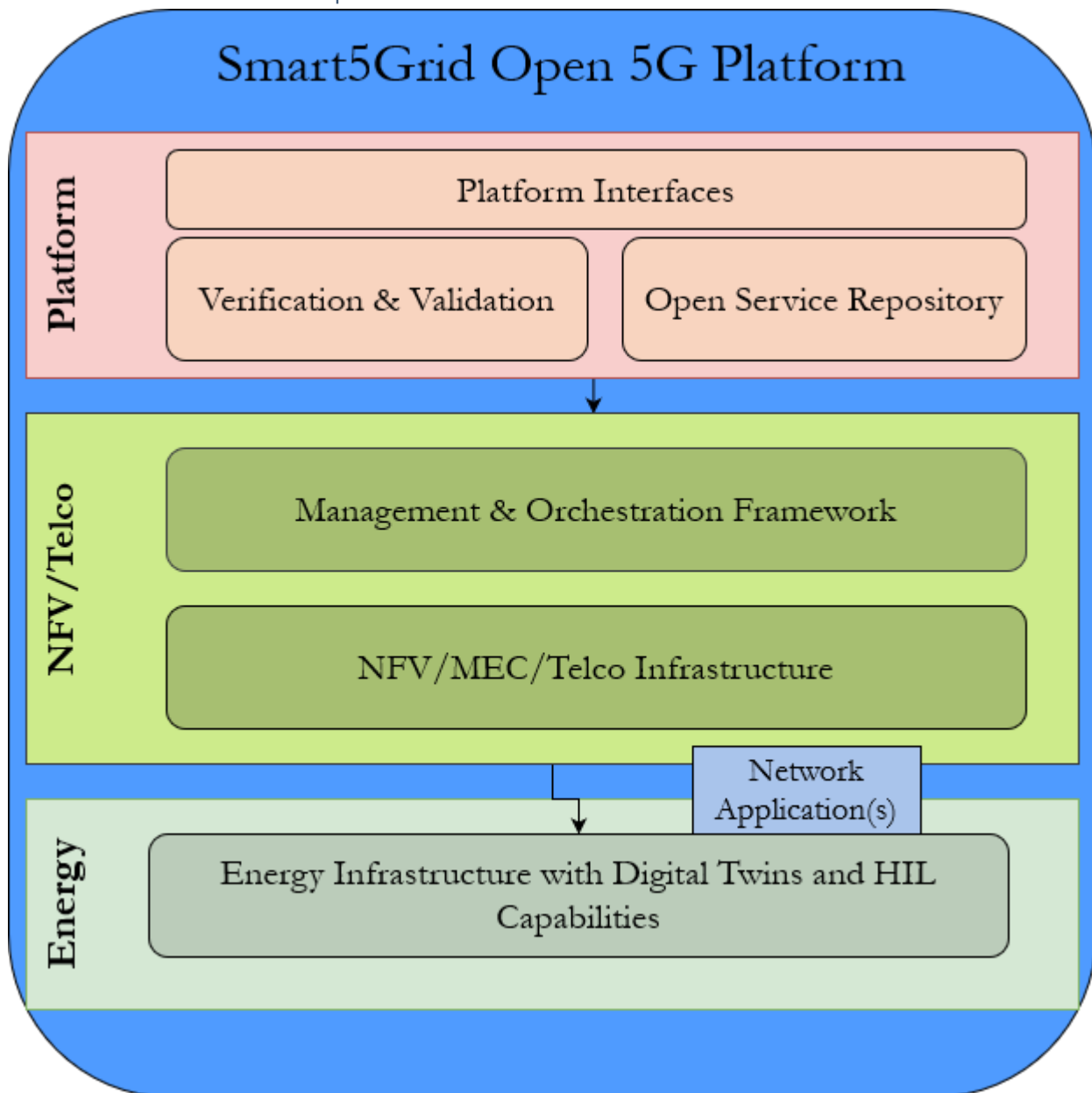


Figure 1. Smart5Grid Open 5G platform High-level Architecture

The Smart5Grid Open 5G Platform is an open, fully softwarised 5G experimental facility which enables developers to upload industry-specific Network Applications that are hosted on the Open Service Repository (OSR) for third parties to find and use. The primary purpose of the OSR is to make it possible for Network App developers to upload (store) and edit their applications and Virtual Network Function(s) (VNF). Additionally, it establishes a connection between the Network App developer and the V&V framework, enabling the verification and validation of Network Application to guarantee their compliance with the Smart5Grid. Through the integration of the application development and operation, this seeks to support DevOps procedures. These applications undergo comprehensive testing by the Validation & Verification (V&V) methodology prior to being made available in the repository, thus establishing a continuous integration and development loop cycle. As a result, a functional Network Application

prototype can be provided fast, and the developer can continue to enhance it based on its earlier tested iterations that are known to function within Smart5Grid's architecture. Developers can solely validate their application through the V&V module or can request a full validation and verification prior to the onboarding process. The certification and deployment of the Network Applications both heavily rely on the Management & Orchestration (M&O) framework. The M&O framework's job is to oversee every facet of Network Application execution throughout its lifecycle across the virtualization and communication infrastructure. This control is carried out by various elements that control a certain infrastructure area (i.e., Virtualization and Edge Computing, Core Network, Radio Access Network). Slices are created from the resources used on each domain to provide deployment segregation. In order to assess the effectiveness of the validation tests carried out and to keep track of the applications' performance, metrics from each execution are gathered from each domain. Figure 1 represents all the high-level modules of the Smart5Grid Open 5G Platform that were previously described.

2.1. Smart5Grid network application vision/definition

Smart5Grid project conceives a Network Application as a "tool" to bridge the gap between 5G networks, software solution providers and the enterprise customers from the several economic verticals. In the context of Smart5Grid and WP4, our purpose is to capacitate developers to build innovative vertical applications making use of all the advantages that 5G networks have to offer but without the need of being experts on 5G technology. According to Smart5Grid, Network Applications must abstract the network complexities into a set of requirements, letting developers concentrate on building the applications specific to the vertical domain they master.

The definition of the Smart5Grid Network Application is based in three key technology trends:

- the Cloud Native paradigm,
- the ETSI NFV framework,
- the edge computing concept

Based on this, Smart5Grid project defines a Network Application as a chain of services and subservices. These services may be modelled in two different ways:

- a) Purely Cloud-Native (CN) application wrapped as a Network Application.
- b) Telco CN application delivered as an OSM-compatible Network Service (NS) wrapped as a Network Application.

The underlying cloud native technology in both cases is the same and thus it is possible to leverage the same vertical application in either approach. The granularity in the models reaches all layers, meaning that the linkage between each layer is easily interchangeable and updated, increasing their portability and, on the other hand, making possible to leverage the potential of 5G cloud and edge deployments, as each of these subservices may have associated their own service level objectives (SLO). In fact, according to Smart5Grid project, a Network Application differs from a plain vertical application in the sense that it contains the necessary access performance requirements that are needed to fulfil the demanding performance constraints that applications in the vertical domain may request. Modularity also allows software reuse, which encourages collaboration, fosters innovation, and widens impact.

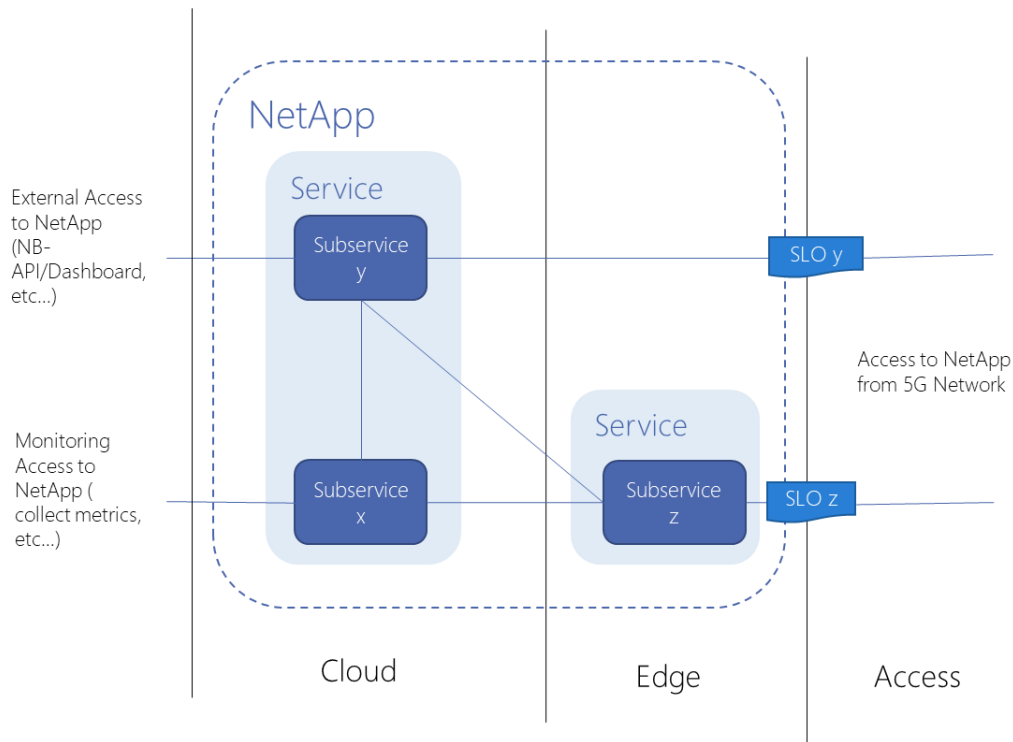


Figure 2. Basic Network App representation

A Network Application is formally defined by the so-called Network Application Descriptor. To facilitate developers the creation of this descriptor, an Information Model (IM) has been defined using YANG as the data modelling language. This IM defines the structure of the Network App and lists each of the fields that compose it, as well as the type of data that each field must follow. The first version of the Smart5Grid Network Application IM was introduced in D2.2 [1] and then improved and updated in D3.3 [2], so we encourage the reader to check this deliverable for further information.

2.2. Smart5Grid Network Application lifecycle in Smart5Grid Platform

This section describes the lifecycle management of Smart5Grid network applications from the point of view of the application developers or providers. The description of the lifecycle of network applications from the point of view of the components involved in the management and orchestration of virtualized grid services has been described in document D3.1 [3] in section 3.2.

The main objective of this section is to provide the basic steps that developers should take in consideration during the deployment of their services in a network infrastructure. While it is true that the software components of the M&O layer will be mentioned in order to describe the stages of network application lifecycle management, the aspects performed by each of them will not be discussed in depth.

Figure 3 shows the main stages of the application lifecycle from the developers' point of view. It is basically divided into 6 steps: Development, Packaging, Validation, Deployment, Operation and Termination. Each of the stages are described in the following subsections.

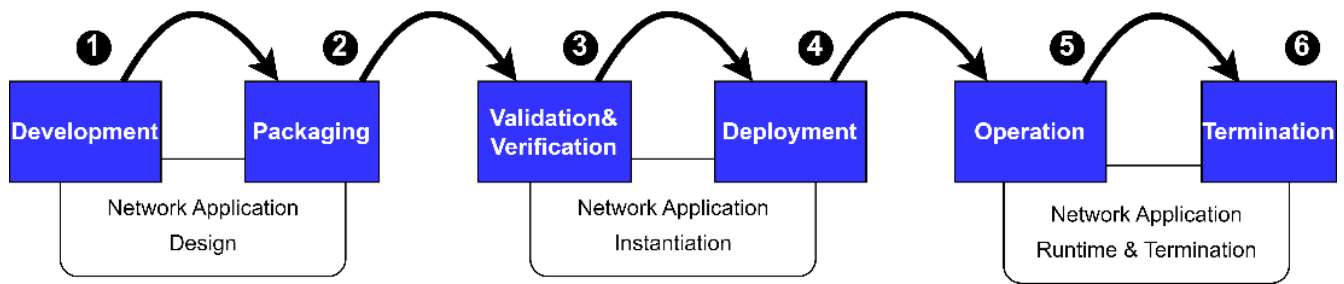


Figure 3. Network Application lifecycle developer's point of view

2.2.1. Network Application Design

Network application design is the very first step involved during the lifecycle management. It involves several steps that developers must execute to generate a package that can be executed on the Smart5Grid platform. Basically, it is divided into two essential steps which are the development of the application itself and the packaging of the application with artifacts that are supported by the Smart5Grid platform. The details of each of these steps have been described in section 2.2.

2.2.2. Network Application Validation and Verification

This stage allows the developers to have a first feedback on the definition of the application in terms of descriptors. The Validation and Verification components check that the content of the descriptors is correct and that it meets the specifications of the implemented information model. Also, the V&V tools can check the various descriptors (e.g., VNFDs and NSD) embedded in the network application packages.

This step is executed when the developers or end user executes the command in the OSR component to validate a network application on the network infrastructure. The OSR component sends the request to the V&V component, and this is where it verifies the integrity of the application artifacts. If the descriptors are correctly defined, the next step will be executed which is the deployment of the application, otherwise it will issue a warning of the problems found in the descriptors and its instantiation will not be executed until they are solved.

2.2.3. Network Application Deployment

If the validation and verification procedure of the network applications was successful, the application instantiation stage is executed. Several components are involved here to manage and configure the network infrastructure where the application will provide services. These steps were described in document D3.1 [3] in section 3.2.2.

The important aspect at this stage from the developer's point of view is that the deployment and instantiation procedure is completely transparent. All the management and orchestration components of the Smart5Grid platform, such as Network App Controller, Slice Manager and Network Functions Virtualization Orchestrations (NFVOs), abstract the complexity of configuring the network and computational aspects while displaying feedback of the status of the instantiation process, i.e., whether it was successful or unsuccessful.

2.2.4. Network Application Operation

This stage is fundamental to validate and evaluate the operation and performance of a network application. The first step during application operation is to validate that the service provided by the application is executed as expected. For this purpose, the following steps must be performed:

1. Enable end users or user equipment to establish connectivity to network applications.
2. Depending on the functionalities of the application, it is necessary to verify whether both the UEs/devices and the applications are generating the expected data traffic. In this step, applications may be required to incorporate portals or dashboards where developers can verify end-to-end traffic flow and performance.
3. At the same time, the Smart5Grid platform monitoring system will take into account the thresholds defined in the network application descriptors and enforce the Service Level Objectives through the Network App Controller to execute scaling and healing actions.

2.2.5. Network Application Termination

The process of decommissioning network applications is also transparent to the developer or end user. The users simply execute the action on the Network App Controller and this is responsible for executing the process of removing the network applications deployed in the network infrastructure by interacting with the Slice Manager and this in turn with the NFVOs. Once the application is removed. The end user will receive an ACK that the process was successful.

3. How to develop a Network Application. A Guide for developers

This section explains in detail how to develop a Network Application in the context of the Smart5Grid project. Figure 4 depicts the different technologies involved in the development of the required artefacts (by-product resulting from the release of a software in a specific technology) for the two variants that Smart5Grid considers, as documented in the previous section.

The first two artefacts from Figure 4 are common but then, each implementation varies as required by the underlying technologies. Some basic knowledge about each of the technologies (Docker, Helm, OSM) is assumed for the reader, although there are references to external documentation whenever mentioned to support the description.

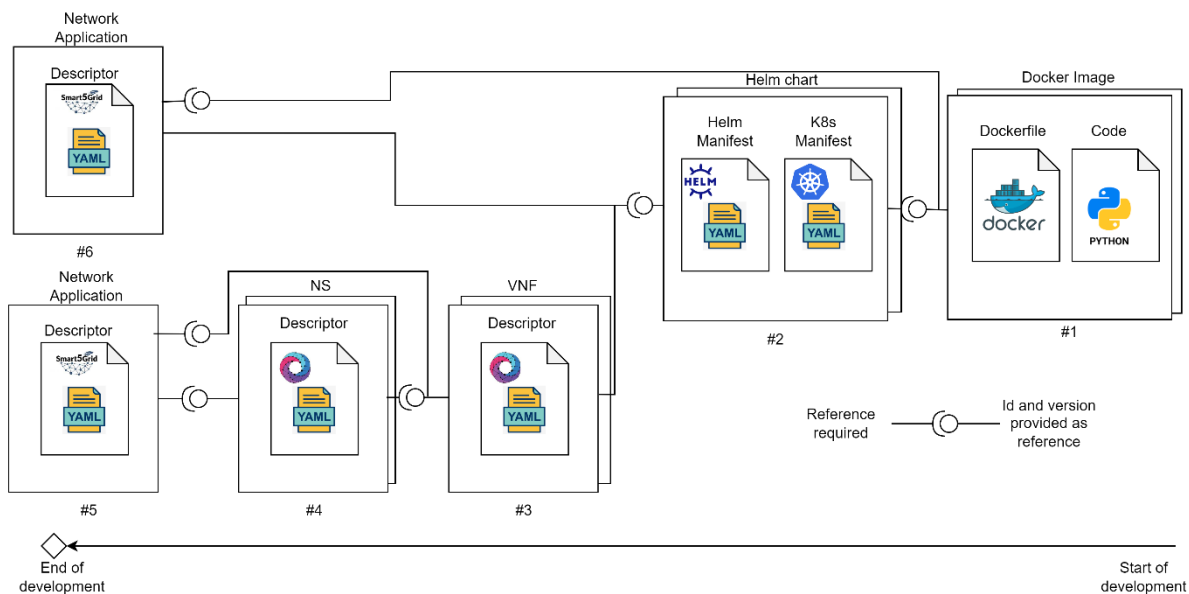


Figure 4. Network Application technologies and artefacts

Overall, the steps to be followed to develop a Network Application are the following:

1. Develop Network Application logic in your programming language of choice.
2. Build Docker image.
3. Create a Helm chart and reference the Docker image.
4. For Telco Network Applications:
 - 4.1. Create the OSM-compatible VNFs descriptor(s) and reference the Helm chart.
 - 4.2. Create an OSM-compatible Network Service(s) descriptor(s) and reference the VNF(s).
 - 4.3. Create the Network Application descriptor and reference the NS(s) as service(s) and the VNF(s) as subservice(s).
5. Pure Cloud-native Network Applications:
 - 5.1. Create the Network Application descriptor and reference the Helm chart(s) as service(s) and the Docker image(s) as subservice(s).

Each of those steps result in the creation of an artefact (by-product of a software development) that must be packaged and pushed to the Open Service Repository using the corresponding Command Line

Interface (CLI) and the OSR API. The following sections describe the process for one, the simple case of a single application (Service):

1. Develop Network Application logic in your programming language of choice.

The design and development of network applications is the first step performed by developers. It is often performed under special aspects and environments with special features. However, it is an essential step to experiment with the functionalities of the network application in the developers' environment. This will allow profiling and characterizing the application in terms of virtualized network functions either based on virtual machines or containers.

At this stage, developers determine the essential resources for the network application to run as expected. It also allows defining the essential aspects that will be required in the context of the network infrastructure and end users, e.g. communication protocols and network capabilities. The main steps in designing a network application are as follows:

1. Definition of the functionality and specification of the actors that would be involved during the provision of the network application services, e.g., end users, electronic devices, etc. Here the client-server model and the communication characteristics such as ports and traffic type are defined.
2. Development of the network application with programming language selected by the developers. It means, the Network Application logic is developed in the chosen programming code (Python, Java, etc.).
3. Implementation and performance evaluation of the network application in developers' environments. Here it is important to perform both network and computing performance measurements. On the network side, it is useful to validate traffic profiles that will allow sizing the network resources that the application will need during execution in production environments. Similarly for the computing part, this will allow to foresee the minimum range of computing resources to be allocated to the application in virtualization environments.
4. Once the application has been tested and the dependencies defined (i.e., "requirement.txt" file in Python), you are required to deliver it as a containerized application based on Docker images. There are multitude of examples and templates online, Docker itself provides a reference guide for this.

You can also push the application code at the gitlab defined for the desired VDU. In case you need to create a new VDU and push the code the steps needed are:

- Create a new VDU


```
curl --location 'https://osr.s5g.gos.y-cloud.eu/vdu/create/' \
--header 'Authorization: Bearer $ACCESS_TOKEN' \
--header 'Content-Type: application/json' \
--data '{"name": "$VDU_NAME", "description": "$VDU_DESCRIPTION", "public": false, "image_type": "docker"}'
```
- Find the "repository_url" for the new VDU. You can find it from <https://osr.s5g.gos.y-cloud.eu/vdu/list/>
- Go to the application folder (\$APP_FOLDER) and type the commands:


```
cd $APP_FOLDER
git init --initial-branch=main
git config user.name $USER_NAME
```

```
git config user.email $USER_EMAIL
git remote add origin ssh://git@repo.s5g.gos.y-cloud.eu:3022/$VDU_NAME
git add .
git commit -m "Initial commit"
git push -u origin main
```

To validate and test the performance of the network application in the Smart5Grid platform (outside the developers' lab), the virtualization technology to be used to instantiate the application must be decided. Here, minimum latencies of service operation must be considered in order to determine the optimal placement of the application, as well as the expected computing resources. It means, if the application is time sensitive, the best location will be in an edge computing environment, otherwise it can be located in any cloud computing. On the other hand, if it is sensitive to delays, and also demands special resources, the available resources of the infrastructure providers must be taken into consideration.

In general, if the application is executed in an Edge Computing environment (in any of its variations, e.g., Far and Extreme Edge), container-based virtualization will be used. For this purpose, the following steps detailed below are followed.

2. Build Docker image.

Assuming you have a container engine installed locally, such as Docker Desktop, the process for building the image is just a single command. In case of not having the engine installed, refer to the following installation guides depending on your operating system:

- For Linux, check out the documentation [here](#).
- For Windows, review this documentation [here](#).

It may be advisable to review the Windows Subsystem for Linux (WSL) documentation as well.

To upload a Docker type VDU to OSR follow the instructions below:

Make sure you are logged in to the docker registry

```
`docker login https://$OSR_REGISTRY_FQDN/$VDU_NAME`
```

Build and tag the local docker image

```
`docker build -t $OSR_REGISTRY_FQDN/$APP_ID:$APP_VERSION -f $PATH_TO_DOCKERFILE $PATH_TO_SOURCECODE`
```

More options are allowed for the docker build command, such as passing environment variables, etc/host entries. Refer to the official documentation [here](#) [4].

The docker build command expects the developer to provide fields that are particular to the development environment. The list below explains each of the parameters that need to be added by the developer.

- **OSR_REGISTRY_URL:** The Smart5Grid Docker registry where the Docker image will be stored until it is required by the Network Application. By creating a VDU in the OSR a new Docker registry is created and you can access it at the URL in the field "registry_url" of the vdu create response. Example creation of VDU for docker is defined in previous chapter.

- `OSR_REGISTRY_FQDN`: This is the domain name of the OSR_REGISTRY. This value is: `c-registry.s5g.gos.y-cloud.eu`
- `APP_ID`: This is the unique identifier that you will give to your application.
- `APP_VERSION`: This is the tag for the release of your application. If nothing is added, then “latest” will be automatically assigned.
- `PATH_TO_DOCKERFILE`: This is the local path from the directory where you are running the Docker command to the “Dockerfile” file that provides the instructions to build the Docker image. Docker reads these instructions, executes them, and creates a Docker image as a result. For reference, you may check the guide mentioned in step 1 or a more detailed one available [here \[4\]](#).
- `PATH_TO_SOURCECODE`: Source code and “Dockerfile” should sit in the same directory because the instructions in the “Dockerfile” can only refer to files and folders starting at the same level as where the file is. In order to change the origin of the visibility of the “Dockerfile”, you have to specify the full path here. In the recommended configuration, your files will look like in **Figure 5** and then `PATH_TO_SOURCECODE = PATH_TO_DOCKERFILE`.

```
$PATH_TO_SOURCECODE
|_ app.py
|_ requirements.txt
|_ Dockerfile
```

Figure 5. Source code directory example for python application

Once completed this step, you will have obtained artefact #1 from Figure 4.

Lastly, push the docker image to OSR:

```
`docker push $OSR_REGISTRY_FQDN/$VDU_NAME/$APP_ID:$APP_VERSION`
```

3. Create a Helm chart and reference the Docker image.

Docker images can be deployed locally for additional debugging and testing but, in the case of Smart5Grid, these will not be the deployment mechanism used. Instead, a more robust and feature-reach orchestration engine will be used: Kubernetes (K8s), as it exploits the scale, elasticity, resilience, and flexibility offered by the cloud.

In the event of not having access to a K8s cluster (group of nodes that run containerized applications in an efficient, automated, distributed, and scalable way) for testing, a couple of very common references are the following:

- [Create a K8s cluster with Kubeadm \[5\]](#)
- [Create a K8s cluster with Minikube \[6\]](#)

The Smart5Grid platform provides and maintains these nodes so you can deploy services the need to have your own K8s cluster. The Network Application Controller (NAC) will be in charge of deploying these services in the K8s cluster available.

A very basic K8s application would be composed of, at least, two main manifests (YAML file where K8s resources are defined): a deployment and a service. In this guide, K8s manifests will not be created individually, instead Helm will be used to create and package the application and also to orchestrate its deployment.

A K8s deployment is a resource that informs K8s how to create and update your application. A deployment allows you to describe an application's life cycle, such as which images to use for the app, the number of pods (minimal workload level in K8s which comprises one or more containers) that needs to be created, and the way in which they should be updated. An example may be obtained directly from the [K8s documentation](#) [7].

A deployment in K8s does not allow external components to interact with it, which is required by the Smart5Grid platform. This is achieved by defining a K8s Service. A Service in K8s is a resource that defines a logical set of Pods and their access policy. Services enable a loose coupling between dependent Pods, exposing their interfaces in different ways. An example may be obtained directly from the K8s documentation [here](#) [8].

As indicated earlier, instead of deploying each manifest individually in a manual approach, the Smart5Grid platforms relies on Helm to orchestrate the entire application. Helm is an open-source tool which manages K8s applications packaged in charts, which are a collection of files that describe a related set of K8s resources. Their syntax is almost identical to the K8s manifests themselves but offers a templating mechanism based on YAML and JinJa2. There are multiple resources online to learn how to use it, the official Helm documentation is available [here](#) [9].

Helm is managed by the NAC in Smart5Grid. However, in order to create the Helm chart and debug the application locally, you may refer to the [official installation guide](#) [10] and get familiarized with the most relevant commands and concepts in order to manage the application and connect to the K8s cluster properly using [additional resource online](#) [11].

To create the first chart, you may start from scratch using the references provided or just run the command `helm create \$APPLICATION_CHART` and then customize the example that is created to your application. More information is available [here](#) [12]. In either case, the minimal files that should be included are shown in Figure 6.

```
$APPLICATION_CHART
|__ templates
| |__ deployment.yaml
| |__ service.yaml
|__ chart.yaml
|__ values.yaml
```

Figure 6. Source code directory for Helm chart

where:

- Templates: This folder includes the template files annotated with JinJa2 code which will be rendered as K8s manifests at instantiation time using the content of the values.yaml file and additional contents provided at the Network Application descriptor level (step 4.3 or 5.1).
- Chart.yaml: Identifies the chart as an application with a unique and version and some additional metadata
- Values.yaml: Contains the default key:value pairs that will be taken to render the templates.

Helm charts are very different depending on the application to be deployed. However, Figure 7 shows some standardized fields that need to be carefully filled due to dependencies with the Smart5Grid

platform. The key:values pairs shown in Figure 7 could be added to either the deployment.yaml or the values.yaml files at the preference of the developer.

```
image:
  name: $OSR_REGISTRY_URL/$APP_ID:$APP_VERSION
  imagePullSecrets: $OSR_REGISTRY_URL_SECRET
```

Figure 7. Helm chart standardized fields

Configure access to docker image OSR registry:

The first field shown is used to reference the Docker image created in step 1 to the Helm chart. Additionally, OSR registries are private and thus credentials need to be provided to allow its use. This is most commonly achieved by referencing to previously created K8s secrets (using the "imagePullSecret" key). To access this image from a Kubernetes deployment you will need to create the following secret:

```
`kubectl create secret osr-registry-secret regcred --docker-
server=$OSR_REGISTRY_FQDN/$VDU_NAME --docker-username=$USER_EMAIL --docker-
password=$CONTAINER_REGISTRY_SECRET --docker-email=$USER_EMAIL`
```

- `$CONTAINER_REGISTRY_SECRET`: You can get it from OSR's endpoint `/users/details/` API GET request. See D3.3 [2] Appendix for details.

Next, it is time to create Helm chart and push to OSR. Helm charts will be stored in the OSR too. The process to do it starts by packaging the chart in an artefact with the command

. To upload the Helm type VDU follow the instructions below:

- Ensure the helm-push plugin is installed to your local workstation
``helm plugin install https://github.com/chartmuseum/helm-push``
- Ensure you have added the Harbor repository
``helm repo add -username==$USER_EMAIL $VDU_NAME
https://$OSR_REGISTRY_FQDN/chartrepo/$VDU_NAME``
- Create the artefact (.tgz file) from chart directory
``helm package $PATH_TO_APPLICATION_CHART`. This will create a new
file called "APPLICATION_CHART-CHART_VERSION.tgz"`
- Push the Helm chart to the container image registry
``helm cm-push $APPLICATION_CHART-CHART_VERSION.tgz $VDU_NAME``

Once completed, you will have finished the creation of artefact #2 from Figure 4.

4. For Telco Network Applications

Telco infrastructure relies on well-established standards and specification. Due to this, Smart5Grid has implemented a variation of the Network Application that allows for the usage of Open-Source MANO (OSM) descriptors which showcases an ETSI MANO stack.

OSM is an Open-Source implementation of a commercial-grade ETSI MANO stack that is well documented [online](#) [13]. For the interest of the Network Applications developers, they need to create NSs, which are also built as the combination of one or more VNFs.

4.1. Create the OSM-compatible Virtual Network Function(s) (VNF) descriptor(s) and reference the Helm chart.

OSM VNFs are structured following the [SOL004 specification \[14\]](#), although it supports a simplified version which is the one that will be used in this guide.

The [Information Model \[15\]](#) (IM) of the VNF descriptor uses the [specification SOL001 \[16\]](#) with some additional functionalities offered by OSM itself explained in the reference provided. Figure 8 depicts an example of a OSM VNF descriptor (left) and the associated package structure (right) that needs to be created, where the content of the file “descriptor.yaml” corresponds to the example provided in the left side of the Figure.

Note that, in this case, the reference to the Helm chart does not point to the OSR, as it would be expected by the previous step in the guide. This is due to a limitation in OSM. Smart5Grid has created a fix as part of the NAC which needs to be taken into account when developing the VNF descriptor. OSM has been configured to pull the Helm chart from a component called localregistry at the Network Application Controller component from the Smart5Grid Platform, which is automatically synchronized with the OSR at instantiation time although other public and accessible Helm chart repositories could be used.

```
vnfd:
  description: VNF using a helm-chart stored in the localregistry
  df:
    - id: default-df
  ext-cpd:
    - id: mgmt-ext
      k8s-cluster-net: mgmtnet
  id: $APPLICATION_VNF_ID
  k8s-cluster:
    nets:
      - id: mgmtnet
  kdu:
    - name: demo_netapp_chart
      helm-chart: localregistry/$APPLICATION_CHART:$CHART_VERSION
  mgmt-cp: mgmt-ext
  product-name: Telco NetApp
  provider: ATOS
  version: '$APPLICATION_VNF_VERSION'
```

\$APPLICATION_VNF
|_descriptor.yaml

Figure 8. Telco Network App's VNF descriptor example (left) and VNF package structure (right)

To upload the VNF to the OSR you must create a new VNF:

```
curl --location 'https://osr.s5g.gos.y-cloud.eu/vnf/create/' \
--header 'Authorization: Bearer $ACCESS_TOKEN' \
--header 'Content-Type: application/json' \
--data '{"vnf_id": $APPLICATION_VNF_ID, "public": false, "descriptor":
$VNF_DESCRIPTOR}'
```

where \$VNF_DESCRIPTOR is the descriptor.yaml content. See D3.3 [2] Appendix for details.

4.2. Create an OSM-compatible Network Service(s) descriptor(s) and reference the VNF(s).

OSM does not instantiate bare VNFs, instead, one or more VNFs are connected as part of a NS. The package structure for this artefact follows the SOL004 specification again and, just like in the case of VNFs, it can be simplified to a single descriptor file. For the contents of the NS descriptor, the [IM \[17\]](#) is defined in the [SOL007 specification \[16\]](#).

```
nsd:
  nsd:
    - description: NS for a Telco NetApp
      designer: ATOS
      df:
        - id: default-df
          vnf-profile:
            - id: vnfprofile1
              virtual-link-connectivity:
                - constituent-cpd-id:
                    - constituent-base-element-id: demo_netapp_chart
                      constituent-cpd-id: mgmt-ext
                virtual-link-profile-id: mgmtnet
              vnfd-id: $APPLICATION_VNF_ID
            id: $APPLICATION_NS_ID
            name: Telco Netapp NS
            version: $APPLICATION_NS_VERSION
            virtual-link-desc:
              - id: mgmtnet
                mgmt-network: true
            vnfd-id:
              - $APPLICATION_VNF_ID
```

Figure 9. Telco Network App's NS example (left) and package structure (right)

To upload the NS to the OSR you must create a new NS:

```
curl --location 'https://osr.s5g.gos.y-cloud.eu/ns/create/' \
--header 'Authorization: Bearer $ACCESS_TOKEN' \
--header 'Content-Type: application/json' \
--data '{"ns_id": $APPLICATION_NS_ID, "public": false, "descriptor":
$NS_DESCRIPTOR}'
```

where \$NS_DESCRIPTOR is the descriptor.yaml content. See D3.3 [2] Appendix for details.

4.3. Create the Network Application descriptor and reference the NS(s) as service(s) and the VNF(s) as subservice(s).

In Smart5Grid, OSM is not used directly, instead a NAC is used to orchestrate one or more NSs as part of a single Network Application. The IM of the Network Applications developed by Smart5Grid has already been described in detail in D3.3 [2].

Figure 10 shows an example of a Telco Network Application that references the artefacts described in previous sections. Notice that there are fields that are not added (internal-links) because it is not required for the specific implementation of a single-service Network Application.

Generate the Network App artefact

```
`tar -czvf $TELCO_NETAPP_NAME-$TELCO_NETAPP_VERSION.tar.gz
$PATH_TO_TELCO_NETAPP_DESCRIPTOR`.
```

This corresponds with artefact #5 from Figure 4.

Create Network app to OSR using the `"/netapp/create"` API POST request.

The `"tar.gz"` file can be uploaded with the `"/netapp/upload-descriptor/:id"` API POST request.

See D3.3 [2] Appendix for details.

```

netapp:
  im-version: 0.1.0
  name: $TELCO_NETAPP_NAME
  description: Telco NetApp example
  provider: ATOS
  version: $TELCO_NETAPP_VERSION
  service-format: osm
  services:
    - name: $APPLICATION_NS_ID
      package: $APPLICATION_NS_ID-$APPLICATION_NS_VERSION.tar.gz
      subservices:
        - name: $APPLICATION_VNF_ID
          package: $APPLICATION_VNF_ID-$APPLICATION_VNF_VERSION.tar.gz
      values: |
        foo: bar
      sap:
        - name: mgmtnet

monitoring-endpoint:
  service-ref: $APPLICATION_NS_ID
  sap-ref: mgmtnet
  url: $MONITORING_URL

external-endpoints:
  - name: external-endpoint1
    service-ref: $APPLICATION_NS_ID
    sap-ref: mgmtnet
    security-group-rules:
      - id-ref: http

access-endpoints:
  - name: access-endpoint1
    service-ref: $APPLICATION_NS_ID
    sap-ref: mgmtnet
    security-group-rules:
      - id-ref: ssh
    policies:
      - key: latency
        value: '6'

SLOs:
  - name: number of connected PMUs
    expression: rate(PMUs_number[5m])
    metric: PMUs_number
    threshold: '10'
    threshold-type: GT
    action:
      target-ref:
        target-service-ref: $APPLICATION_NS_ID
        target-subservice-ref: $APPLICATION_VNF_ID
      action-step: trigger-scale-up
    granularity: '3'
    cycles: '4'

security-group-rules:
  - id: http
    description: http rule
    direction: ingress
    ether-type: ipv4
    protocol: tcp
    port-range-min: 80
    port-range-max: 80

```

Figure 10. Telco Network Application example

5. Pure Cloud-native Network Applications:

Complementary to the implementation explained in step 4, Smart5Grid has developed a non-Telco approach which allows for vertical service providers to leverage the features of the Smart5Grid platform outside of the requirement of the Telco specifications. Instead, Network Applications are linked directly to Helm charts from step 3 and orchestrated with a different type of NAC.

5.1. Create the Network Application descriptor and reference the Helm chart(s) as service(s) and the Docker image(s) as subservice(s).

Error! Reference source not found. Figure 11 depicts a Cloud-Native Network Application that leverages the artefacts from steps 1-3. Notice that there are several keys that are not used due to limitations of this kind of implementations:

- access-endpoints: in this environment, there is no integration with User Plane Functions or Slice Managers
- security-group-rules and internal-links: these are delegated to the Helm chart.

```
netapp:
  im-version: 0.1.0
  name: $CN_NETAPP_ID
  description: Cloud-Native NetApp
  provider: ATOS
  version: $CN_NETAPP_VERSION
  service-format: helm
  services:
    - name: $APPLICATION_CHART_NAME
      package: $OSR_CHART_REPOSITORY/$APPLICATION_CHART_NAME:$APPLICATION_CHART_VERSION
      subservices:
        - name: $APP_ID
          package: $OSR_REGISTRY_URL/$APP_ID:$APP_VERSION
      values: |
        foo: bar
      sap:
        - name: na1_sap

  monitoring-endpoint:
    service-ref: $APPLICATION_CHART_NAME
    sap-ref: na1_sap
    url: $MONITORING_URL

  external-endpoints:
    - name: external-endpoint1
      service-ref: $APPLICATION_CHART_NAME
      sap-ref: mgmtnet

  SLOs:
    - name: number of connected PMUs
      expression: rate(PMUs_number[5m])
      metric: PMUs_number
      threshold: '10'
      threshold-type: GT
      action:
        target-ref:
          target-service-ref: $APPLICATION_CHART_NAME
          target-subservice-ref: $APP_ID
        action-step: trigger-scale-up
      granularity: '3'
      cycles: '4'
```

Figure 11. Cloud-Native Network Application example

Generate the NetworkApp artefact with the command:

```
`tar -czvf $CN_NETAPP_NAME-$CN_NETAPP_VERSION.tar.gz $PATH_TO_CN_NETAPP`.
```

This corresponds with artefact #6 from Figure 4.

Push the Network App artefact to OSR:

The "tar.gz" file can be uploaded with the `"/netapp/upload-descriptor/:id"` API POST request. See D3.3 [2] Appendix for details.

4. UC#1 Network Application(s)

In the context of this UC, a Network Application has been developed to support the DSO (EDI), or the contracted provider that manages its communication network, with the purpose of monitoring in real-time the connectivity performance. This aspect is crucial because without an efficient communication layer, the real-time self-healing will not operate properly and might cause a huge impact in terms of affected users in case of a grid fault. Until now, DSOs can only see the connectivity provided by the TELCO as a black box, where any performance measurement can be executed only end-to-end. In this scenario, it is very difficult to properly address cases of misconnection or degradation of service level. Such implementation introduces a novelty for both DSO and TELCO operators, that is: performing a test from inside the TELCO network, right before the RAN network, which allows the identification whether the fault is in the RAN network and/or the field devices or some other part of the entire communication chain.

4.1. Technical specification

The UC1 (Use Case) showed in Figure 12 involves the implementation of a telecommunications network monitoring mechanism. This mechanism oversees the status of the local network, with the aim of improving remote-control connectivity between entities that are involved in the power distribution ecosystem. While EDI provides an advanced real-time self-healing mechanism, not all primary and secondary substations are equipped with communication infrastructure required to activate the mechanism. Consequently, in the event of communication problems, technicians are required to invest a considerable amount of effort and time in the troubleshooting process. Since there are no tools available to assist with this process, a series of inspections for connectivity checks on the field devices will be necessary. The use of 5G infrastructure provides excellent flexibility in grid management, offering the necessary combination of availability, reliability, and network latency to the Network Application.

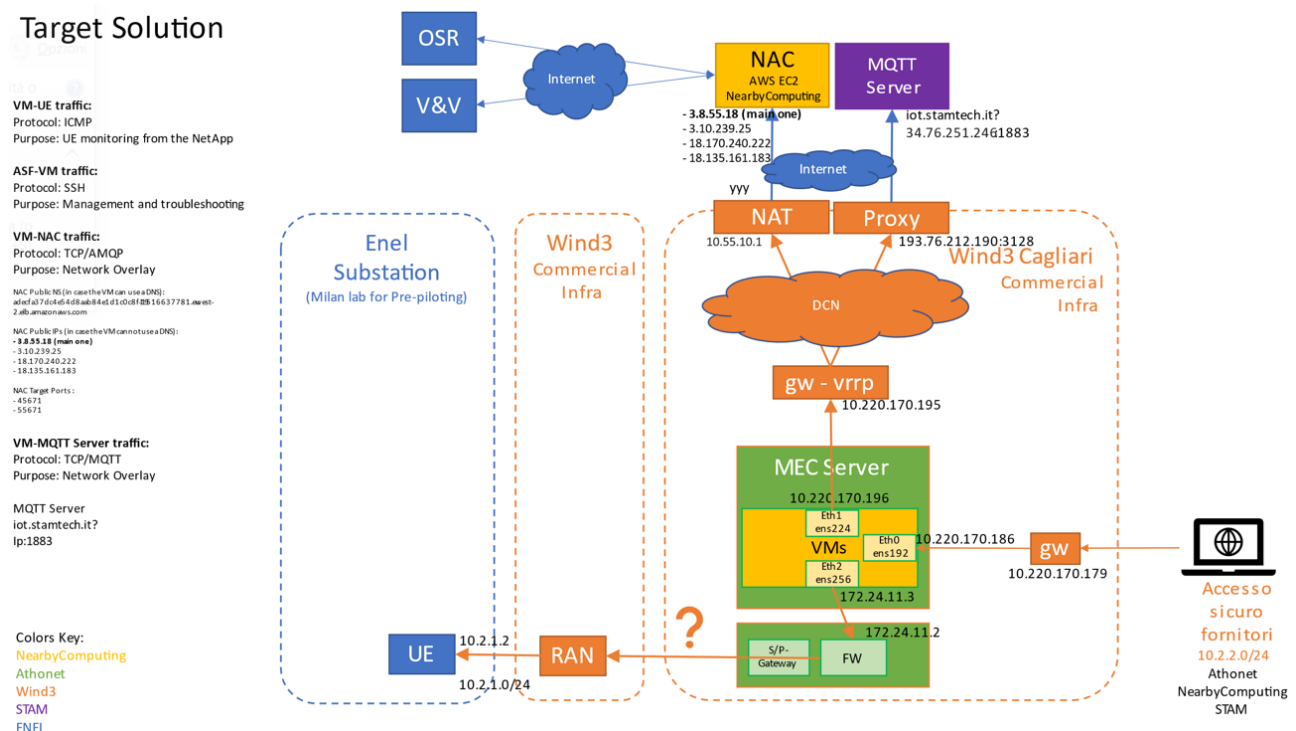


Figure 12 UC1 Network Application - High level architecture solution

4.2. Functional and Technical Requirements

The evaluation of network performance is critical to ensuring optimal system functionality. In this context, the following functional and technical requirements were measured and analysed for a network system: latency, jitter, slice deployment and adaptation time and data rate. For latency, the goal was to achieve an average latency of below 25ms. For jitter, a low standard deviation was required to indicate a stable connection. For slice deployment and adaptation time, the objective was to have an efficient deployment process that takes the minimum required time. Finally, for data rate, high rates were desired for both uplink and downlink, with accurate measurements for both TCP and UDP protocols. The following sections present the results obtained for each of these requirements.

FPVM-01 (Latency):

- **Functionality:** Measure latency of the system.
- **Technical requirement:** Achieve an average latency of below 25 ms.

FPVM-02 (Jitter):

- **Functionality:** Measure stability of the connection.
- **Technical requirement:** Have a low standard deviation to indicate a stable connection.

FPVM-03 (Packet Loss)

- **Functionality:** Counting packet loss involves monitoring failed data packets in a network to identify connectivity issues.
- **Technical requirement:** Provide real-time feedback and alerts in case of significant packet loss.

FPVM-04 (Availability)

- **Functionality:** Measure the percentage of time that a system or service is available and functioning as expected
- **Technical requirement:** Have percentage over alert threshold of availability of the system.

FPVM-05 (Reliability)

- **Functionality:** Measure the ability of the system to remain operational and accessible to users
- **Technical requirement:** the system or service needs to maintain an uptime of at least 99.9%

4.3. Network Application Functionality

For the successful execution of this use case, the e-distribuzione Control Center manages Medium Voltage/Low Voltage substations that are located in Olbia, which are responsible for operating the electricity grid for the Sardinia region. This control center is connected to a public 5G network, along with a Primary Substation and 5 Secondary Substations. To test the Automatic Power Distribution Grid Fault Detection technology provided by 5G Network Application, e-distribuzione s.p.a. and its partners, including

Wind Tre and other WP5 partners, will upgrade the necessary components of the existing infrastructure to successfully pilot and demonstrate the Network App under real-life conditions.

The developed service is a powerful tool for analysing the quality of traffic within a 5G network. It accomplishes this task by utilizing a Python script that is designed to detect and quantify various metrics, including P90 and P99 latency values, jitter, and the total number of packets transmitted. These metrics are then published via MQTT, which allows for easy integration with other systems and tools.

One of the key benefits of this service is its ability to generate alarms in response to certain thresholds being exceeded. For example, if the latency values for a particular IP address exceed a pre-defined limit, the service will automatically generate an alarm. This feature is critical for ensuring that network administrators are alerted to potential issues before they can escalate into larger problems.

4.4. Integration & Deployment

1. Develop Network Application logic in your programming language of choice.

To use the service, users must provide a few key parameters via an input file. These parameters include the IP address to be analysed, a token for publishing data to a dashboard, and the host where the data should be published. This approach makes it easy to customize the service for a variety of different use cases and scenarios.

The service can be hosted on an MEC (Multi-Access Edge Computing) server using one of two deployment methods. The first method involves secure access providers, which ensure that the service can be accessed securely and reliably. The second method involves using a Helm chart, which simplifies the deployment process and makes it easy to manage and scale the service as needed.

By default, the service performs analyses every 5 minutes on all IP addresses specified in the input file. This ensures that network administrators have access to real-time data and can take action as needed to maintain the health and performance of the network.

Finally, the service is packaged as a Docker container, which makes it easy to deploy and manage in a variety of different environments. A docker-compose.yaml file is included with the service, which can be run using the "docker-compose up" command. This ensures that the service can be quickly and easily deployed without any complex configuration or setup required.

2. Build Docker image.

The microservice will be available in a shared repository and the structure will be like in the Figure 13.

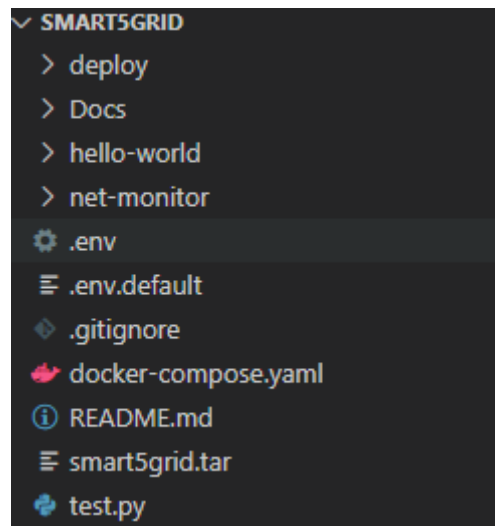


Figure 13. UC1 Network Application - Folder tree

An env file, short for "environment file," typically contains configuration information that an application needs to run. In this case, the env file includes the host information for a broker, an access token for publishing data, and an IP address to ping for extracting information. The broker host information likely relates to a messaging system that the application uses for communication, while the access token allows the application to authenticate with the system to publish data. Finally, the IP address to ping is likely used to retrieve information from a remote server, potentially related to monitoring the system or its performance.

```

BROKER_HOST=34.76.251.246
GOLFOARAN_1_A_ACCESS_TOKEN=7D4yDADyKlKJk1gmu9Cw
GOLFOARAN_1_IP_TARGET=10.2.1.1
GOLFOARAN_2_A_ACCESS_TOKEN=jnbS3eknTWkYZrI7E8vu
GOLFOARAN_2_IP_TARGET=10.2.1.2
GOLFOARAN_3_A_ACCESS_TOKEN=u5VWHPcxP6NcF6as7RZB
GOLFOARAN_3_A_IP_TARGET=10.2.1.3
EUCALIPTU_1_A_ACCESS_TOKEN=VnzRr882fDIM1sBLM03G
EUCALIPTU_1_A_IP_TARGET=10.2.1.4
EUCALIPTU_2_A_ACCESS_TOKEN=lw9vXEDMfxyOSrL9tz4q
EUCALIPTU_2_A_IP_TARGET=10.2.1.5
EUCALIPTU_3_A_ACCESS_TOKEN=z52II78iaeFAVePI9qLp
EUCALIPTU_3_A_IP_TARGET=10.2.1.6
PICIAREDDA_1_A_ACCESS_TOKEN=7LFcFalpmwMzPkMiW7bV
PICIAREDDA_1_A_IP_TARGET=10.2.1.7
PICIAREDDA_2_A_ACCESS_TOKEN=0HAWvpXlykn2WhyNNiJu
PICIAREDDA_2_A_IP_TARGET=10.2.1.8
PICIAREDDA_3_A_ACCESS_TOKEN=tlWuhtoXDPKvd1WlqggT
PICIAREDDA_3_A_IP_TARGET=10.2.1.9
PICIAREDDA_CONF_ACCESS_TOKEN=w90viBLp2aBAihp0vn9o
PICIAREDDA_CONF_IP_TARGET=10.2.1.10
PUTZOLU_1_A_ACCESS_TOKEN=v5EpaN350745hvBRsk6p
PUTZOLU_1_A_IP_TARGET=10.2.1.11
PUTZOLU_2_A_ACCESS_TOKEN=xPcGAUKkhHjrDNmavfr7

```

```

PUTZOLU_2_A_IP_TARGET=10.2.1.12
PUTZOLU_3_A_ACCESS_TOKEN=fFL0z1cJfJwCCsn9GcNo
PUTZOLU_3_A_IP_TARGET=10.2.1.13
PUTZOLU_CONF_ACCESS_TOKEN=ZQj1PsD0DyQkcSpXcmz8
PUTZOLU_CONF_IP_TARGET=10.2.1.14
TESTLAB_MILANO_1_ACCESS_TOKEN=Nvh7sk0ESkari6pH7ywk
TESTLAB_MILANO_1_IP_TARGET=10.2.1.26
TESTLAB_ROMA_1_ACCESS_TOKEN=NNswkyILgdgmDGk5BG8E
TESTLAB_ROMA_1_IP_TARGET=10.2.1.27
TESTLAB_MILANO_2_ACCESS_TOKEN=6u2cormgEM00qTzHrAcm
TESTLAB_MILANO_2_IP_TARGET=10.2.1.28
TESTLAB_MILANO_3_ACCESS_TOKEN=czhBBTFE4KN1IhwdHJdU
TESTLAB_MILANO_3_IP_TARGET=10.2.1.29
TESTLAB_MILANO_4_ACCESS_TOKEN=WneyBWa2tqd6f4gBDNoB
TESTLAB_MILANO_4_IP_TARGET=10.2.1.30

```

This is a YAML file that contains the configuration for a Docker Compose application. It specifies a version 3 of the Docker Compose syntax. The file defines a single service called GOLFOARAN_1_A, which is based on a Docker image from the registry.gitlab.com registry.

The environment section of the configuration sets environment variables that are used by the GOLFOARAN_1_A service. Specifically, it sets the ACCESS_TOKEN, BROKER_HOST, IP_TARGET, and SLEEP_TIME variables to values that are defined elsewhere in the environment, likely through other means such as environment variables or a separate configuration file.

Overall, this configuration file is used to deploy a Docker container based on the specified image, with the necessary environment variables set to enable the container to function properly.

```

version: '3'

services:
  GOLFOARAN_1_A:
    image: registry.gitlab.com/stamtech/industrial-automation-solutions/smart5grid
    environment:
      ACCESS_TOKEN: ${GOLFOARAN_1_A_ACCESS_TOKEN}
      BROKER_HOST: ${BROKER_HOST}
      IP_TARGET: ${GOLFOARAN_1_A_IP_TARGET}
      SLEEP_TIME: ${SLEEP_TIME}

  GOLFOARAN_2_A:
    image: registry.gitlab.com/stamtech/industrial-automation-solutions/smart5grid
    environment:
      ACCESS_TOKEN: ${GOLFOARAN_2_A_ACCESS_TOKEN}
      BROKER_HOST: ${BROKER_HOST}
      IP_TARGET: ${GOLFOARAN_2_A_IP_TARGET}
      SLEEP_TIME: ${SLEEP_TIME}

```

```

replicaCount: 1
image:
  repository: "registry.gitlab.com/stamtech/energy-management-
platform/smart5grid/netmonitor"
  tag: "latest"

imagePullSecrets:
  - name: gitlab-credentials

config:
  ipTarget: 127.0.0.1
  brokerHost: 34.76.251.XXX
  accessToken: XXXXXXXXXXXXXXXX
  sleepTime: 300
  packetCount: 100
  packetInterval: 0.1

serviceAccount:
  create: false

ingress:
  enabled: false

autoscaling:
  enabled: false

```

To run the developed network application will then be sufficient, enter the project folder and run the following command:

“docker-compose up”

3. Pure Cloud-native Network Applications:

- 3.1. Create the Network Application descriptor and reference the Helm chart(s) as service(s) and the Docker image(s) as subservice(s).

To implement a descriptor for a Helm chart, you need to create a values.schema.json file that describes the structure of the values that can be used with the chart. Here are the steps to create a descriptor for a Helm chart:

- Create a values.schema.json file in the root directory of your Helm chart.
- Define the structure of the values that can be used with the chart using JSON schema.
- Use the helm lint command to validate your schema and ensure that it is valid.
- Include the values.schema.json file in the Helm chart by including it in the files section of the Chart.yaml file.

The following is the structure of the values file:

```

replicaCount: 1
image:
  repository: "registry.gitlab.com/stamtech/energy-management-
platform/smart5grid/netmonitor"
  tag: "latest"

imagePullSecrets:
  - name: gitlab-credentials

config:
  ipTarget: 127.0.0.1
  brokerHost: 34.76.251.246
  accessToken: xxxxxxxxxxxxxx
  sleepTime: 300
  packetCount: 100
  packetInterval: 0.1

serviceAccount:
  create: false

ingress:
  enabled: false

autoscaling:
  enabled: false

```

The following is the content of chart file

```

apiVersion: v2
name: netmonitor
description: Continuously check the connection to an IP for network statistics
and send data to MQTT

# A chart can be either an 'application' or a 'library' chart.
#
# Application charts are a collection of templates that can be packaged into
# versioned archives to be deployed.
#
# Library charts provide useful utilities or functions for the chart developer.
# They're included as a dependency of application charts to inject those
# utilities and functions into the rendering pipeline. Library charts do not
# define any templates and therefore cannot be deployed.
type: application

# This is the chart version. This version number should be incremented each time
# you make changes
# to the chart and its templates, including the app version.
# Versions are expected to follow Semantic Versioning (https://semver.org/)
version: 0.0.6

# This is the version number of the application being deployed.
# This version number should be incremented each time you make changes to the
# application. Versions are not expected to follow Semantic Versioning.
# They should reflect the version the application is using.
# It is recommended to use it with quotes.
appVersion: "1.0.0"

```

The network application descriptor file is necessary for the deployment. It describes the services used by the network application and the package files that install them:

```
netapp:
  im-version: 0.0.1
  name: Netapp-netmonitor
  description: Netmonitor from STAM
  provider: STAM
  version: 0.0.1
  service-format: helm
  services:
    - id: stam-netmonitor
      package: charts/stam-netmonitor-0.0.6.tgz
      images:
        - images/stam-netmonitor-0.0.6.tar.gz
      sap:
        - id: netmonitor
```

5. UC#2 Network Application(s)

UC2 Network application seeks to create an automated method for identifying personnel and the tools they are using while entering a primary power substation. Ultra-wideband (UWB) cameras and sensors are used for this detection, and they require quick processing and low latency. A private 5G network with edge computing capabilities was deployed in the pre-piloting testbed since the zones must be delineated in real-time. In order to monitor activities and construct a 3D volumetric security zone as well as to activate audio-visual, electronic, and physical alerts when necessary, the Real-Time Location System (RTLS), built as a use-case specific Network App for the Spanish Demo. In order to activate a danger alarm signal in the event that the employees or their tools are in the danger zone, the Network App receives the information gathered by the sensors, processes it, verifies the data, and evaluates it. Several preliminary procedures, including functional testing of the Network App and tests of communication between sensors and important 5G network components, were carried out as part of the pre-piloting operations. The sections of this chapter that follow provide a summary of those activities' specifics.

The second UC provides a mechanism that recognizes in real time when employees and their tools are exposed when working in the EcoGarraf substation when they enter a restricted area. Since electrical stations are high-risk locations with voltages over 66 kV, the safety of the employees who work there is of utmost concern to all energy operators. The greatest safety requirements will be upheld while ongoing adjustments are made to incorporate the most recent technological advancements. Through the development of various Network Apps, the 5G functions in the framework of Smart5Grid will strengthen the security measures and offer a safer working environment.

The current infrastructure uses two different safety systems: the first is a camera software with integrated image recognition that gives a view of the employees who are physically present in the substation, and the second is UWB (Ultra-Wideband), which provides basic location information through tags and anchors. The use of 5G technology to combine data from many sources, process them, and build communication channels represents a technological advance over the infrastructure that now exists.

5.1. Technical specification

The Network App installed in this UC#2 will continuously process the data coming in from the various sensors placed across the substation and utilize it to determine whether a worker has entered a restricted area. A notification or alarm will be set off if such a circumstance is discovered, alerting the employee to the breach and alerting him to the potential risk.

The motion sensors on the workers' uniforms and the cameras positioned around the job site transmit the coordinates to the Network App receiver subcomponent in the initial phase. Once the appropriate pre-processing has been done to normalize the various inputs and export the harmonized data flow, the input data are turned into the unified data flow and forwarded to the Network App computing subcomponent, where they are then forwarded into the Network App synchronization subcomponent. The worker is warned if he has entered the restricted area after the final choice regarding whether to sound the alarm or not is made.

5.2. Functional and Technical Requirements

The Network application for UC2 has several requirements both technical and functional in order to support the use case and ensure the high level of worker's safety.

- **Communication with Camera Sensor(s)**

The Network application should have the proper procedures to connect with the camera sensors in order to receive data.

- **Communication with UWB Tag Sensor(s)**

The Network application should have the proper procedures to connect with the UWB Tag sensors in order to receive data.

- **Synchronization function**

The Network application should have the proper procedures to receive and process data from multiple sources.

- **Login function**

The Network application should have the proper procedures to allow Login function with privileges and different levels of access.

- **Configuration**

The Network application should have the proper procedures to allow the changes and selection of safety zones through a dashboard depending on the different level of access.

- **MQTT Broker service**

The Network application should have a messaging bus that all components use to communicate messages. For that reason the usage of MQTT is utilized which is a lightweight messaging protocol to send and receive data, using a publish/subscribe model.

5.3. Network Application Functionality

The brief functionality of the application is that firstly, a set of safe zones coordinates are set in the configuration web app to begin the process and calculate alarm responses based on these values. The publish of these values is achieved by the mqttbroker which is responsible for sending the coordinates to the synchronisation app. Then, camera and uwb components have the capability to establish socket connections to exchange messages and calculate the exact position of workers in the substation area. After calculations, data is sent to the synchronisation component which decides, based on the safe zones, if it must send an alarm activation/deactivation to the camera or uwb. To achieve all of the requirements, different services were developed in the scope of the Network Application for UC2. The different services that comprise the application are:

- **Configuration component**
- **MQTT Broker component**
- **Synchronization component**
- **Camera socket component**
- **UWB socket component**

The application firstly provides the means of receiving data from two different sources named Camera and UWB Tag Sensors. This is achieved by the UWB socket and Camera socket components. After that, the processing of the data is made in the Synchronization component. There, the data from both systems are processed to see if an alarm should be emitted. The application also provides the functionality of changing/selecting the safety zones through a dashboard. This is performed by the administrator and several workers depending on their privileges and the credentials that they provide to the login page.

5.4. Integration & Deployment

Several steps have been made for the proper integration and deployment of this Network Application. The detailed steps can be found in D3.4 [18] which include the pre-piloting steps. The steps that we followed based on the Guide for developers of [Section 3](#) are:

- **Develop Network App logic (Python, C, java, etc)**

We developed the logic of the application using the programming language called Python, tested it locally and created the dependencies required.

- **Build image (Create a Dockerfile)**

After installing docker, we created the dockerfile that is needed for the next step.

- **Kubernetes deployment (Deploy your application with Kubernetes)**

Several Kubernetes deployments were tested for this application including locally clusters and the final cluster that will be used in UC2. The creation of the cluster was made with Kubernetes deployment configuration that was later used to expose our application and including services.

Name	Labels	Type	Cluster IP	Internal Endpoints	External Endpoints	Created ↑
configuration	app.kubernetes.io/managed-by: Helm	NodePort	10.102.220.66	configuration:8020 TCP configuration:30079 TCP	-	8 minutes ago
mqttbroker-svc	app.kubernetes.io/managed-by: Helm	ClusterIP	10.106.207.76	mqttbroker-svc:1883 TCP mqttbroker-svc:0 TCP	-	8 minutes ago
camerawebsocket2	app.kubernetes.io/managed-by: Helm	NodePort	10.108.84.163	camerawebsocket2:80 TCP camerawebsocket2:32081 TCP	-	9 minutes ago
camerawebsocket4	app.kubernetes.io/managed-by: Helm	NodePort	10.98.8.168	camerawebsocket4:80 TCP camerawebsocket4:32083 TCP	-	9 minutes ago
camerawebsocket5	app.kubernetes.io/managed-by: Helm	NodePort	10.109.187.100	camerawebsocket5:80 TCP camerawebsocket5:32084 TCP	-	9 minutes ago
camerawebsocket6	app.kubernetes.io/managed-by: Helm	NodePort	10.105.237.192	camerawebsocket6:80 TCP camerawebsocket6:32085 TCP	-	9 minutes ago
camerawebsocket1	app.kubernetes.io/managed-by: Helm	NodePort	10.106.188.13	camerawebsocket1:80 TCP camerawebsocket1:32080 TCP	-	9 minutes ago
camerawebsocket3	app.kubernetes.io/managed-by: Helm	NodePort	10.101.130.100	camerawebsocket3:80 TCP camerawebsocket3:32082 TCP	-	9 minutes ago
uwbsocet1	app.kubernetes.io/managed-by: Helm	NodePort	10.108.253.196	uwbsocet1:9015 UDP uwbsocet1:31258 UDP	-	9 minutes ago
uwbtalarm9011	app.kubernetes.io/managed-by: Helm	ClusterIP	10.96.93.35	uwbtalarm9011:9011 UDP uwbtalarm9011:0 UDP	-	9 minutes ago

Figure 14. UC2 Network Application - Kubernetes services

- **Encapsulate this K8s files in Helm-chart**

The K8s files from the previous step, were encapsulated In Helm charts, after we installed, configured and familiarise ourselves with this packaging format. Our deployment on Kubernetes consists of two helm charts:

1. uc2-main-components-helm-chart
2. uc2-sensors-helm-chart

In the main helm chart we deploy 3 containerized applications:

1. Configuration
2. Synchronisation
3. MqttBroker

- **Add Helm-chart in Network App descriptor**

The Network application will be created by the 2 helm chart packages that we have described above and will be integrated into the Network App descriptors. The first step is to package the chart with the appropriate helm command. That will create a compressed tar file containing the chart files, which can then be shared with other users or stored in a chart repository. In our case, we can add a reference to our Network App descriptors to use these packaged helm charts.

```
1 vnfd:
2   description: Main Components of UC2 NetApp
3   df:
4     - id: default-df
5   ext-cpd:
6     - id: configuration-ext
7       k8s-cluster-net: configuration
8   id: main_vnfd
9   mgmt-cp: eth0-ext
10  product-name: main_vnfd
11  provider: SID
12  k8s-cluster:
13    nets:
14      - id: mgmtnet
15  kdu:
16    - name: main
17      helm-chart: uc2-main-components-helm-chart/uc2-main-components-helm-chart
18  version: '1.0'
```

Figure 15. UC2 Network Application - VNF Descriptor with helm chart package integration

6. UC#3 Network Application(s)

The Network Application for UC3 provides real-time data monitoring (i.e., in milliseconds from transformer), of the wind farm production to the owner and the power system operator (i.e., TSO) gathering measurements from sensors capturing the performance of key components of the wind turbines. The services that comprise the UC3 Network Application are two named real-time performance monitoring (RTPM) and Predictive maintenance (PM). The network application is installed on a virtual machine in the VIVACOM cloud. The virtual machine hosts Kubernetes cluster that is managed by the NetAppController software. Each service has a web interface which is accessible on a specific EndPoint - IP address and port.

6.1. Technical specification

The UC3 Network Application has two components, which are the predictive maintenance enabler and the real-time energy performance monitoring and they are implemented as different VNFs, which are interconnected and interact with the input sources in order to provide the necessary outputs to fulfil the functional requirements of this UC, as shown in Figure 16.

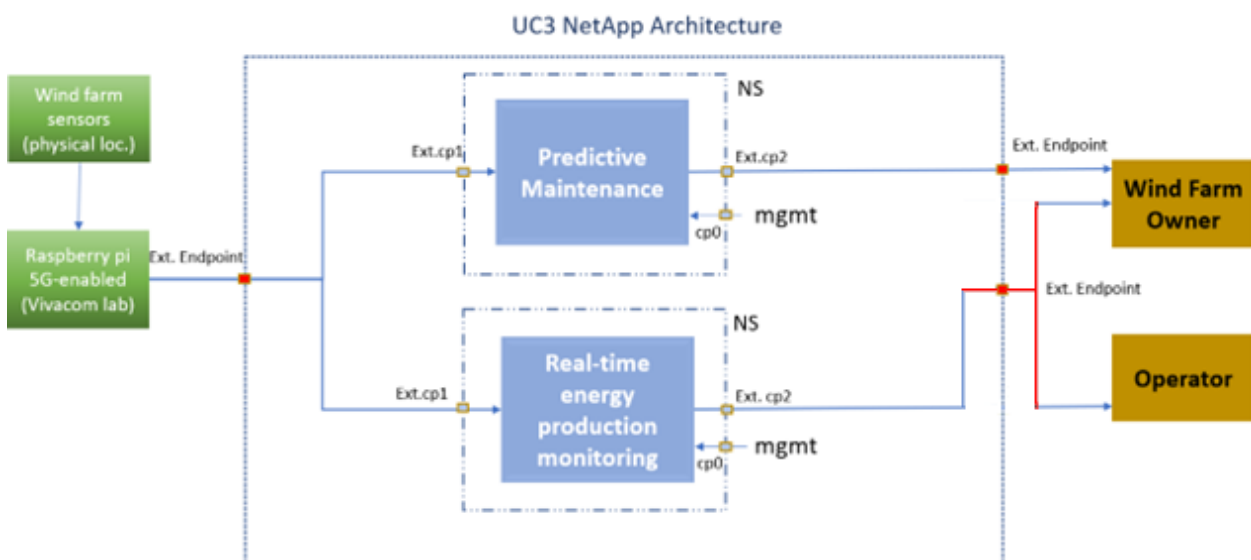


Figure 16. UC3 Network Application Architecture

The predictive maintenance VNF has two external CPs, one as an input to collect the data from the Raspberry Pi supporting 5G connectivity, located at the Vivacom lab, and the other one as an output to provide the outcome of the internal processes to the wind farm owner.

The real-time operation VNF has two external CPs, one as an input to collect the data from the Raspberry Pi supporting 5G connectivity, located at the Vivacom lab, and one as an output to inform both the wind farm owner and operator about real-time production.

The two components have a management CP in order to grant access for configuration or debugging purposes.

6.2. Functional and Technical Requirements

6.2.1. RTPM

- Data transfer protocol support:

MQTT protocol has been selected as basic communication control between RES devices and NET App. Supported version of the MQTT protocol are version 5, version 3.1.1 and version 3.1.

- **Data aggregation with time alignment to relative time:**

The RTPM aligns data received from MQTT Broker topics and transmits the data for archiving and visualization.

- **Device List**

Visualization of list of connected devices.

- **Location**

A map indicating device's location in the power system.

- **Device information**

The device's name, address, type, connection properties.

- **Power information**

The total energy production, generator active and reactive power information is displayed

- **Environment information**

Parameters of the environment are displayed

- **Technical information**

Displayed technical parameters as grid side phases, frequency, temperature measurement of generator rotor phases and etc.

- **Monitoring**

Real-time signal monitoring for each device in the list. Support several different visualizations depending of the type of device: wind turbine, hydropower turbine, solar park, sensor or particular IoT device.

- **Alarms visualization**

RTPM service will show general alarms to the user after reading input stream.

- **Cyber security**

Cyber security will be evaluated by all RTPM interfaces, while maintaining the reliability of the service.

6.2.2. PM

- **Alarms**

PM service will implement different calculations and according to their results will detect abnormality in the device.

- **Alarms log**

PM service will keep log of all raised alarms.

- **Keeping historical data in a database**

PM service will save the received data(signals) in a database.

- **Export data to other applications for analysis**

PM service will export the saved historical data to other applications like Excel and Grafana for analysis.

6.3. Network Application Functionality

6.3.1. Common functionality of the services

- **Configuration of the services.**

Firstly we have the configuration of the services. Each service has a SQL configuration script. That script is part of the “values.yaml” file of the Helm chart of the service. The configuration of the script can be done by the administrator. The script describes input data provided to the service. For example, this is a simple configuration script of RTPM service:

```
INSERT INTO Device(Acronym, Name, AccessID, ProtocolID, Longitude, Latitude, ConnectionString,
MeasuredLines)
```

```
VALUES('WIND', 'WIND', 2, 16, 26.309964, 42.775758, 'Host=212.72.214.236; Port=30583; Topic=Aris/full;
Source=wind; Type=wind; Description=Wind Turbine', 3);
```

```
INSERT INTO Measurement( DeviceID, PointTag, SignalTypeID, SignalReference, Description)
VALUES((SELECT ID FROM Device where acronym='WIND'), 'WI_HORWDSPD', (SELECT ID FROM
SignalType where acronym='WSPD'), 'WI_HORWDSPD', 'Horizontal wind speed');
```

```
INSERT INTO Measurement( DeviceID, PointTag, SignalTypeID, SignalReference, Description)
VALUES((SELECT ID FROM Device where acronym='WIND'), 'WI_W', (SELECT ID FROM SignalType where
acronym='ACPR'), 'WI_W', 'Generator active power');
```

```
INSERT INTO Measurement( DeviceID, PointTag, SignalTypeID, SignalReference, Description)
VALUES((SELECT ID FROM Device where acronym='WIND'), 'WI_HORWDSPDEST', (SELECT ID FROM
SignalType where acronym='WSPD'), 'WI_HORWDSPDEST', 'Horizontal wind speed estimated');
```

```
INSERT INTO Measurement( DeviceID, PointTag, SignalTypeID, SignalReference, Description)
VALUES((SELECT ID FROM Device where acronym='WIND'), 'WI_TOTVARH', (SELECT ID FROM SignalType
where acronym='TREP'), 'WI_TOTVARH', 'Total (net) reactive energy production');
```

```
INSERT INTO Measurement( DeviceID, PointTag, SignalTypeID, SignalReference, Description)
VALUES((SELECT ID FROM Device where acronym='WIND'), 'WI_TOTWH', (SELECT ID FROM SignalType
where acronym='TAEP'), 'WI_TOTWH', 'Total (net) active energy production');
```

```
INSERT INTO Measurement( DeviceID, PointTag, SignalTypeID, SignalReference, Description)
VALUES((SELECT ID FROM Device where acronym='WIND'), 'WI_ENVTMP', (SELECT ID FROM SignalType
where acronym='ENVT'), 'WI_ENVTMP', 'Temperature of environment');
```

```
INSERT INTO Measurement( DeviceID, PointTag, SignalTypeID, SignalReference, Description)
VALUES((SELECT ID FROM Device where acronym='WIND'), 'WI_VAR', (SELECT ID FROM SignalType
where acronym='REPR'), 'WI_VAR', 'Generator reactive power');
```

```
INSERT INTO Measurement( DeviceID, PointTag, SignalTypeID, SignalReference, Description)
VALUES((SELECT ID FROM Device where acronym='WIND'), 'WI_GRIHZ', (SELECT ID FROM SignalType
where acronym='FREQ'), 'WI_GRIHZ', 'Frequency');
```

```
INSERT INTO Measurement( DeviceID, PointTag, SignalTypeID, SignalReference, Description)
VALUES((SELECT ID FROM Device where acronym='WIND'), 'WI_ALARM', (SELECT ID FROM SignalType
where acronym='ALRM'), 'WI_ALARM', 'Alarm status');
```

```
INSERT INTO Measurement( DeviceID, PointTag, SignalTypeID, SignalReference, Description)
VALUES((SELECT ID FROM Device where acronym='WIND'), 'WI_GRIW', (SELECT ID FROM SignalType
where acronym='GRPR'), 'WI_GRIW', 'Grid side power');
```

```
INSERT INTO Measurement( DeviceID, PointTag, SignalTypeID, SignalReference, Description)
VALUES((SELECT ID FROM Device where acronym='WIND'), 'WI_GRIPF', (SELECT ID FROM SignalType
where acronym='GSPF'), 'WI_GRIPF', 'Grid side 3 phase power factor');
```

```
INSERT INTO Measurement( DeviceID, PointTag, SignalTypeID, SignalReference, Description)
VALUES((SELECT ID FROM Device where acronym='WIND'), 'WI_ROTSPD', (SELECT ID FROM SignalType
where acronym='RSPD'), 'WI_ROTSPD', 'Rotational speed');
```

```
INSERT INTO Measurement( DeviceID, PointTag, SignalTypeID, SignalReference, Description)
VALUES((SELECT ID FROM Device where acronym='WIND'), 'WI_ROTTMP1', (SELECT ID FROM SignalType
where acronym='TMPR'), 'WI_ROTTMP1', 'Temperature measurements for generator rotor, phase 1');
```

```
INSERT INTO Measurement( DeviceID, PointTag, SignalTypeID, SignalReference, Description)
VALUES((SELECT ID FROM Device where acronym='WIND'), 'WI_ROTTMP2', (SELECT ID FROM SignalType
where acronym='TMPR'), 'WI_ROTTMP2', 'Temperature measurements for generator rotor, phase 2');
```

```
INSERT INTO Device(Acronym, Name, AccessID, ProtocolID, Longitude, Latitude, ConnectionString,
MeasuredLines)
```

```
VALUES('HYDRO', 'HYDRO', 2, 16, 24.042109, 41.723833, 'Host=212.72.214.236; Port=30583;
Topic=Hydro/full; Source=hydro; Type=hydro; Description=Hydropower Turbine', 3);
```

```
INSERT INTO Measurement( DeviceID, PointTag, SignalTypeID, SignalReference, Description)
VALUES((SELECT ID FROM Device where acronym='HYDRO'), 'HY_PW', (SELECT ID FROM SignalType
where acronym='ACPR'), 'HY_PW', 'Active Power');
```

```
INSERT INTO Measurement( DeviceID, PointTag, SignalTypeID, SignalReference, Description)
VALUES((SELECT ID FROM Device where acronym='HYDRO'), 'HY_GV', (SELECT ID FROM SignalType
where acronym='DIGI'), 'HY_GV', 'Guide Vanes');
```

```
INSERT INTO Measurement( DeviceID, PointTag, SignalTypeID, SignalReference, Description)
VALUES((SELECT ID FROM Device where acronym='HYDRO'), 'HY_WL', (SELECT ID FROM SignalType
where acronym='DIGI'), 'HY_WL', 'Water basin level');
```

```
INSERT INTO Measurement( DeviceID, PointTag, SignalTypeID, SignalReference, Description)
VALUES((SELECT ID FROM Device where acronym='HYDRO'), 'HY_ALARM', (SELECT ID FROM SignalType
where acronym='ALRM'), 'HY_ALARM', 'Alarm status');
```

```
INSERT INTO Device(Acronym, Name, AccessID, ProtocolID, Longitude, Latitude, ConnectionString,
MeasuredLines)
```

```
VALUES('SOLAR', 'SOLAR', 2, 16, 24.783791, 42.631622, 'Host=212.72.214.236; Port=30583;
Topic=Solar/full; Source=solar; Type=solar; Description=Solar Farm', 3);
```

```
INSERT INTO Measurement( DeviceID, PointTag, SignalTypeID, SignalReference, Description)
VALUES((SELECT ID FROM Device where acronym='SOLAR'), 'PV_AP', (SELECT ID FROM SignalType where
acronym='ACPR'), 'PV_AP', 'Active power');
```

```
INSERT INTO Measurement( DeviceID, PointTag, SignalTypeID, SignalReference, Description)
VALUES((SELECT ID FROM Device where acronym='SOLAR'), 'PV_RP', (SELECT ID FROM SignalType where
acronym='REPR'), 'PV_RP', 'Reactive power');
```

```
INSERT INTO Measurement( DeviceID, PointTag, SignalTypeID, SignalReference, Description)
VALUES((SELECT ID FROM Device where acronym='SOLAR'), 'PV_SI', (SELECT ID FROM SignalType where
acronym='IRRD'), 'PV_SI', 'Solar irradiance');
```

- **Login**

Then we have the login functionality for each of the services that is performed by *user name* and *password* that are provided by the administrator. The login interface is shown in Figure 17.




Figure 17. UC3 Network Application - Login Interface

- **Service state**

Each service has service state menu item for monitoring of technical parameters of the service as shown in Figure 18 below.

Devices		Service State	Log Out			
Category	Counter	Last	Average	Maximum	Units	
Process	Process CPU Usage	2.60	1.30	24.30	Average % / CPU	
Process	Process Thread Count	39.00	39.00	39.00	System Threads	
Process	Process Memory Usage	214.30	183.28	214.30	Megabytes	
.NET CLR LocksAndThreads	Thread Queue Size	0.00	0.00	0.00	Waiting Threads	
.NET CLR LocksAndThreads	Lock Contention Rate	5.80	6.87	19.60	Attempts / sec	
.NET CLR Memory	CLR Memory Usage	0.00	0.00	0.00	Megabytes	
.NET CLR Memory	Large Object Heap	0.00	0.00	0.00	Megabytes	
.NET CLR Exceptions	Exception Count	14934.00	14599.08	14934.00	Total Exceptions	
Network Interface	IP Outgoing (eth0)	4536.47	9002.72	662328.00	Bytes / sec	
Network Interface	IP Incoming (eth0)	1655.89	683.46	20704.08	Bytes / sec	
Network Interface	IP Outgoing (lo)	959.51	257.11	7030.87	Bytes / sec	
Network Interface	IP Incoming (lo)	959.51	257.11	7030.85	Bytes / sec	

Figure 18. UC3 Network Application - Service state

- List of devices


RTPM service shows a list of connected devices.

Devices

Service State

Log Out

Project Info



Demonstration of 5G solutions for SMART energy GRIDs of the future

UTC Time

Server Time

04-11-2022 01:37:48.396

Client Time

04-11-2022 01:37:48.396

Local Time (PM)


Server Time

04-Nov-2022 03:37:48.396

Client Time

04-Nov-2022 03:37:48.396

4 Registered Devices



HYDRO


Hydropower Turbine

Host: 212.72.214.154 Port: 1883 Protocol: MQTT

Topic: Hydro/full Location(lat,lon): (41.723833, 24.042109)

Data

Info



SOLAR


Solar Farm

Host: 212.72.214.154 Port: 1883 Protocol: MQTT

Topic: Solar/full Location(lat,lon): (42.631622, 24.783791)

Data

Info



UCY_HIL

UCY - UC3 HIL demo Signal list and MQTT broker connection

Host: 159.89.103.242 Port: 1883 Protocol: MQTT

Topic: UCY_HIL Location(lat,lon): (35.160790, 33.377010)

Data

Info

Figure 19. UC3 Network Application - List of devices (RTPM)

Supported type of devices have their own icons, also RTPM provides the default icon for other IoT devices.

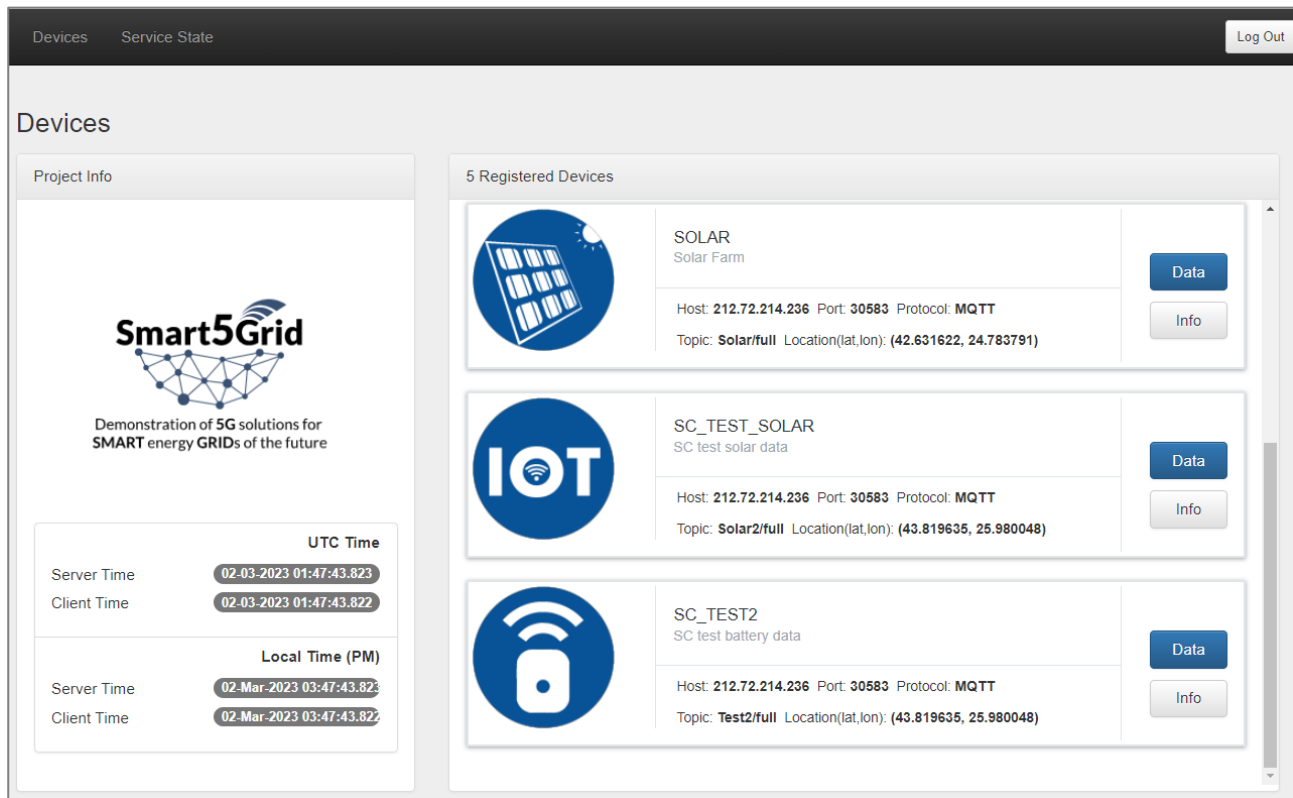


Figure 20. UC3 Network Application - Default IoT icon (RTPM)

- **Device properties**

The device properties are shown in each service. The page has three areas. Top left area contains information about device and device connection. Top right area presents device location on the map. The bottom area – table contains device signals abbreviation, description and run time values.

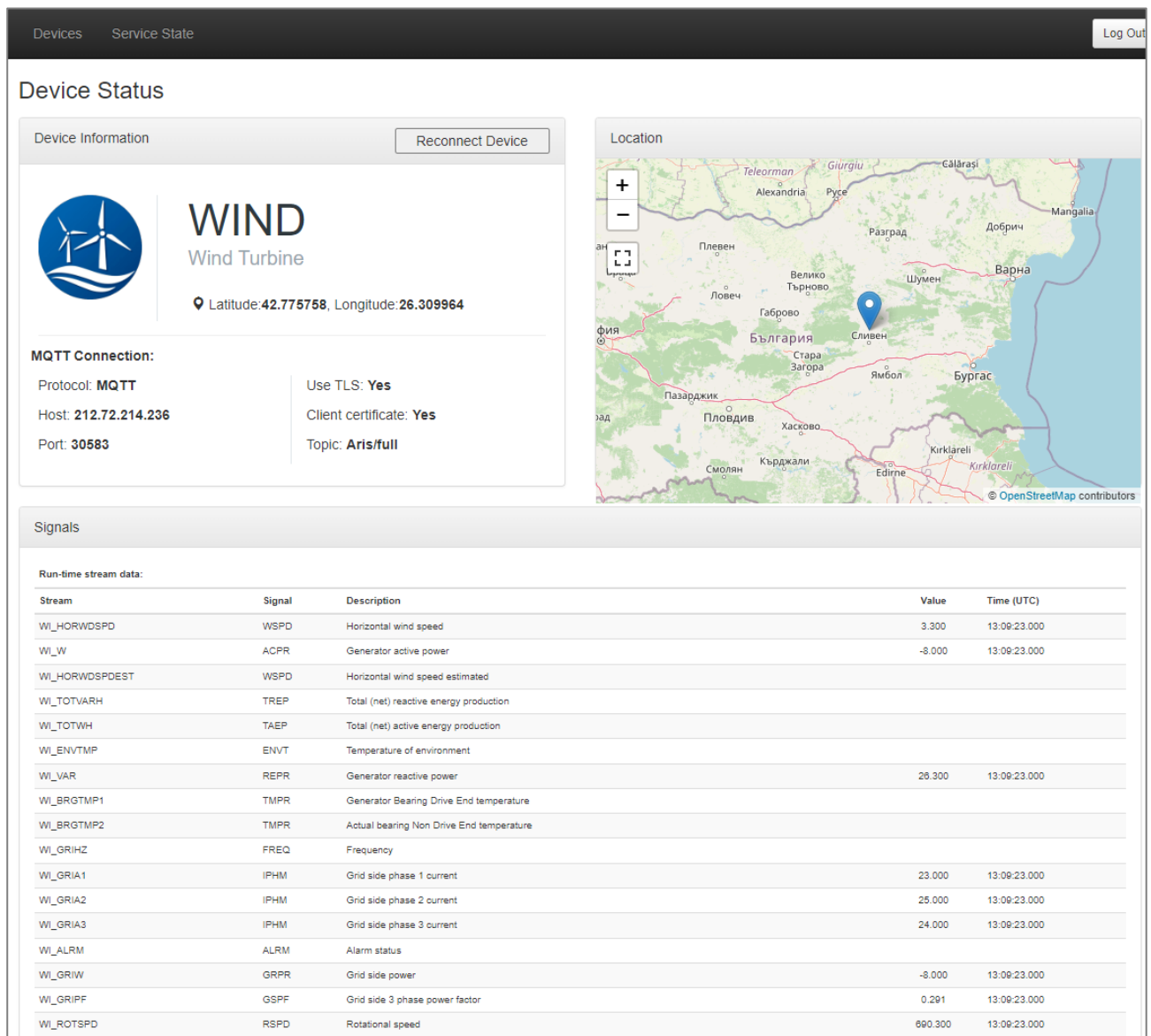


Figure 21. UC3 Network Application - Device Status interface

6.3.2. MQTT Broker service

This service is Eclipse Mosquitto open-source message broker which implements MQTT protocol version 5, version 3.1.1 and version 3.1. This protocol is selected as main communication protocol for the Network App. RES Source devices are configured as MQTT Publishers. RTPM and PM services are configured as MQTT subscribers for specific – defined in connection string topics. To secure communication between clients and server the MQTT Broker uses server/client certificates method for authentication of the clients.

6.3.3. RTPM service

RTPM service visualizes the RES source device data. There are different visualizations for different RES sources. The layout of the application contains two main areas: Device signal area located on the left and chart area. The user has ability to select one of more signals and their visual representation of the charts.

For Wind turbine visualization signals are divided in three tabs: energy production, environment params and technical data. The Energy production signals interface can be seen at Figure 22, the Environment signals can be seen at Figure 23 and the Technical signals at Figure 24.



Figure 22. UC3 Network Application - Wind turbine visualization (Energy production signals)

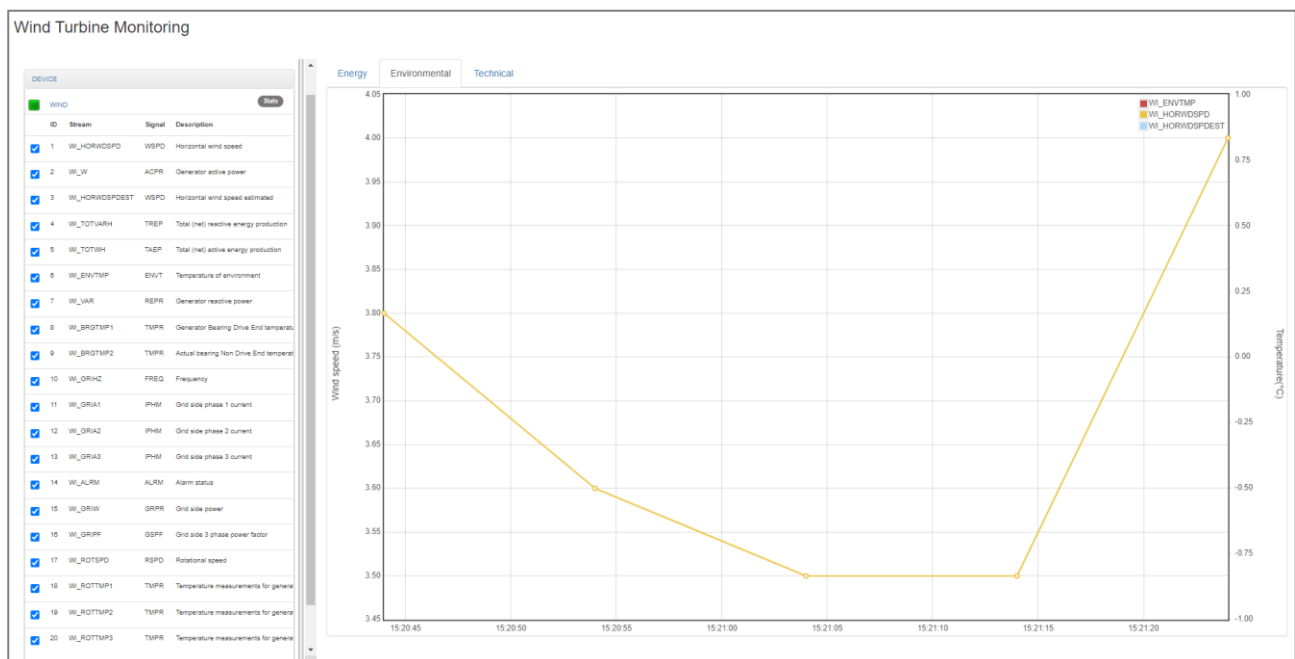


Figure 23. UC3 Network Application - Wind turbine visualization (Environment signals)

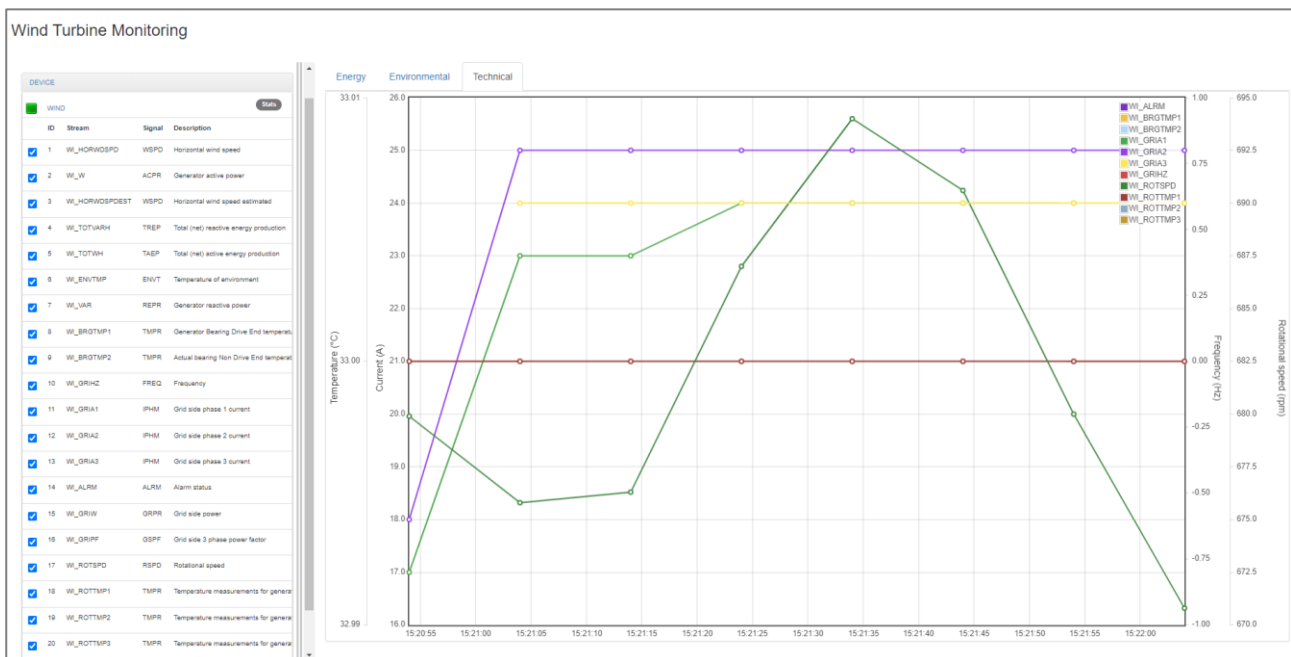


Figure 24. UC3 Network Application - Wind turbine visualization (Technical signals)

The layout of Hydro Turbine, Solar Park and IoT devices not in the list of supported RES devices are similar.

6.3.4. Predictive Maintenance Enabler service

Gathering measurements from sensors capturing the performance of key components of the wind turbines, and thus offering to the wind farm owner information regarding the asset performance of the wind farm, and to the power system operator (i.e., TSO) information about operational availability of the asset.

Exploring measurements from the database.

- **SELECT SIGNALS.**

User could select signal using checkbox.

Measurements Analysis Alarms Alarms log Service State Log Out

Select/Visualize/Export Measurements Instance: WIND Records: 20

Selected Points 0 Save Load Sort Clear All

Select Points Trend Data Export Data

ID	Type	Tag Name	Current Value	Signal Reference	Description
<input type="checkbox"/> 14	ALRM	WI_ALRM	NaN	WI_ALRM	Alarm status
<input type="checkbox"/> 8	TMPR	WI_BRGTMP1	NaN	WI_BRGTMP1	Generator Bearing Drive End temperature
<input type="checkbox"/> 9	TMPR	WI_BRGTMP2	NaN	WI_BRGTMP2	Actual bearing Non Drive End temperature
<input type="checkbox"/> 6	ENV	WI_ENVTMP	NaN	WI_ENVTMP	Temperature of environment
<input type="checkbox"/> 11	IPHM	WI_GRIA1	9.000	WI_GRIA1	Grid side phase 1 current
<input type="checkbox"/> 12	IPHM	WI_GRIA2	8.000	WI_GRIA2	Grid side phase 2 current
<input type="checkbox"/> 13	IPHM	WI_GRIA3	8.000	WI_GRIA3	Grid side phase 3 current
<input type="checkbox"/> 10	FREQ	WI_GRIHZ	NaN	WI_GRIHZ	Frequency
<input type="checkbox"/> 16	GSPF	WI_GRIPF	0.907	WI_GRIPF	Grid side 3 phase power factor
<input type="checkbox"/> 15	GRPR	WI_GRIW	-7.000	WI_GRIW	Grid side power
<input type="checkbox"/> 1	WSPD	WI_HORWDSPD	1.300	WI_HORWDSPD	Horizontal wind speed
<input type="checkbox"/> 3	WSPD	WI_HORWDSPDEST	NaN	WI_HORWDSPDEST	Horizontal wind speed estimated
<input type="checkbox"/> 17	RSPD	WI_ROTSPD	68.600	WI_ROTSPD	Rotational speed

Export CSV

1 of 2

Figure 25. UC3 Network Application - Select signals visualization

- **VISUALIZE DATA FOR THE SELECTED SIGNAL(S)**

Visualization of historical data for selected signals for defined time interval

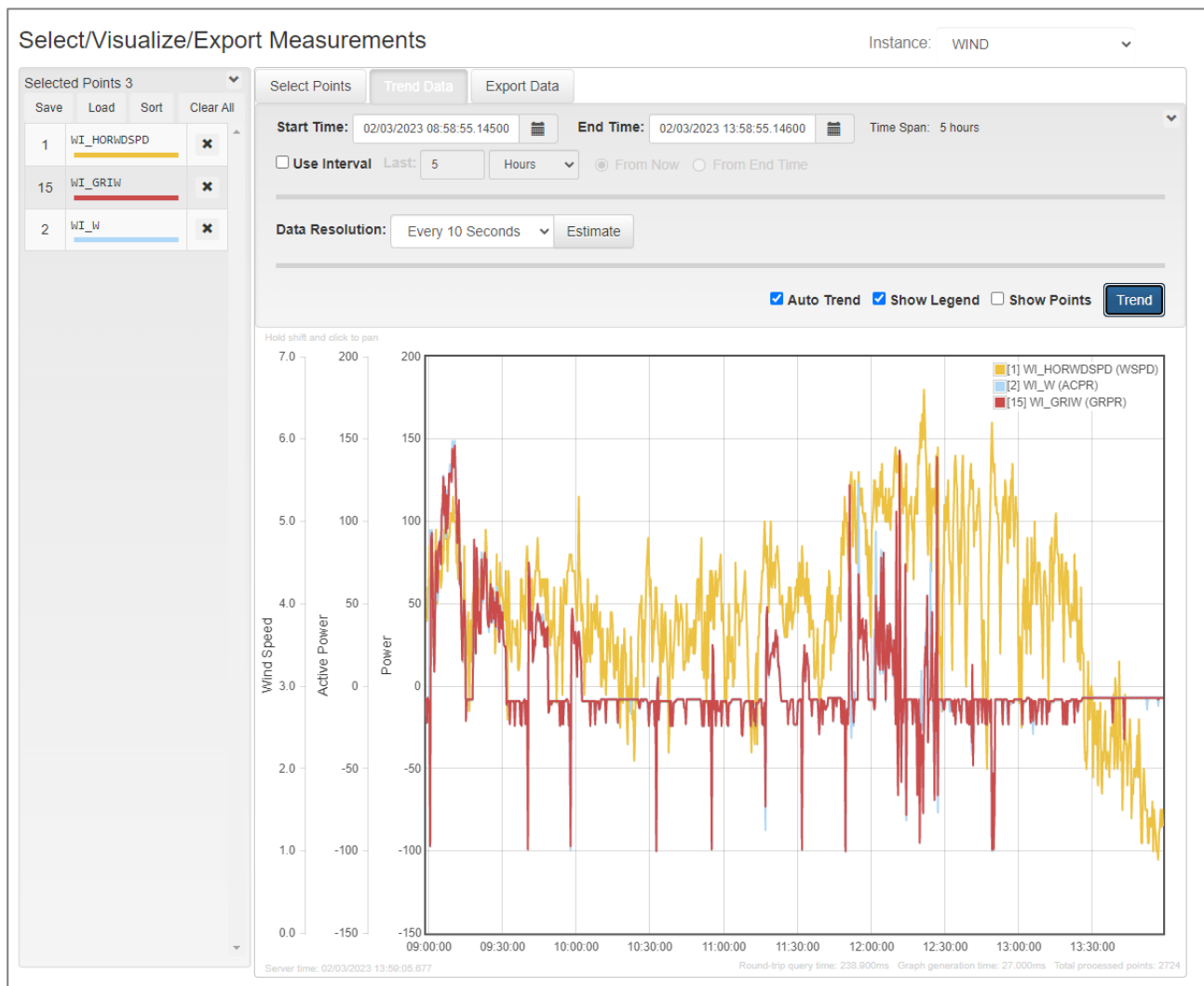


Figure 26. UC3 Network Application - Historical signal data visualization

- EXPORT DATA FOR THE SELECTED SIGNAL(S)

Export to Excel file

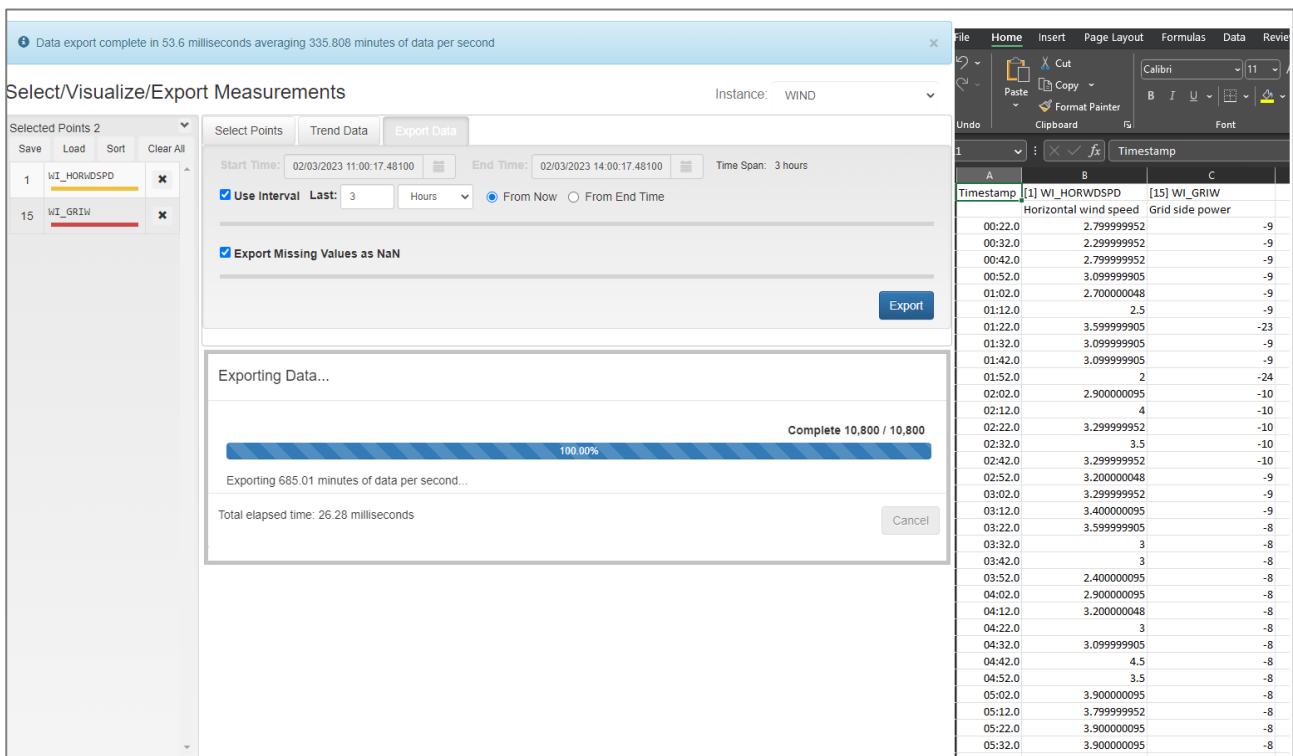


Figure 27. UC3 Network Application - Export Data graphical interface

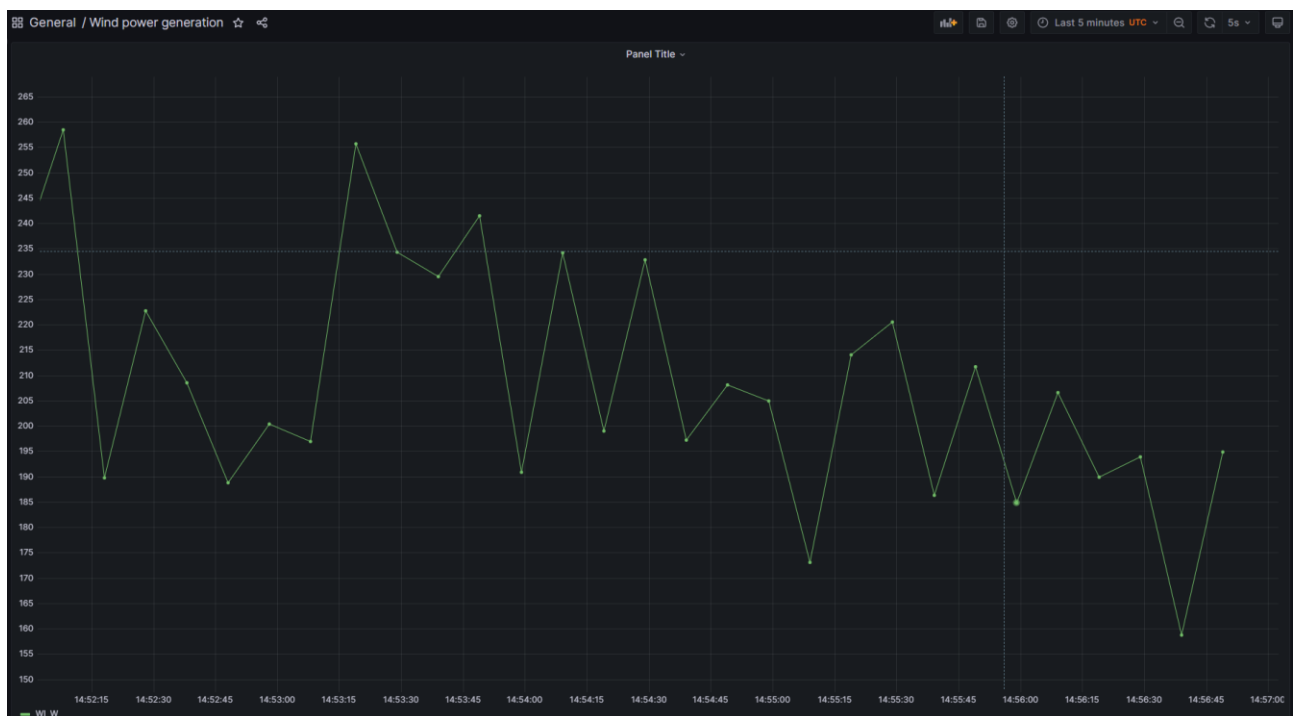


Figure 28. UC3 Network Application - Graph from exported data

- Alarms

Alarms conditions are defined in the configuration script. The raised alarms are shown in Alarms page. All alarm actions are saved in alarms log. The log is visible in "Alarms log" page.

Measurements

Analysis

Alarms

Alarms log

Service State

Log Out

ID	Tag Name	Severity	State	Time Raised	Value at Time Raised	Description	Operation	Set Point	Delay	Hysteresis	
2	ML_ROTSPD_LESS	300	1	02/03/2023 11:55:17	54.90000015258709	Rotational speed less than 60 rpm		22	60	1	0
4	ML_H0RMSPD_LESS	300	1	02/03/2023 11:55:57	8.899999976158142	Horizontal wind speed less than 1 m/s		22	1	1	0

Figure 29. UC3 Network Application - Alarm page user interface

6.4. Integration & Deployment

Each service is transformed into a Docker image and for each service a Helm chart is created. Helm chart is a collection of files that describe a related set of Kubernetes resources necessary for the service. Helm is the package manager for Kubernetes used to install the service in Kubernetes cluster using the correspondent Helm chart. The links to the Helm charts of the services are included in the Network App descriptor file.

The operating system Linux (Ubuntu 22.10) is used in Docker images. The services are developed with Visual Studio 2022 tool using C# programming language and .NET Framework 4.8.1. The web interface is developed using JavaScript. The database used for the configuration of the services is PostgreSQL 14.

6.4.1. Docker images of the services

The Docker Images of RTPM and PM services are almost the same. The Docker Image of Advisory service is most complicated, it is created by the following Docker file:

```
FROM ubuntu:22.10

WORKDIR /app

ARG DEBIAN_FRONTEND=noninteractive

RUN apt-get update

# install ca-cert

RUN apt-get install -y ca-certificates

COPY ./build/Certificates/ca.crt /app/ca.crt

RUN cp /app/ca.crt /usr/local/share/ca-certificates

RUN update-ca-certificates

# install mono

RUN apt-get update

RUN apt -y install -y mono-xsp4

# install postgresql-14
```



```

RUN apt -y install postgresql-14
USER postgres
RUN echo "host all    0.0.0.0/0   trust" >> /etc/postgresql/14/main/pg_hba.conf &&\
    echo "listen_addresses='*'" >> /etc/postgresql/14/main/postgresql.conf &&\
    /etc/init.d/postgresql start &&\
    createuser -s -i -d -r -l -w root &&\
    psql --command "ALTER USER postgres WITH PASSWORD '#S5GValentin';" &&\
    /etc/init.d/postgresql stop

#expose ports

# 5432 - PostgreSQL

# 80 - web interface of RTPM

EXPOSE 443

#copy files
USER root
COPY ./PostgreSQL/CreateRTPM.sql /app/PostgreSQL/
COPY ./PostgreSQL/InitialDataSet.sql /app/PostgreSQL/
COPY ./start.sh /app/
COPY ./createdb.sh /app/

# configure RTPM database in PostgreSQL
RUN /app/createdb.sh
COPY ./build/ /opt/RTPM/
COPY ./config/ /opt/RTPM/
CMD ["/app/start.sh"]

```

6.4.2. Helm charts of the services.

The Helm charts of RTPM and PM services are almost the same. The creation of RTPM helm chart is shown as an example.

Creating empty rtpm helm chart with the following command:

```
helm create rtpm
```

The command will create rtpm folder with several files:

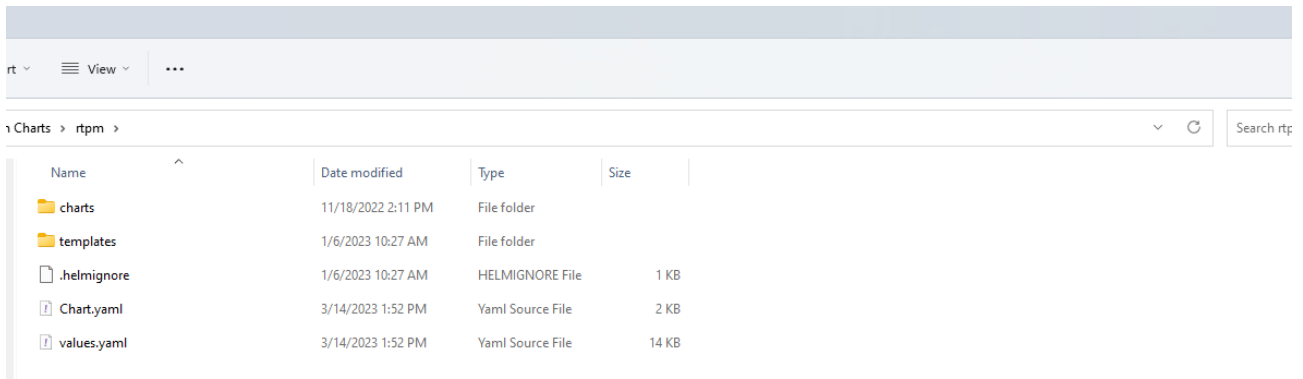


Figure 30. UC3 Network Application - Folder and files output of helm command

In the Helm chart we need to define the following items:

- The image that should be installed.
- The service Kubernetes component with the external port that will be used to access the application.
- The startupProbe used for detecting the successful starting.
- The name of the component.

The edit output of the “values.yaml” file can be observed at Figure 31.

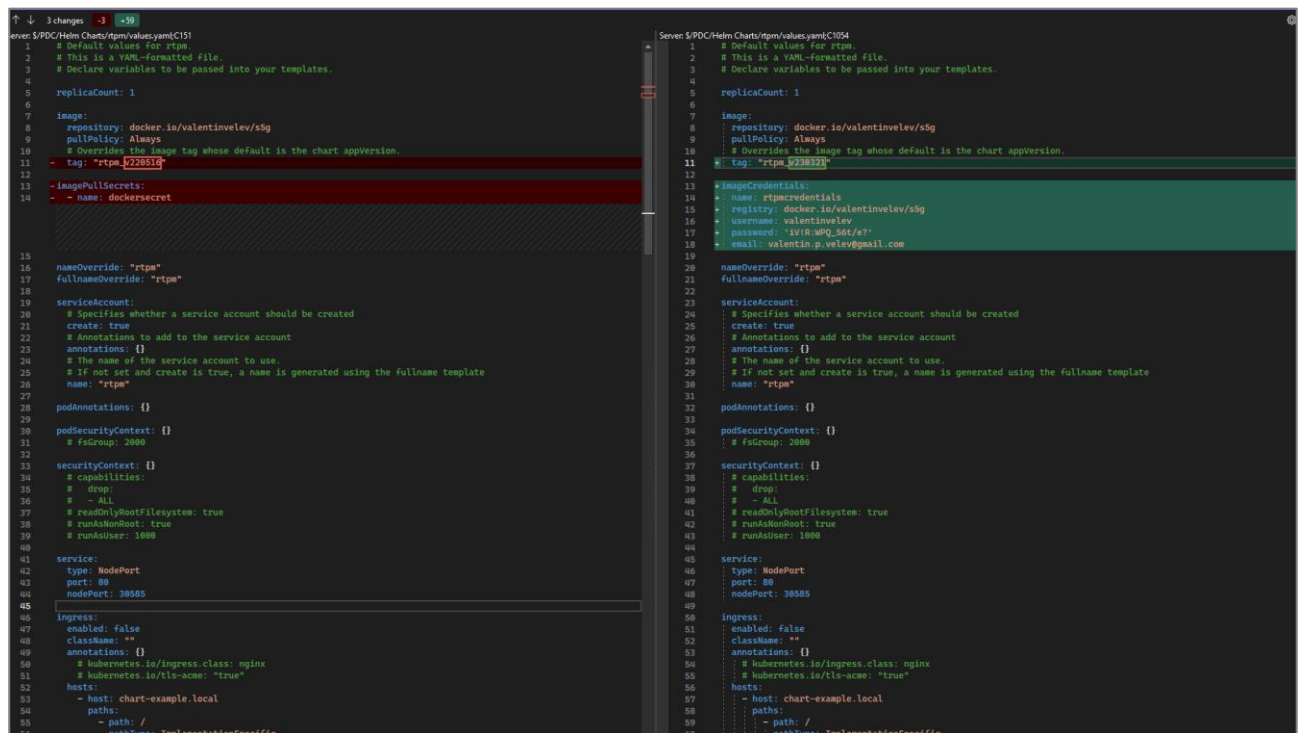


Figure 31. UC3 Network Application – YAML(values) file edit

In the image section the location of the image that will be installed is described.

repository: docker.io/valentinevelev/s5g

tag: "rtpm_test"

pullPolicy: Always (this change is optional)

imagePullSecrets:

- name: dockersecret (this is the name of the secret object that contain the name and password for the access to the repository)

Filling the name of the chart

nameOverride: "rtpm"

fullnameOverride: "rtpm"

serviceAccount:

name: "rtpm"

In the service section we define the parameters used in service template:

type: NodePort

port: 80

nodePort: 30584

There are three different types of Kubernetes service, we use the type NodePort.

The nodePort is the port used by the external applications for accessing the WAM service.

The edit output of the "templates\deployment.yaml" file which adds the following section can be observed at Figure 32.

```

1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: {{ include "rtpm.fullname" . }}
5    labels:
6      {{- include "rtpm.labels" . | nindent 4 }}
7  spec:
8    {{- if not .Values.autoscaling.enabled }}
9    replicas: {{ .Values.replicaCount }}
10   {{- end }}
11   selector:
12     matchLabels:
13       {{- include "rtpm.selectorLabels" . | nindent 6 }}
14   template:
15     metadata:
16       {{- with .Values.podAnnotations }}
17       annotations:
18         {{- toYaml . | nindent 8 }}
19       {{- end }}
20     labels:
21       {{- include "rtpm.selectorLabels" . | nindent 8 }}
22     spec:
23       imagePullSecrets:
24         - name: {{ .Values.imageCredentials.name }}
25       serviceAccountName: {{ include "rtpm.serviceAccountName" . }}
26       securityContext:
27         {{- toYaml .Values.podSecurityContext | nindent 8 }}
28       containers:
29         - name: {{ .Chart.Name }}
30           securityContext:
31             {{- toYaml .Values.securityContext | nindent 12 }}
32           image: "{{ .Values.image.repository }}:{{ .Values.image.tag | default .Chart.AppVersion }}"
33           imagePullPolicy: {{ .Values.image.pullPolicy }}
34           ports:
35             - name: http
36               containerPort: 80
37               protocol: TCP
38           livenessProbe:
39             httpGet:
40               path: /
41               port: http
42           readinessProbe:
43             httpGet:
44               path: /
45               port: http
46           startupProbe:
47             httpGet:
48               path: /
49               port: http
50             failureThreshold: 6
51             periodSeconds: 30
52           resources:
53             {{- toYaml .Values.resources | nindent 12 }}
54           volumeMounts:
55             - name: rtpmconf
56               mountPath: /config
57               readOnly: true
58           volumes:
59             - name: rtpmconf
60               configMap:
61                 name: cm-rtpm-config
62             {{- with .Values.nodeSelector }}
63             nodeSelector:
64               {{- toYaml . | nindent 12 }}
65             {{- end }}
66             {{- with .Values.affinity }}
67             affinity:
68               {{- toYaml . | nindent 12 }}
69             {{- end }}
70             {{- with .Values.tolerations }}
71             tolerations:
72               {{- toYaml . | nindent 12 }}
73             {{- end }}

```

Figure 32. UC3 Network Application – YAML(deployment) file edit

startupProbe:

httpGet:

path: /

port: http

failureThreshold: 6

periodSeconds: 30

Define startupProbe with the path "/" on the "http" port with the number of failures "6" and period "30" seconds. Kubernetes will try to load the main web page of the web service, when loading is successful, the Kubernetes will know that RTPM web service is started.

The edit output of the "templates\service.yaml" file which adds the following properties can be observed at Figure 33.

```

1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: {{ include "rtpm.fullname" . }}
5    labels:
6      {{- include "rtpm.labels" . | nindent 4 }}
7  spec:
8    type: {{ .Values.service.type }}
9    ports:
10     - port: {{ .Values.service.port }}
11       targetPort: {{ .Values.service.port }}
12       protocol: TCP
13       name: http
14       nodePort: {{ .Values.service.nodePort }}
15     selector:
16       {{- include "rtpm.selectorLabels" . | nindent 4 }}
17

```

Figure 33. UC3 Network Application – YAML(service) file edit

targetPort: {{ .Values.service.port }}

nodePort: {{ .Values.service.nodePort }}

The targetPort and nodePort will get the values defined in "values.yaml" file.

6.4.3. Network App descriptor

The network application descriptor file is necessary for the deployment. It describes the services used by the network application and the package files that install them:

im-version: 0.1.0

name: NetappUC3

description: Run Time Production Monitoring and Predictive Maintenance services

provider: Software Company

version: 1.0.19

service-format: helm

services:

- name: rtpm

package: <https://registry.nearbycomputing.com/chartrepo/smart5grid-apps/rtpm:1.0.19>

- name: pm

package: <https://registry.nearbycomputing.com/chartrepo/smart5grid-apps/pm:1.0.19>

- name: mqttbroker

package: <https://registry.nearbycomputing.com/chartrepo/smart5grid-apps/mqttbroker:1.0.12>

6.4.4. Deployment of network application in the NetAppController

The steps for the Deployment of the network application in the NetAppController are as follows:

- Selecting of the network application in MarketPlace page of Designer service of NetAppController of NearByComputing.

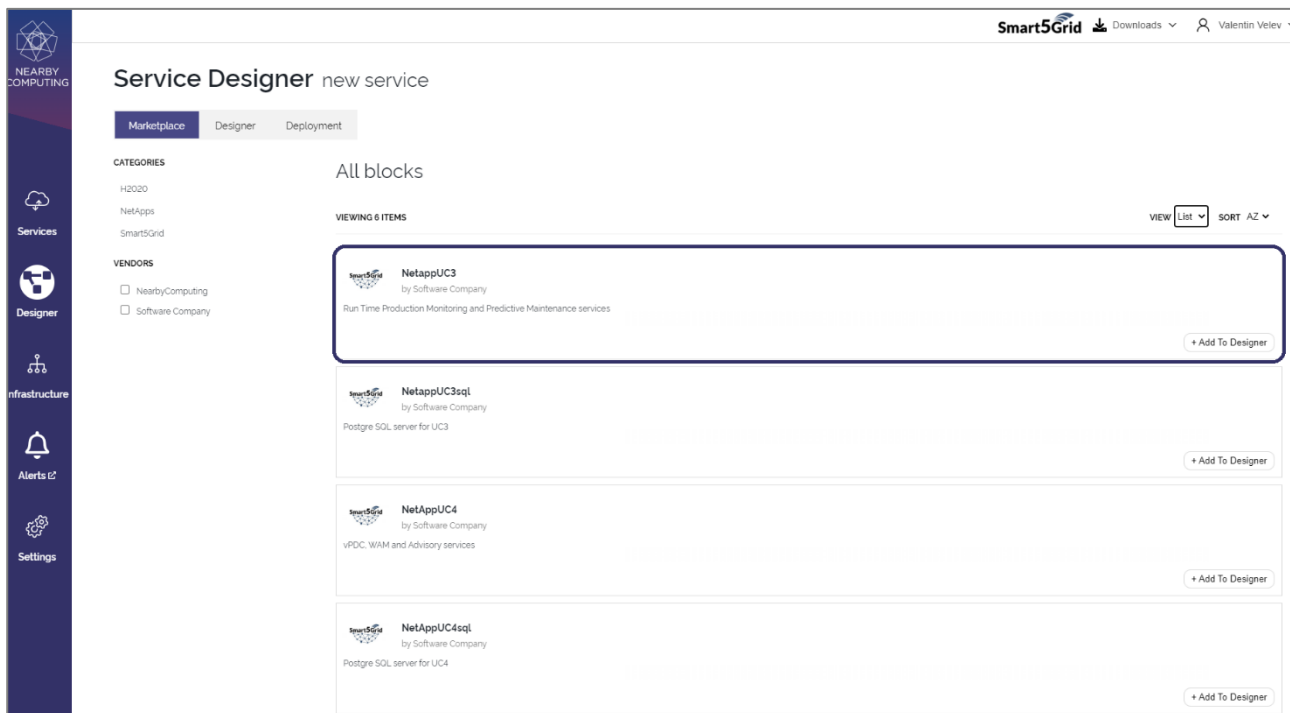


Figure 34. UC3 Network Application – Service Designer (Selecting of Network App)

- Configuring of the Network App, selecting of the site where the Network App will be deployed.

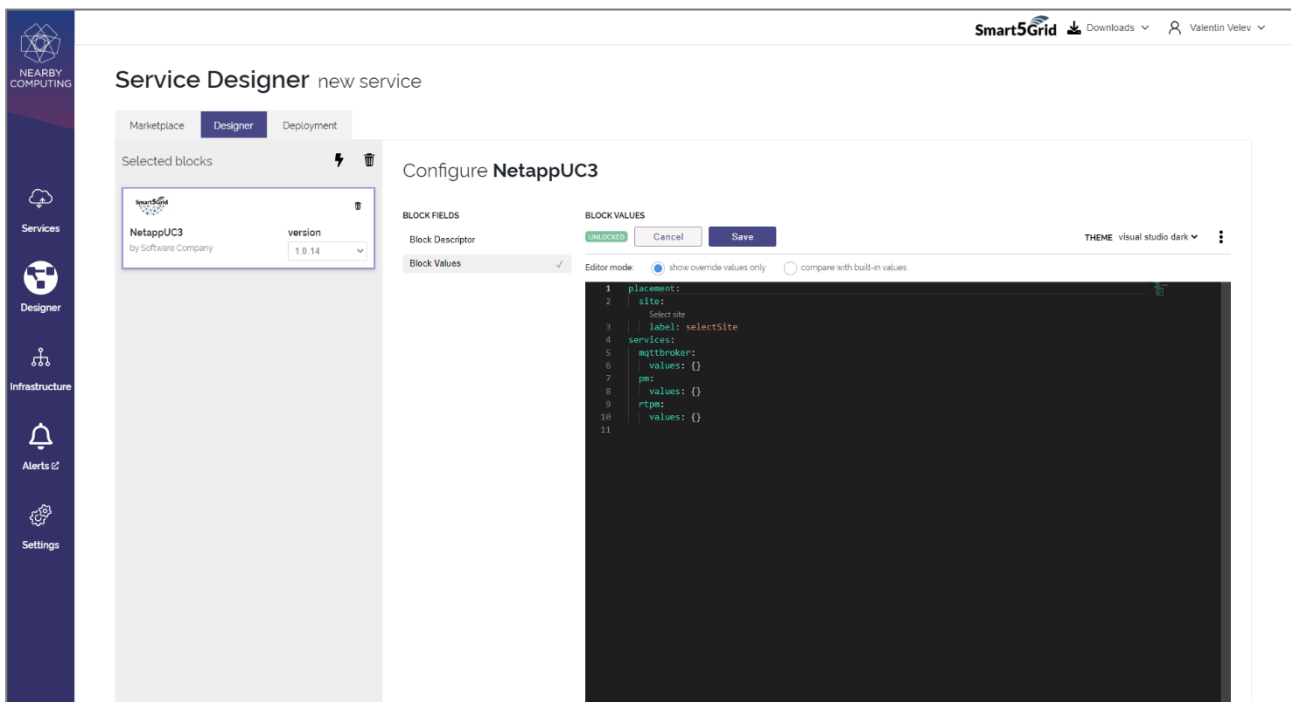


Figure 35. UC3 Network Application – Service Designer (Selecting deployment site)

- Deployed Network App appears in the list of the deployed services in “Ready” state.

NAME	STATUS	DEPLOYED	USER	ACTIONS
NetApp UC1 in Wind3 (service chain)	DEPLOYING	10/20/2022, 11:22:50 AM	ea847d76-a217-4e16-9b51-b4c34a6cf6da	...
TEST (service chain)	DEPLOYING	12/7/2022, 11:13:42 AM	ea847d76-a217-4e16-9b51-b4c34a6cf6da	...
UC3 NetApp (service chain)	READY	2/23/2023, 11:04:11 AM	9ad75838-d95d-4824-b040-a6f46cfc3551	...
NetappUC3	READY	2/23/2023, 11:04:11 AM	9ad75838-d95d-4824-b040-a6f46cfc3551	...
chart-deployment-mqttbroker-5wv25 kind: ChartRelease	READY			...
chart-deployment-pm-dv8r8 kind: ChartRelease	READY			...
chart-deployment-rtpm-4pzc kind: ChartRelease	READY			...
chart-deployment-mqttbroker kind: ChartDeployment	READY			...
chart-deployment-pm kind: ChartDeployment	READY			...
chart-deployment-rtpm kind: ChartDeployment	READY			...
UC3 Postgre SQL server (service chain)	READY	2/22/2023, 7:32:07 PM	9ad75838-d95d-4824-b040-a6f46cfc3551	...
NetappUC3sql	READY	2/22/2023, 7:32:07 PM	9ad75838-d95d-4824-b040-a6f46cfc3551	...
chart-deployment-postgresqluc3-w282x kind: ChartRelease	READY			...
chart-deployment-postgresqluc3 kind: ChartDeployment	READY			...
UC4 NetApp (service chain)	READY	2/23/2023, 11:07:56 AM	9ad75838-d95d-4824-b040-a6f46cfc3551	...

Figure 36. UC3 Network Application - Services interface

7. UC#4 Network Application(s)

The role of the network application for UC4 is to collect the measurements from the installed Phasor Measurement Units (PMUs) and provide live monitoring and analysis in the concerned wide area of the power system to the TSO. The network application of UC#4 has three services – vPDC, WAM and Advisory. The network application is installed on the virtual machine in VIVACOM cloud. The virtual machine hosts Kubernetes cluster, that is managed by the Network ApplicationController software. Each service has a web interface. The web interface is accessible on the specific EndPoint - IP address and port.

7.1. Technical Specification

vPDC VNF has two external CPs for the input from the PMU to arrive there. After the processing of the data, the output will be forwarded to the next blocks via the internal CPs. The WAM VNF has an internal CP where the input from the vPDC arrives. The Advisory VNF has an internal CP where the input from the vPDC arrives and an external one to provide the advice messages to the TSOs.

All three components have a management CP – web interface.

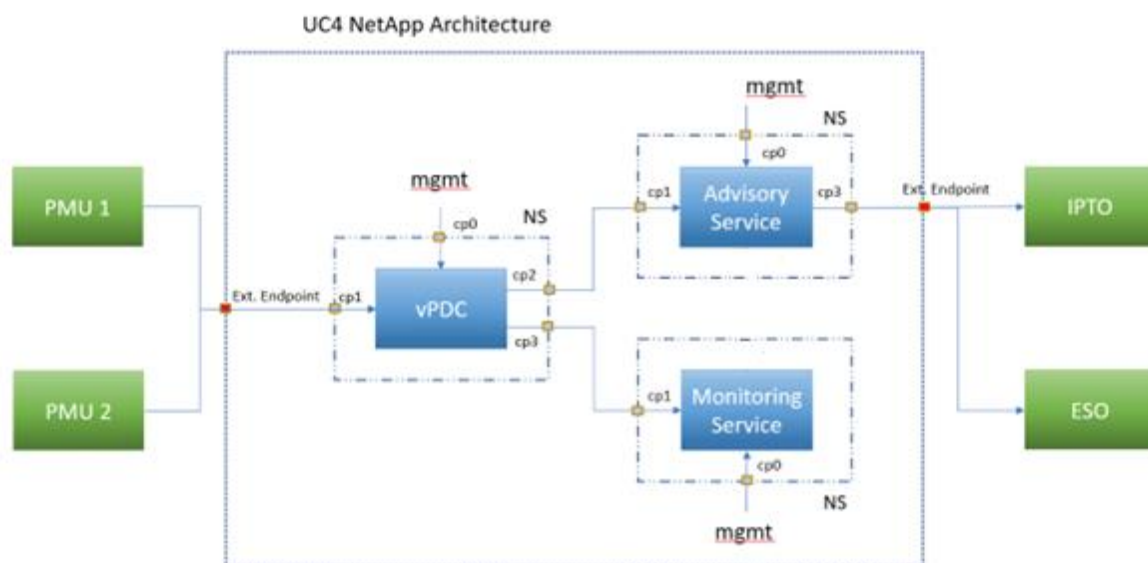


Figure 37. UC4 Network Application Architecture

7.2. Functional and Technical Requirements

7.2.1. vPDC

- **Data aggregation with time alignment to relative time:**

The PDC aligns data received from PMUs/PDCs and transmits the combined data in one or more output data streams to other PDCs or applications such as archiving, visualization, or control.

- **Data validation:**

The PDC performs basic data validation and checking of the data arriving at the PDC. This includes checking the data status flags and time quality of all PMUs and performing data integrity checks on all received data.

- **Data transfer protocol support:**

PMU data may be available in different synchrophasor data transfer protocols such as IEEE Std C37.118.2-2011, IEEE Std C37.118-2005, IEEE Std 1344-1995, IEC 61850-90-5.

- **Configuration management**

Configuration management is designed to assure availability of appropriate data for the local functions of the PDC.

- **Performance monitoring:**

PDC monitors quality of data and communication with PMUs and other PDCs.

- **Cyber security**

Cyber security will be evaluated by all PDC interfaces, while maintaining the reliability of the service.

7.2.2. WAM Service

- **Location**

A map indicating device's location in the power system.

- **Device information**

The device's name, address, model, serial number, and firmware version.

- **Grid information**

The nominal grid frequency [Hz] and the current reporting speed [fps].

- **Phase diagram**

Real-time phase diagram with voltage and current vectors.

- **Monitoring**

Real-time voltage magnitude, current magnitude and angle difference monitoring.

7.2.3. Advisory Service

- **Alarms**

Advisory service will implement different calculations and according to their results will detect abnormality in the grid.

- **Alarms monitoring**

Advisory service will show raised alarms to the user.

- **Alarms log**

Advisory service will keep log of all raised alarms.

- **Alarms messages**

Advisory service will inform the two operators via MQTT messages that corrective moves need to be done in case of an abnormality in the grid.

- **Keeping historical data in a database**

Advisory service will save the received PMUs data in a database.

- **Export data to other applications for analysis**

Advisory service will export the saved historical data to other applications like Excel and Grafana for analysis.

7.3. Network Application Functionality

7.3.1. Common functionality of the services

- **Configuration of the services:**

Each service has SQL configuration script. The script is part of the “values.yaml” file of the Helm chart of the service. The script can be edited by the administrator. The script describes input data provided to the service. This is a simple configuration script of vPDC service:

```
INSERT INTO Device(ParentID, Acronym, Name, IsConcentrator, AccessID, ProtocolID, Longitude, Latitude,
ConnectionString) VALUES( NULL, 'STERPMU_211', 'Sterpmu 211', 0, 211, 1, 23.0943, 42.0209,
'transportProtocol=File; file=S211.PmuCapture; useHighResolutionInputTimer=True');
INSERT INTO Phasor(DeviceID, Label, Type, Phase, SourceIndex) VALUES((SELECT ID FROM Device where
acronym='STERPMU_211'), 'VL1', 'V', '1', 1);
INSERT INTO Measurement (DeviceID, PointTag, SignalTypeID, PhasorSourceIndex, SignalReference,
Description) VALUES((SELECT ID FROM Device where acronym='STERPMU_211'), 'STERPMU_211-FQ', 5,
NULL, 'STERPMU_211-FQ', 'Sterpmu 211 Frequency');
INSERT INTO Measurement (DeviceID, PointTag, SignalTypeID, PhasorSourceIndex, SignalReference,
Description) VALUES((SELECT ID FROM Device where acronym='STERPMU_211'), 'STERPMU_211-DF', 6,
NULL, 'STERPMU_211-DF', 'Sterpmu 211 Frequency Delta (dF/dt)');
INSERT INTO Measurement (DeviceID, PointTag, SignalTypeID, PhasorSourceIndex, SignalReference,
Description) VALUES((SELECT ID FROM Device where acronym='STERPMU_211'), 'STERPMU_211-SF', 8,
NULL, 'STERPMU_211-SF', 'Sterpmu 211 Status Flags');
INSERT INTO Measurement (DeviceID, PointTag, SignalTypeID, PhasorSourceIndex, SignalReference,
Description) VALUES((SELECT ID FROM Device where acronym='STERPMU_211'), 'STERPMU_211-PM1', 1, 1,
'STERPMU_211-PM1', 'Sterpmu 211 VL1 Magnitude');
INSERT INTO Measurement (DeviceID, PointTag, SignalTypeID, PhasorSourceIndex, SignalReference,
Description) VALUES((SELECT ID FROM Device where acronym='STERPMU_211'), 'STERPMU_211-PA1', 2, 1,
'STERPMU_211-PA1', 'Sterpmu 211 VL1 Current Phase Angle');
INSERT INTO OutputStreamDevice( IDCode, Acronym, BpaAcronym, Name) VALUES( 3,
'STERPMU_211', '', 'Sterpmu 211');
INSERT INTO OutputStreamMeasurement (PointID, SignalReference) VALUES ((SELECT pointid FROM
measurement WHERE SignalReference='STERPMU_211-FQ'), 'STERPMU_211-FQ');
INSERT INTO OutputStreamMeasurement (PointID, SignalReference) VALUES ((SELECT pointid FROM
measurement WHERE SignalReference='STERPMU_211-DF'), 'STERPMU_211-DF');
INSERT INTO OutputStreamMeasurement (PointID, SignalReference) VALUES ((SELECT pointid FROM
measurement WHERE SignalReference='STERPMU_211-SF'), 'STERPMU_211-SF');
INSERT INTO OutputStreamMeasurement (PointID, SignalReference) VALUES ((SELECT pointid FROM
measurement WHERE SignalReference='STERPMU_211-PM1'), 'STERPMU_211-PM1');
INSERT INTO OutputStreamMeasurement (PointID, SignalReference) VALUES ((SELECT pointid FROM
measurement WHERE SignalReference='STERPMU_211-PA1'), 'STERPMU_211-PA1');
```

- **List of devices**

Each service shows a list of connected devices

ID	Acronym	Name	Connection String
1	PMULAB1	PMULAB1	transportProtocol=File; file=PMU99.PmuCapture; useHighResolutionInputTimer=True
2	PMULAB2	PMULAB2	transportProtocol=File; file=PMU201.PmuCapture; useHighResolutionInputTimer=True
3	STERPMU_211	Sterpmu 211	transportProtocol=File; file=S211.PmuCapture; useHighResolutionInputTimer=True

Figure 38. UC4 Network Application - List of devices per service interface

- **Device properties**

The device properties are shown in each service.

View Device

Local Device ID: 3

Longitude: 23.0943

Acronym: STERPMU_211

Latitude: 42.0209

Name: Sterpmu 211

Protocol: IEEE C37.118V1

Access ID: 211

Frames Per Second: 50

Connection String: transportProtocol=File; file=S211.PmuCapture; useHighResolutionInputTimer=True

Map showing location in the Balkans region.

Cancel

Figure 39. UC4 Network Application - Device properties per service interface

- **Service state**

Each service has service state menu item for monitoring of technical parameters of the service.

Category	Counter	Last	Average	Maximum	Units
Process	Process CPU Usage	2.65	2.23	7.37	Average % / CPU
Process	Process Thread Count	67.00	65.28	67.00	System Threads
Process	Process Memory Usage	301.72	253.45	301.72	Megabytes
.NET CLR LocksAndThreads	Thread Queue Size	0.00	0.11	2.00	Waiting Threads
.NET CLR LocksAndThreads	Lock Contention Rate	207.60	108.27	332.60	Attempts / sec
.NET CLR Memory	CLR Memory Usage	0.00	0.00	0.00	Megabytes
.NET CLR Memory	Large Object Heap	0.00	0.00	0.00	Megabytes
.NET CLR Exceptions	Exception Count	482.00	132.36	482.00	Total Exceptions
Network Interface	IP Outgoing (lo)	0.00	543.43	19550.82	Bytes / sec
Network Interface	IP Incoming (lo)	0.00	543.43	19550.81	Bytes / sec
Network Interface	IP Outgoing (tunl0)	0.00	0.00	0.00	Bytes / sec
Network Interface	IP Incoming (tunl0)	0.00	0.00	0.00	Bytes / sec
Network Interface	IP Outgoing (sit0)	0.00	0.00	0.00	Bytes / sec
Network Interface	IP Incoming (sit0)	0.00	0.00	0.00	Bytes / sec
Network Interface	IP Outgoing (eth0)	18644.70	26435.54	412182.90	Bytes / sec
Network Interface	IP Incoming (eth0)	4308.78	3789.95	13887.59	Bytes / sec

Figure 40. UC4 Network Application - Service state interface

7.3.2. vPDC service

vPDC receives and time-synchronizes phasor data from multiple PMUs to produce a time-aligned output data stream. The vPDC transmits the combined data to vPDCs, WAM and Advisory services. PMU data may be available in different synchrophasor data transfer protocols such as IEEE Std C37.118.2-2011, IEEE Std C37.118-2005, IEEE Std 1344-1995, IEC 61850-90-5. vPDC performs basic data validation and checking of the data arriving at the PDC. This includes checking the data status flags and time quality. The vPDC creates correctness and completeness reports each day.

Statistics provided by the vPDC for each PMU device:

Statistics for device Sterpmu 211			
ID	Description	Value	Time (UTC)
PMU571	Device statistic for Number of data quality errors reported by device during last reporting interval.	0	09:06:10.929
PMU572	Device statistic for Number of time quality errors reported by device during last reporting interval.	0	09:06:10.929
PMU573	Device statistic for Number of device errors reported by device during last reporting interval.	0	09:06:10.929
PMU574	Device statistic for Number of measurements received from device during last reporting interval.	15,500	09:06:10.929
PMU575	Device statistic for Expected number of measurements received from device during last reporting interval.	0	09:06:10.929
PMU576	Device statistic for Number of measurements received while device was reporting errors during last reporting interval.	0	09:06:10.929
PMU577	Device statistic for Number of defined measurements from device during last reporting interval.	0	09:06:10.929
PMU578	Device statistic for Device time deviation from average for all input devices in seconds	0.000 s	09:06:10.929
IS1571	InputStream statistic for Total number of frames received from input stream during last reporting interval.	500	09:06:10.929
IS1572	InputStream statistic for Timestamp of last received data frame from input stream.	53120.000	09:06:10.929
IS1573	InputStream statistic for Number of frames that were not received from input stream during last reporting interval.	0	09:06:10.929
IS1574	InputStream statistic for Number of CRC errors reported from input stream during last reporting interval.	0	09:06:10.929
IS1575	InputStream statistic for Number of out-of-order frames received from input stream during last reporting interval.	0	09:06:10.929
IS1576	InputStream statistic for Minimum latency from input stream, in milliseconds, during last reporting interval.	28.000 ms	09:06:10.929
IS1577	InputStream statistic for Maximum latency from input stream, in milliseconds, during last reporting interval.	67.000 ms	09:06:10.929
IS1578	InputStream statistic for Boolean value representing if input stream was continually connected during last reporting interval.	True	09:06:10.929
IS1579	InputStream statistic for Boolean value representing if input stream has received (or has cached) a configuration frame during last reporting interval.	False	09:06:10.929
IS15710	InputStream statistic for Number of configuration changes reported by input stream during last reporting interval.	0	09:06:10.929
IS15711	InputStream statistic for Number of data frames received from input stream during last reporting interval.	500	09:06:10.929
IS15712	InputStream statistic for Number of configuration frames received from input stream during last reporting interval.	0	09:06:10.929
IS15713	InputStream statistic for Number of header frames received from input stream during last reporting interval.	0	09:06:10.929
IS15714	InputStream statistic for Average latency, in milliseconds, for data received from input stream during last reporting interval.	31.000 ms	09:06:10.929
IS15715	InputStream statistic for Frame rate as defined by input stream during last reporting interval.	50 frames / second	09:06:10.929
IS15716	InputStream statistic for Latest actual mean frame rate for data received from input stream during last reporting interval.	49.999 frames / second	09:06:10.929
IS15717	InputStream statistic for Latest actual mean Mbps data rate for data received from input stream during last reporting interval.	0.056 Mbps	09:06:10.929
IS15718	InputStream statistic for Number of data units that were not received at least once from input stream during last reporting interval.	0	09:06:10.929
IS15719	InputStream statistic for Number of bytes received from the input source during last reporting interval.	70,000	09:06:10.929
IS15720	InputStream statistic for Number of processed measurements reported by the input stream during the lifetime of the input stream.	5,565,952	09:06:10.929
IS15721	InputStream statistic for Number of bytes received from the input source during the lifetime of the input stream.	24,350,794	09:06:10.929
IS15722	InputStream statistic for The minimum number of measurements received per second during the last reporting interval.	1,600	09:06:10.929
IS15723	InputStream statistic for The maximum number of measurements received per second during the last reporting interval.	1,632	09:06:10.929
IS15724	InputStream statistic for The average number of measurements received per second during the last reporting interval.	1,603	09:06:10.929
IS15725	InputStream statistic for Minimum latency from input stream, in milliseconds, during the lifetime of the input stream.	16 ms	09:06:10.929
IS15726	InputStream statistic for Maximum latency from input stream, in milliseconds, during the lifetime of the input stream.	112 ms	09:06:10.929
IS15727	InputStream statistic for Average latency, in milliseconds, for data received from input stream during the lifetime of the input stream.	31 ms	09:06:10.929
IS15728	InputStream statistic for Total number of seconds input stream has been running.	3,488.823 s	09:06:10.929

Figure 41. UC4 Network Application - vPDC Statistics for each PMU device interface

7.3.3. WAM service

WAM service visualizes the PMUs data.

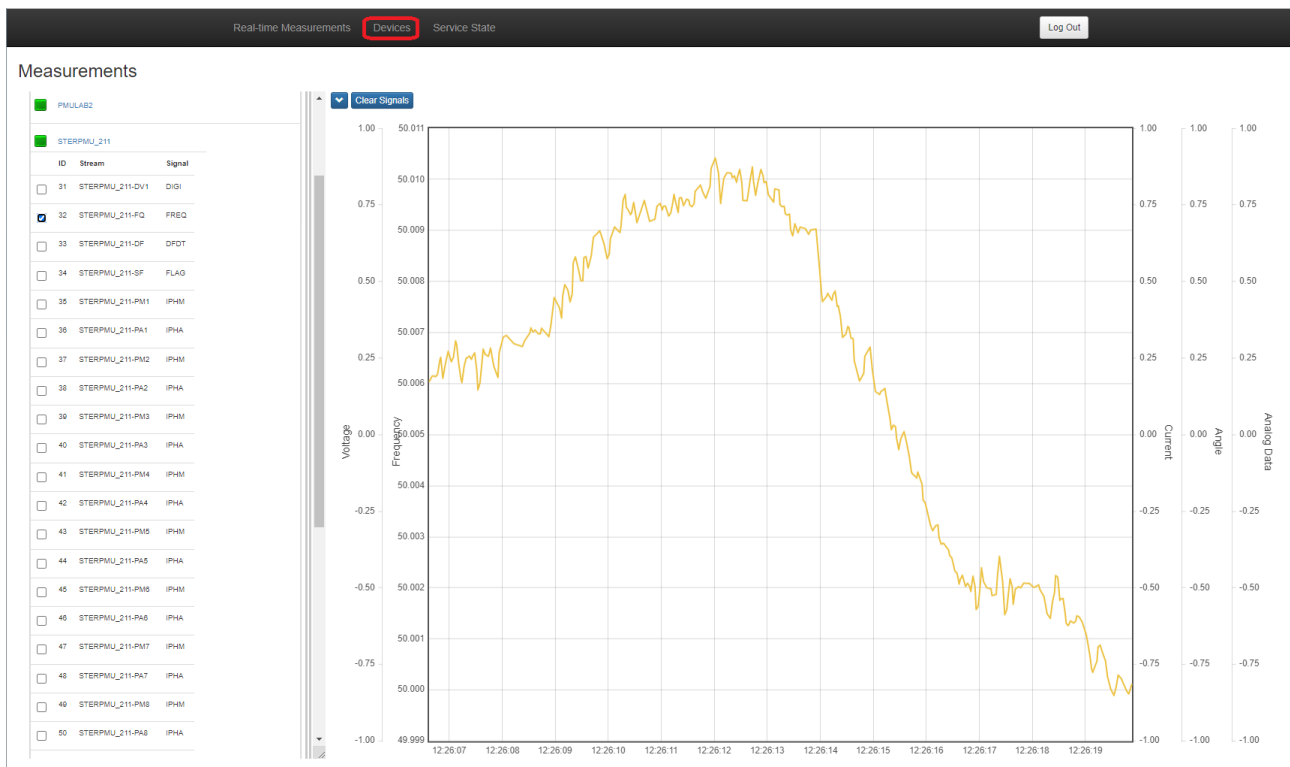


Figure 42. UC4 Network Application - WAM service, PMUs data visualization interface

7.3.4. Advisory service

Advisory service provides alarms for real-time operation to both TSOs and ex-post analysis in case of severe event occurrence.

- Exploring measurements data from the database

Select signals

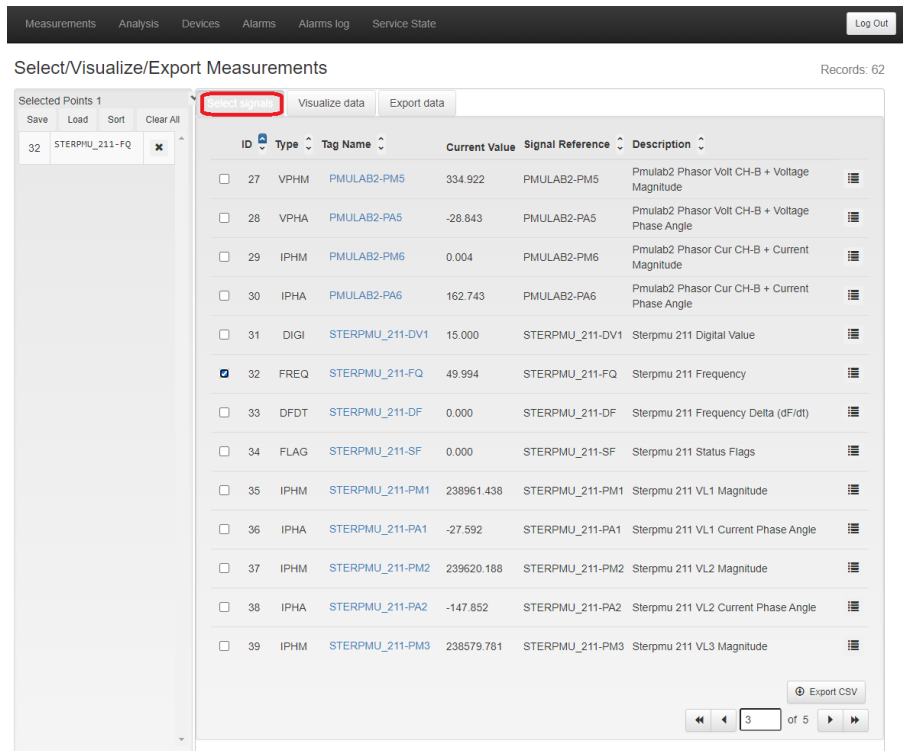


Figure 43. UC4 Network Application - Select signal interface

Visualize data for the selected signal(s)

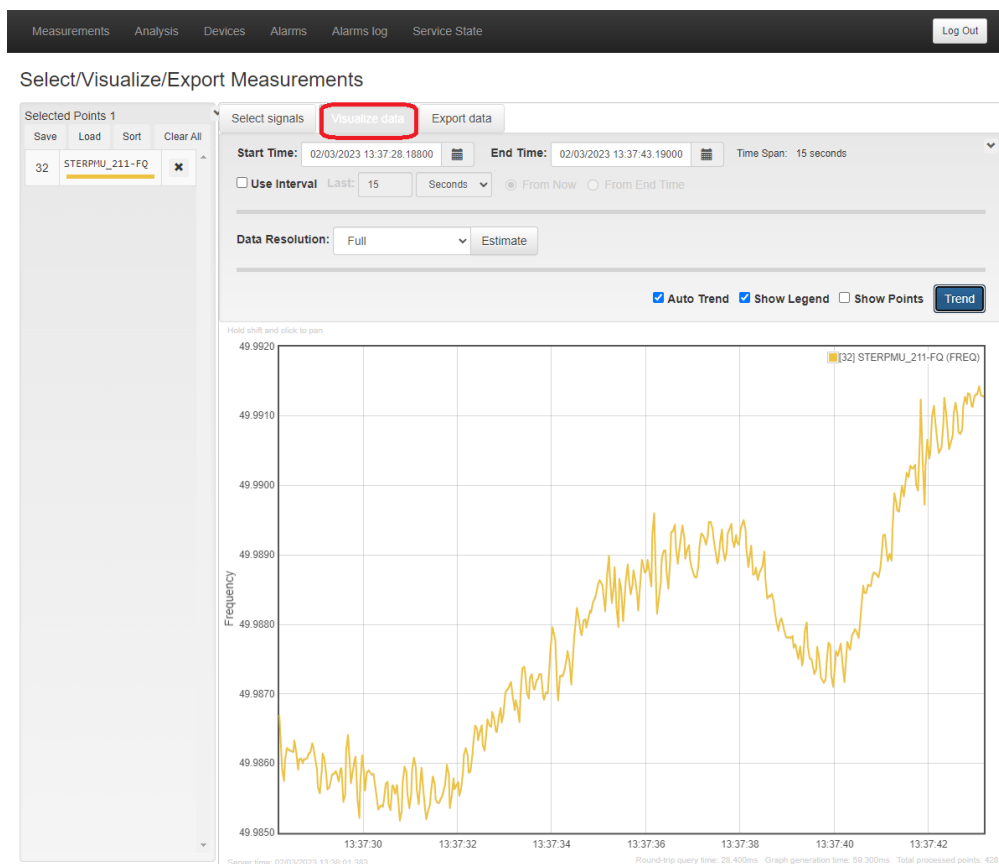


Figure 44. UC4 Network Application - Visualization of selected signal(s) interface

Export data for the selected signal(s)

Figure 45. UC4 Network Application - Export data for selected signal(s) interface

- **Alarms**

Alarms conditions are defined in the configuration script. The raised alarms are shown in Alarms page.

ID	Tag Name	Severity	State	Time Raised	Value at Time Raised	Description	Operation	Set Point	Delay	Hysteresis
5	STERPMU_FREQUENCY_GREATER	850	1	02/03/2023 14:02:45	50.0109939575195	STERPMU_211: Frequency greater than 50.01	12	50.01	0.25	0

Figure 46. UC4 Network Application - Alarms interface

- **Alarms log**

All alarm actions are saved in alarms log. The log is visible in "Alarms log" page.

Measurements

Analysis

Devices

Alarms

Alarms log

Service State

Log Out

Alarms log

Records: 326

ID	Signal	Previous State	New State	Time Stamp	Value
294	STERPMU_211-FQ	STERPMU_FREQUENCY_LESS	Normal	2023/03/02, 02:03:58.380	49.99010467529297
282	PMULAB1-FQ	PMULAB1_FREQUENCY_LESS	Normal	2023/03/02, 02:03:58.240	49.999427795410156
281	PMULAB1-FQ	Normal	PMULAB1_FREQUENCY_LESS	2023/03/02, 02:03:58.180	49.85760498046875
293	STERPMU_211-FQ	Normal	STERPMU_FREQUENCY_LESS	2023/03/02, 02:03:58.080	49.9899786376953
292	STERPMU_211-FQ	STERPMU_FREQUENCY_LESS	Normal	2023/03/02, 02:03:58.040	49.990055084228516
291	STERPMU_211-FQ	Normal	STERPMU_FREQUENCY_LESS	2023/03/02, 02:03:57.580	49.98958969116211
290	STERPMU_211-FQ	STERPMU_FREQUENCY_LESS	Normal	2023/03/02, 02:03:57.540	49.990013122558594
279	PMULAB1-FQ	PMULAB1_FREQUENCY_GREATER	Normal	2023/03/02, 02:03:56.780	49.873043060302734
278	PMULAB1-FQ	Normal	PMULAB1_FREQUENCY_GREATER	2023/03/02, 02:03:56.740	50.133506774902344
289	STERPMU_211-FQ	Normal	STERPMU_FREQUENCY_LESS	2023/03/02, 02:03:56.320	49.98887252807617
288	STERPMU_211-FQ	STERPMU_FREQUENCY_GREATER	Normal	2023/03/02, 02:03:56.080	49.98911666870117
298	STERPMU_211-FQ	STERPMU_FREQUENCY_LESS	Normal	2023/03/02, 02:03:55.920	49.990203857421875
297	STERPMU_211-FQ	Normal	STERPMU_FREQUENCY_LESS	2023/03/02, 02:03:55.820	49.98978805541992
296	STERPMU_211-FQ	STERPMU_FREQUENCY_LESS	Normal	2023/03/02, 02:03:55.780	49.99001693725586
277	STERPMU_211-FQ	Normal	STERPMU_FREQUENCY_LESS	2023/03/02, 02:03:54.320	49.98957061767578
323	STERPMU_211-FQ	Normal	STERPMU_FREQUENCY_LESS	2023/03/02, 02:03:53.520	49.98918151855469
314	PMULAB1-FQ	PMULAB1_FREQUENCY_LESS	Normal	2023/03/02, 02:03:51.180	50.088401794433594
313	PMULAB1-FQ	Normal	PMULAB1_FREQUENCY_LESS	2023/03/02, 02:03:50.920	49.897708892822266
276	STERPMU_211-FQ	STERPMU_FREQUENCY_LESS	Normal	2023/03/02, 02:03:50.720	49.99004364013672
275	STERPMU_211-FQ	Normal	STERPMU_FREQUENCY_LESS	2023/03/02, 02:03:50.520	49.989742279052734
274	STERPMU_211-FQ	STERPMU_FREQUENCY_LESS	Normal	2023/03/02, 02:03:50.480	49.99000930786133
273	STERPMU_211-FQ	Normal	STERPMU_FREQUENCY_LESS	2023/03/02, 02:03:50.320	49.98978805541992

Export CSV

1 of 15

Figure 47. UC4 Network Application - Alarms log

7.4. Integration & Deployment

Each service is put in Docker image. For each service is created a Helm chart. Helm chart is a collection of files that describe a related set of Kubernetes resources necessary for the service. Helm is the package manager for Kubernetes used to install the service in Kubernetes cluster using the correspondent Helm chart. The links to the Helm charts of the services are included in the descriptor file of the network application.

The operating system Linux (Ubuntu 22.10) is used in Docker images. The services are developed with Visual Studio 2022 tool using C# programming language and .NET Framework 4.8.1. The web interface is developed using JavaScript. The database used for the configuration of the services is PostgreSQL 14.

7.4.1. Docker images of the services

The Docker Images of vPDC, WAM and Advisory services are almost the same. The Docker Image of Advisory service is most complicated, it is created by the following Docker file:

```
FROM UBUNTU:22.10

WORKDIR /APP

ARG DEBIAN_FRONTEND=NONINTERACTIVE

RUN APT-GET UPDATE
```



```

# INSTALL MONO
RUN APT -Y INSTALL -Y MONO-XSP4

# INSTALL POSTGRESQL-14
RUN APT -Y INSTALL POSTGRESQL-14

# INSTALL GRAFANA
RUN APT-GET INSTALL -Y GNUPG2 CURL SOFTWARE-PROPERTIES-COMMON
RUN APT-GET UPDATE
RUN CURL HTTPS://PACKAGES.GRAFANA.COM/GPG.KEY | APT-KEY ADD -
RUN ADD-APT-REPOSITORY "DEB HTTPS://PACKAGES.GRAFANA.COM/OSS/DEB STABLE MAIN"
RUN APT-GET UPDATE
RUN APT-GET -Y INSTALL GRAFANA

USER POSTGRES

RUN ECHO "HOST ALL 0.0.0.0/0 TRUST" >> /ETC/POSTGRESQL/14/MAIN/PG_HBA.CONF &&\
ECHO "LISTEN_ADDRESSES=*" >> /ETC/POSTGRESQL/14/MAIN/POSTGRESQL.CONF &&\
/ETC/INIT.D/POSTGRESQL START &&\
CREATEUSER -S -I -D -R -L -W ROOT &&\
PSQL --COMMAND "ALTER USER POSTGRES WITH PASSWORD '#S5GVALENTIN';" &&\
/ETC/INIT.D/POSTGRESQL STOP

#EXPOSE PORT 80 - WEB INTERFACE OF ADVISORY
EXPOSE 80

#COPY FILES
USER ROOT
COPY ./POSTGRESQL/CREATEADVISORY.SQL /APP/POSTGRESQL/
COPY ./POSTGRESQL/INITIALDATASET.SQL /APP/POSTGRESQL/
COPY ./START.SH /APP/
COPY ./CREATEDB.SH /APP/

# CONFIGURE ADVISORY DATABASE IN POSTGRESQL
RUN /APP/CREATEDB.SH
COPY ./BUILD/ /OPT/ADVISORY/
COPY ./CONFIG/ /OPT/ADVISORY/
COPY ./GRAFANALINUX/CONF/GRAFANA.INI /ETC/GRAFANA/GRAFANA.INI

```

```
COPY ./GRAFANALINUX/PLUGINS /VAR/LIB/GRAFANA/PLUGINS
```

```
COPY ./GRAFANALINUX/DATA/GRAFANA.DB /VAR/LIB/GRAFANA
```

```
CMD ["/APP/START.SH"]
```

7.4.2. Helm charts of the services

The Helm charts of vPDC, WAM and Advisory services are almost the same. The creation of WAM helm chart is shown as an example.

Creating empty wam helm chart with the following command:

```
helm create wam
```

The command will create wam folder with several files:

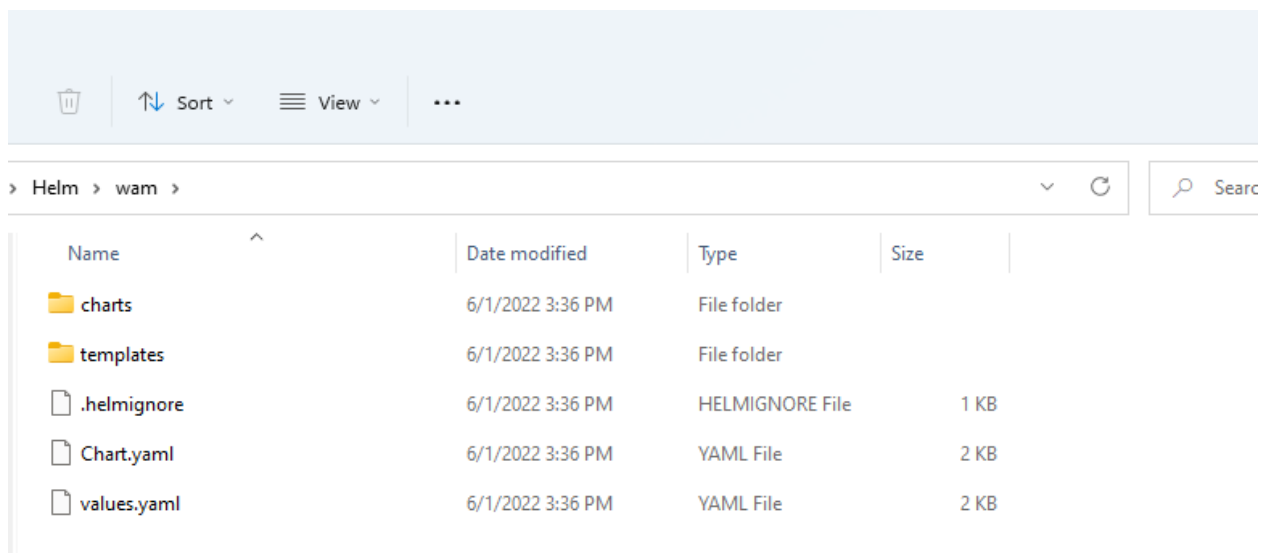


Figure 48. UC4 Network Application - Folder and files output of helm command

In the Helm chart we need to define the following items:

- The image that should be installed.
- The service Kubernetes component with the external port that will be used to access the application.
- The startupProbe used for detecting the successful starting.
- The name of the component.
- Edit "values.yaml" file:

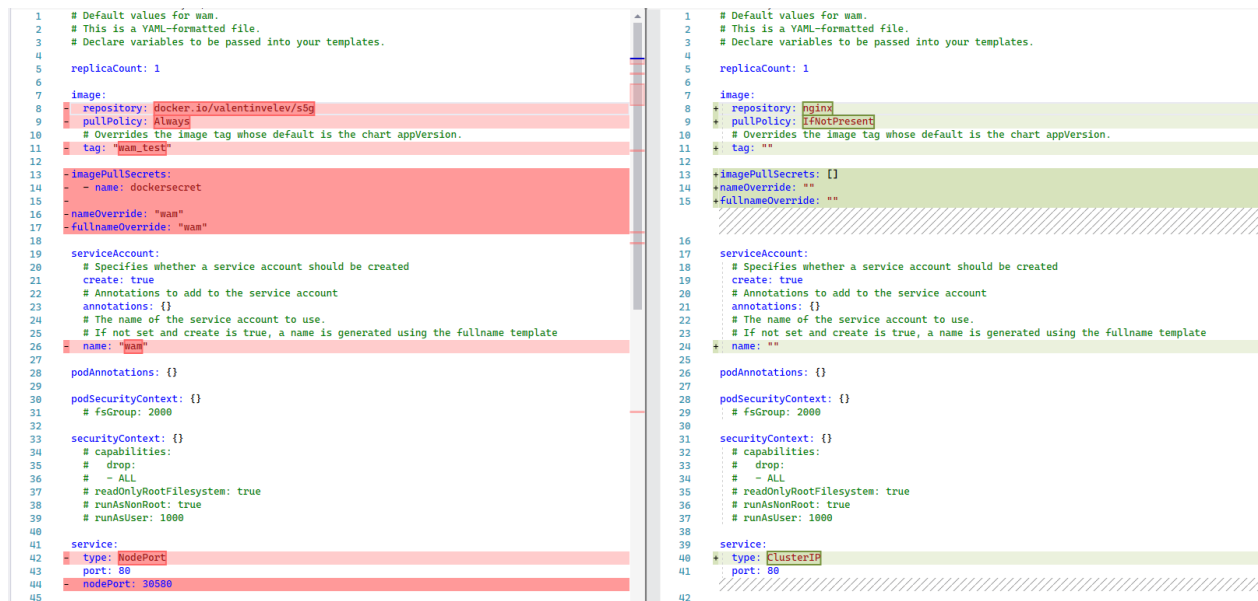


Figure 49. UC4 Network Application – YAML(values) file edit

In image section is described the location of the image that will be installed.

repository: docker.io/valentinvelev/s5g

tag: "wam_test"

pullPolicy: Always (this change is optional)

imagePullSecrets:

- name: dockersecret (this is the name of the secret object that contain the name and password for the access to the repository)

Filling the name of the chart

nameOverride: "wam"

fullnameOverride: "wam"

serviceAccount:

name: "wam"

In the service section we define the parameters used in service template:

type: NodePort

port: 80

nodePort: 30580

There are three different types of Kubernetes service, we use the type NodePort. The nodePort is the port used by the external applications for accessing the WAM service.

After that, the "templates\deployment.yaml" file is edited and the following section must be added:

Figure 50. UC4 Network Application – YAML(deployment) file edit

startupProbe:

httpGet:

path: /

port: http

failureThreshold: 6

periodSeconds: 30

Define startupProbe with the path "/" on the "http" port with the number of failures "6" and period "30" seconds. Kubernetes will try to load the main web page of the web service, when loading is successful, the Kubernetes will know that WAM web service is started.

Then, the "templates\service.yaml" file should be edited and the following properties added:

Figure 51. UC4 Network Application – YAML(service) file edit

targetPort: {{ .Values.service.port }}

```
nodePort: {{ .Values.service.nodePort }}
```

The targetPort and nodePort will get the values defined in “values.yaml” file.

7.4.3. Network application descriptor

The network application descriptor file is necessary for the deployment. It describes the services used by the network application and the package files that install them:

```
netapp:
  im-version: 0.1.0
  name: NetAppUC4
  description: vPDC, WAM and Advisory services
  provider: Software Company
  version: 1.0.19
  service-format: helm
  services:
    - name: mqttbrokeradvisory
      package: https://registry.nearbycomputing.com/chartrepo/smart5grid-
apps/mqttbrokeradvisory:1.0.19
    - name: vpdc
      package: https://registry.nearbycomputing.com/chartrepo/smart5grid-
apps/vpdc:1.0.19
    - name: wam
      package: https://registry.nearbycomputing.com/chartrepo/smart5grid-
apps/wam:1.0.19
    - name: advisory      package: https://registry.nearbycomputing.com/chartrepo/smart5grid-
apps/advisory:1.0.19https://registry.nearbycomputing.com/chartrepo/smart5grid-
apps/advisory:1.0.19
```

7.4.4. Deployment of network application in the NetAppController

Steps:

Selecting of the network application in MarketPlace page of Designer service of Network ApplicationController of NearByComputing.

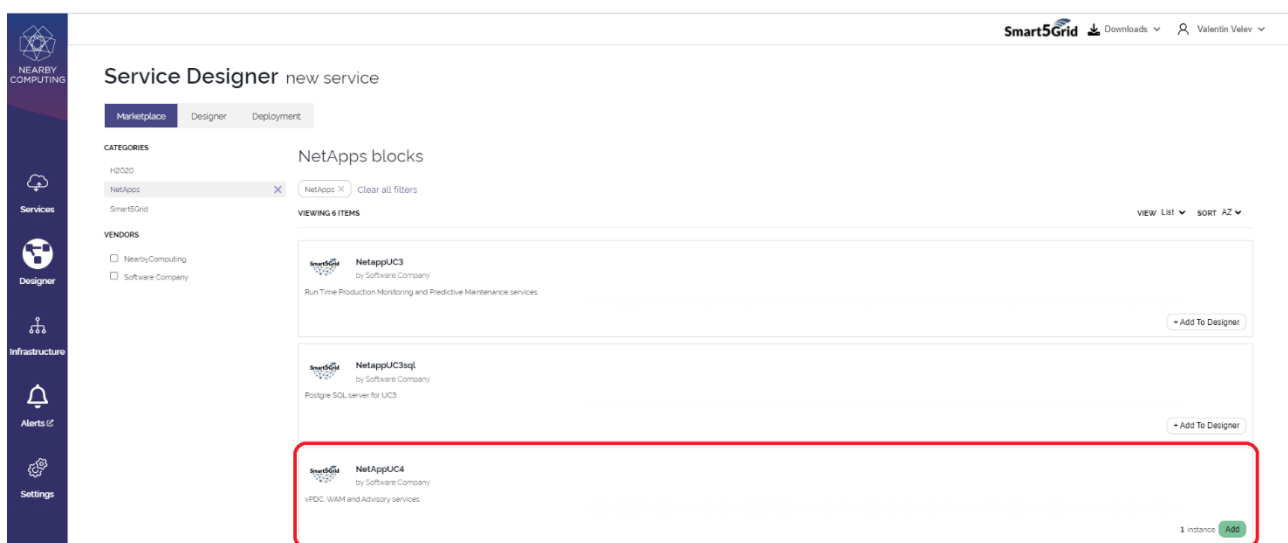


Figure 52. UC4 Network Application – Service Designer (Selecting of Network App)

Configuring of the Network Application, selecting of the site where the network application will be deployed.

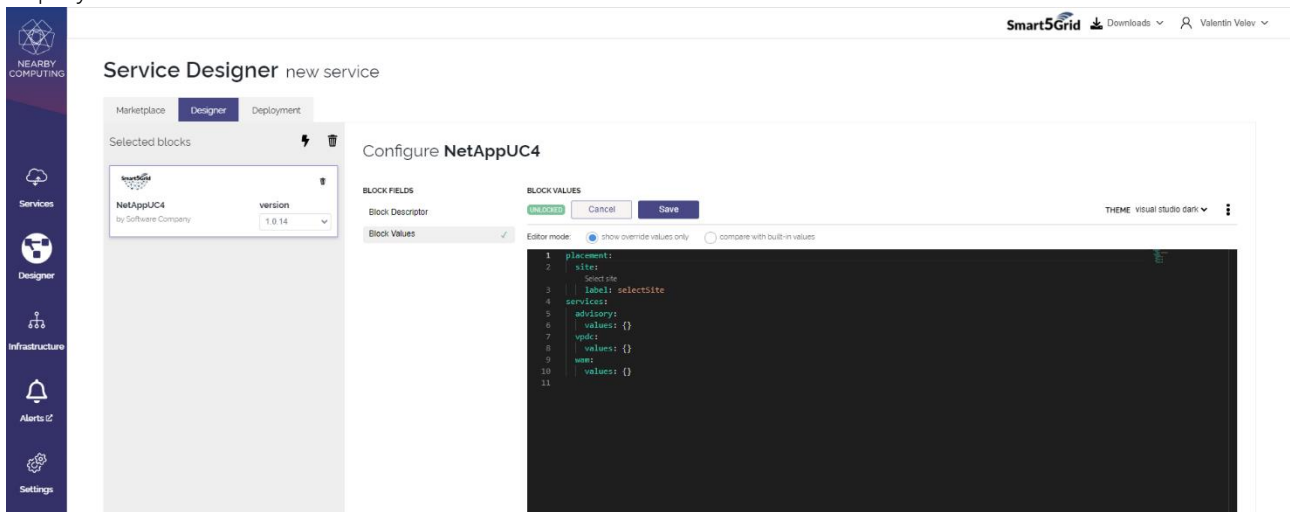


Figure 53. UC4 Network Application – Service Designer (Selecting deployment site)

Deployed Network Application appears in the list of the deployed services in “Ready” state.

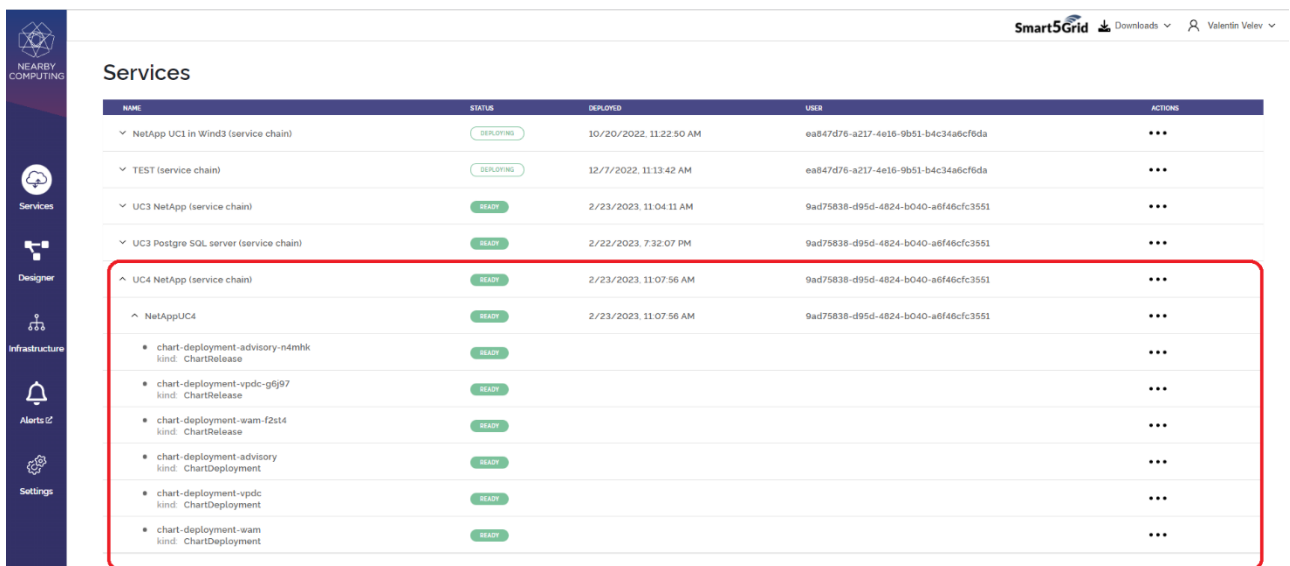


Figure 54. UC4 Network Application - Services interface

8. Conclusions

In this deliverable, the details on the development and deployment of Network Applications were provided with specific information and steps. The Smart5Grid open 5G platform was explained accompanied with a figure describing its benefits. The Network Applications' vision and definition so that external readers can understand the concept of the Network Apps was provided along with the whole simplified lifecycle from designing the application to operation and termination.

After all of that information, the detailed guide on how to develop a Network application to leverage the Smart5Grid Open 5G platform was provided with detailed steps and instructions from the beginning to the end of the development. All UCs that developed specific Network Applications followed that guide and provided their different developer side view, to assist each other and external developers that will read and use this guide.

As this deliverable is public, entities and individuals who seek to know more about Network Apps can use the content provided in this document to gain further insights into some of the developments that were made in Smart5Grid and particularly in WP4.

9. References

- [1] Smart5Grid D2.2, "Overall Architecture Design, Technical Specifications and Technology Enablers," [Online]. Available: https://smart5grid.eu/wp-content/uploads/2022/11/Smart5Grid_WP2__D2.2_PU_Overall-Architecture-Design-Technical-Specifications-and-Technology-Enablers_V2.0.pdf.
- [2] Smart5Grid D3.3, "Open NetApp repository," [Online]. Available: https://smart5grid.eu/wp-content/uploads/2023/03/Smart5Grid_WP3__D3.3_PU_Open-NetApp-repository_V1.0.pdf.
- [3] Smart5Grid D3.1, "Interim report for the development of the 5G network facility," [Online]. Available: https://smart5grid.eu/wp-content/uploads/2022/11/Smart5Grid_WP3__D3.1_PU_Interim-Report-for-the-development-of-the-5G-network-facilities_V1.0.pdf.
- [4] "Build your Python image," [Online]. Available: <https://docs.docker.com/language/python/build-images/#build-an-image>.
- [5] "Creating a cluster with kubeadm | Kubernetes," [Online]. Available: <https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/>.
- [6] "Using Minikube to Create a Cluster | Kubernetes," [Online]. Available: <https://kubernetes.io/docs/tutorials/kubernetes-basics/create-cluster/cluster-intro/>.
- [7] "Deployments | Kubernetes," [Online]. Available: <https://kubernetes.io/docs/concepts/workloads/controllers/deployment/>.
- [8] "Service | Kubernetes," [Online]. Available: <https://kubernetes.io/docs/concepts/services-networking/service/>.
- [9] "Helm | Getting started," [Online]. Available: https://helm.sh/docs/chart_template_guide/getting_started/.
- [10] "Helm | Installing Helm," [Online]. Available: <https://helm.sh/docs/intro/install/>.
- [11] "Helm | Quickstart Guide," [Online]. Available: <https://helm.sh/docs/intro/quickstart/>.
- [12] "Helm | Charts," [Online]. Available: <https://helm.sh/docs/topics/charts/>.
- [13] "OSM," [Online]. Available: <https://osm.etsi.org/>.
- [14] "Experience with NFV architecture, interfaces, and Information Models," [Online]. Available: https://osm.etsi.org/images/OSM_White_Paper_Experience_implementing_NFV_specs_final.pdf.
- [15] "ANNEX 3: OSM Information Model — Open Source MANO documentation," [Online]. Available: <https://osm.etsi.org/docs/user-guide/latest/11-osm-im.html>.

- [16] "Welcome to Open Source MANO's VNF Onboarding guide! — Open Source MANO 6.0 documentation," [Online]. Available: <https://osm.etsi.org/docs/vnf-onboarding-guidelines/>.
- [17] "ANNEX 3: OSM Information Model — Open Source MANO documentation," [Online]. Available: <https://osm.etsi.org/docs/user-guide/latest/11-osm-im.html>.
- [18] Smart5Grid D3.4, "Smart5Grid platform integration and HIL testing activities," [Online]. Available: https://smart5grid.eu/wp-content/uploads/2023/03/Smart5Grid_WP3__D3.4_PU_Smart5Grid-platform-integration-and-HIL-testing-activities_V1.0.pdf.