Demonstration of **5G** solutions for
SMART energy **GRID**s of the future

Deliverable 3.3

Open Network App Repository

Version 1.0 - Date 31/12/2022

This project has received funding from the European Union's *Horizon 2020 research and innovation programme* under grant agreement n° 101016912

D3.3 – Open Network App Repository

Document Information

Programme	Horizon 2020 Framework Programme – Information and Communication Technologies
Project acronym	Smart5Grid
Grant agreement number	101016912
Number of the Deliverable	D3.3
WP/Task related	WP3
Type (distribution level)	PU Public
Date of delivery	31-12-2022
Status and Version	Version 1.0
Number of pages	85 pages
Document Responsible	Yanos Angelopoulos AXON
Author(s)	Angelos Antonopoulos – NBC Antonello Corsi – ENG Daniela Patrício – UW George Kontopoulos – 8BELLS Lenos Hadjidemetriou – UCY Nicola Cadenelli – NBC Nicola Sergnese – ENG Paula Encinar – ATOS Pedro Teixeira – UW

	Sonia Castro – ATOS
	Yanos Angelopoulos – AXON
Reviewers	Giampaolo Fiorentino – ENG
	Irina Ciornei – UCY

Revision History

Version	Date	Author/Reviewer	Notes
0.1	22/08/2022	Yanos Angelopoulos – AXON	Initial table of content
0.2	20/11/2022	George Kontopoulos – 8BELLS Nicola Cadenelli – NBC Yanos Angelopoulos – AXON	First draft
0.3	28/11/2022	Angelos Antonopoulos – NBC Daniela Patrício – UW George Kontopoulos – 8BELLS Lenos Hadjidemetriou – UCY Nicola Cadenelli – NBC Nicola Sergnese – ENG Paula Encinar – ATOS Pedro Teixeira – UW Sonia Castro – ATOS Yanos Angelopoulos – AXON	Most partner contributions submitted.
0.4	05/11/2022	Antonello Corsi – ENG Paula Encinar – ATOS Yanos Angelopoulos – AXON	Complete draft with all contributions. Some content was restructured and text was formatted.
0.5	19/12/2022	Giampaolo Fiorentino – ENG Irina Ciornei – UCY	Revised Version
1.0	31/12/2022	Yanos Angelopoulos – AXON	Submitted Version

Executive summary

Smart5Grid builds an open-access platform that accommodates experimenters to access, develop in a collaborative manner, share and test Network Apps on simulated 5G and energy vertical production environments. One of the prominent components of the Smart5Grid platform is the Open Service Repository (OSR). This deliverable describes in detail the design, specifications, and implementation of the Smart5Grid platform OSR and the User Interface. The OSR is the open Network App repository that stores Network Apps and all their subcomponents Network Services (NSs), Virtual Network Functions (VNFs), Virtual Deployment Units (VDUs), and images.

This deliverable reports the results produced from the work done in Task 3.2 "Open Service Repository" with the addition of the development of the Platform User Interface. By the time of the submission of this deliverable the OSR service (T3.2) is considered complete. The report includes an overview of related software services with similar goals and their functionality, existing in the market or being developed by ICT-41 projects. The OSR strengths and functionality are presented in detail. In order to better understand the main object whose information is stored in the OSR, there is a dedicated section on the Network App information model (IM). While the OSR can support the evolution of the information included in the Network App IM, it follows its main structure and defines a compatible data model. For the complete understanding of the OSR's internal and external communications all the methods per component interface provided by the OSR are explained. Finally, we explain all technological decisions, implementation, deployment, testing, and CI/CD methods performed in the process of creating the OSR and the Platform User Interface.

Table of contents

List of figures

List of tables

1. Introduction

1.1. Scope of the document

The scope of this document is to report the work done by the involved partners in Task 3.2 "*Open Service Repository*" (OSR). The expected outcomes are the delivery of a tested and fully operational prototype repository accessible from the 5G and energy infrastructure and publicly available to external stakeholders. In addition to that, we add detailed information on the Platform User Interface which was developed in accordance with the OSR's Access Point Interfaces (APIs) to provide easier access to the users of the Smart5Grid platform.

1.2. Document Structure

Initially, this document analyses the main concepts and terminology of an open service repository as well as the current status and progress related to software repositories in Open-Source software development communities and various implementations from relevant 5G European-funded projects. The next part of the document analyses the pertinent requirements of the OSR and the way the OSR conforms and adds value to the Smart5Grid overall architecture. It describes the chosen data model to represent the required information, and the system use cases covered. This section is essential for the architecture definition and the implementation of the software. The design is highly modular and expandable and can be easily enhanced with further improvements (i.e. health monitoring, high availability, etc). The refined details of the Network App operations and the functional use cases are also provided. In the next section, we document the purpose and functionality of the OSR through all of its subcomponents. We describe both the internal and external interfaces provided. The document continues with a detailed description of the Smart5Grid Platform User Interface (UI) detailing the architecture, the use cases, and the views provided to the users. In the following sections, we assess the open-source technologies we considered for the implementation of each subcomponent and we explain the decisions we took. Moreover, we describe the configuration details of the developed and deployed software components, and the integration between subcomponents and external interfaces. The implementation details include the laboratory environment used, at the time of writing this document, the workflow pipelines followed in our development process, and the testing mechanisms we used to validate and ensure the functional integrity of our solution. The document concludes with the "User Guide" on how to efficiently use the OSR and the Smart5Grid Platform UI to perform actions offered by the platform.

1.2.1. Notations, abbreviations and acronyms

Item	Description
5G	5th Generation (of mobile telecommunication networks)
5G PPP	5G Infrastructure Public Private

	Partnership
A&A	Authentication and Authorization
API	Application Programming Interface
AWS	Amazon Web Services
CI/CD	Continuous Integration/Continuous Deployment
CIAM	Customer identity and access management
CIDR	Classless Inter-Domain Routing
CLI	Command Line Interface
CNCF	Cloud Native Computing Foundation
CNI	Container Network Interface
DevOps	Development and Operations
dnsmasq	DNS masquerade
ELK	Elasticsearch Logstash Kibana
ETSI	European Telecommunications Standards Institute
GitOps	Git Operations
HDD	Hard Disk Drive
HTTPS	Hypertext Transfer Protocol Secure
IAM	Identity and Access Management
ICT	Information and Communication Technologies
IM	Information Model
JSON	JavaScript Object Notation
JWT	JSON Web Token
K3S	Kubernetes (alternative)
K8S	Kubernetes
KVM	Kernel-based Virtual Machine
MAAS	Metal As A Service
MAC	Media Access Control
NAC	Network App Controller
Network App	Network Application
NF	Network Function
NFS	Network File System
NS	Network Service
NSD	Network Service Descriptor
OCI	Open Container Initiative
OIDC	OpenID Connect
OSR	Open Service Repository
OSR A&A	OSR Authentication and Authorization
OSR CV	OSR Code Versioning

OSR EL	OSR Event Loggings
OSR NCAT	OSR Network App Catalogue
PVC	Persistent Volume Claim
PXE	Pre-boot Execution Environment
PYPL	Popularity of Programming Language Index
RAID	Redundant Array of Independent Disks
RBAC	Role Based Access Control
RP	Relying Party
SAP	Service Access Point
SDK	Software Development Kit
SLO	Service Level Objective
SME	Small and Medium-sized Enterprises
SQL	Structured Query Language
SSH	Secure Shell
UC	Use Case
UI	User Interface
UID	Unique Identifier
VDU	Virtual Deployment Unit
VNF	Virtual Network Function
VNFFGD	VNF Forwarding Graph Descriptor
VXLAN	Virtual Extensible LAN
YANG	Yet Another Next Generation

Table 1: Acronyms list

1.3. Target Audience

This document provides a high-level description of the OSR software developed in Smart5Grid, as well as detailed information of the implementation. In this context, the document is targeted to researchers and developers that work in similar open repository concepts aiming to facilitate development and collaboration in a niche industry like the Energy and 5G Telecommunications sector. While the Network Apps developed in the project's context are focused on the aforementioned sectors, the OSR could be useful to an extended set of verticals. Such audience could work on developing applications and services that will use the OSR's open APIs to leverage or extend OSR's functionality.

2. Concepts and Related Work

In this section we provide some background knowledge regarding Service Repository for Containerized applications, and we motivate why we need a Network App Repository.

2.1.Service Repository Concepts

While Smart5Grid defines Services as containerized software services that get deployed in Kubernetes clusters, the Cambridge Dictionary defines Repository as "*a place where things are stored and can be found*". Therefore, in this context, a Service Repository is a place where containerized software services are stored and can be found. Intuitively, a Network App Repository is a place where Network Apps are stored and can be found.

Repositories, whether of code or services, have become crucial to share knowledge over the Internet to the entire globe. In fact, nowadays there are many different repositories for Docker images and other artifacts. However, since Smart5Grid defined its own concept of Network App, it also needed to provide a Network App repository to store and share such artifacts. In more details, Smart5Grid's Network App relies on two artifacts kind: Docker images and Helm Charts

For Containers, to standardize the adoption of just one distribution method, the Open Container Initiative (OCI) Distribution Specification [1] was defined and adopted by many projects. Ever since, this OCI distribution method started to be adopted to distribute different kind of artifacts, for instance, Helm charts. This push for standardization was beneficial and let some registries, likes Harbor [2], store different kind of artifacts at the same time[1]. This means that IT departments from all over the world, can store and share their services using these registries, either public or private. And this is exactly what the OSR aims to do in the context of Smart5Grid.

2.2. Related Products

Over the years many Docker repositories have emerged. Docker Hub [2] and RedHat Quay [4] container registry These projects are among the world's largest repositories of container images. Their repositories allow you to share container images with a team, customers, or the community. Similarly, Helm charts repositories like Artifacthub.io [5] and VMware's Bitnami [6], just to name a few, have also emerged.

Finally, the previously mentioned Harbor "*is an open-source registry that secures artifacts with policies and role-based access control, ensures images are scanned and free from vulnerabilities, and signs images as trusted. Harbor, a CNCF Graduated project, delivers compliance, performance, and interoperability to help you consistently and securely manage artifacts across cloud native compute platforms like Kubernetes*

[1] While a repository is a collection of similar artifacts with the same name but different tags, a registry is a service that is storing some artifacts. Usually, a registry allows to create many repositories or even create projects to provide a role base access control (RBAC) rules to control the who and how has access to each project.

and Docker." [2] Harbor allows its users to self-host dedicated registry to store Docker containers and Helm charts. We will see in the next sections how Smart5Grid OSR comprises of a dedicated Harbor instance.

2.2.1. Network App repositories ICT-41 Projects

In this section, we are going to illustrate the main different ICT-41 projects that have managed to store and retrieve the Network Apps. We will not go into depth on the technologies used but rather we will do a high-level overview to illustrate how each project operated and made available these functionalities.

All ICT-41 projects cover different domains/verticals but at the same time share many common aspects on how to store Network Applications and how they are exploited by third parties experimenters in providing new applications/services.

5GASP [7] has put in place a Network App Marketplace that provides a public registry of SMEs and their registered products: reusable Network Apps, Network Function (NF) and Network Service (NS) links to open-source repositories, and useful documentation that an SME needs to know also to put in place a certification process. This portal is complemented by a Network App community that supports third parties in development.

Another interesting approach is the one followed by the project EVOLVED-5G [8]. There the Network App repository is composed of two separate artifacts. The first one is GitHub on which EVOLVED-5G has created a template to generate the Network App file structure as part of the Software Development Kit (SDK) toolchain. In this repository, the developers can share their Network App code. The second repository, namely the Open Repository is used to store and manage all the Network App images (binaries) and is an extension to other repositories similar to source code repositories, i.e., GitHub. For the validation and certification stages, the Repository will be connected to a third component of the workspace, allowing CI/CD life cycle.

5G IANA [9] has an architecture component named the Network Applications Toolkit, whose goal is to simplify the design and onboarding of new automotive services from third-party experimenters. Inside this component, there is a catalogue of available Network Applications that can be used to compose advanced vertical services.

The VITAL-5G Open Online Repository [10] is one of the core components of the VITAL-5G Platform, providing the catalogue of the Network Apps developed for the project and giving the opportunity to third-party experimenters and developers to download and select them for experimentation, as well as to onboard their own Network Apps to build new vertical services. The main artifacts are 3 different catalogues that allow respectively to query and onboard Network App packages, Vertical Service Blueprints, and Descriptors, as well as Experiment Blueprints and Descriptors, together with their associated VNF packages and NFV Network Service Descriptors.

In 5G-MEDIA HUB [11] there is a Network Apps Repository application that has the main purpose to enable Network Apps to be onboarded on the underlying 5G testbeds. This allows the users to design, validate and deploy their Network Apps based on the set of available VNF present in the service catalogue. The Catalogue Management and Service Ordering features from the underlying 5G testbeds allow the users to design Network Apps in a "drag and drop" modality where the constituent VNFs can be dropped onto the canvas and connecting the VNFs to generate the forwarding graphs.

3. OSR Specifications

3.1. Relation with Overall Smart5Grid requirements

The OSR is one of the primary front-facing services of the Smart5Grid platform. Together with the Platform User Interface, they consist of the main endpoint for the users to access the Smart5Grid offerings. The OSR provides a repository to store Network Apps and all the sub-components that compose a Network App. As of the time of writing this document, the definition of the Network App follows a dual format. One ETSI compliant and another simplified and more flexible for the purposes of the project. The OSR, also, consists of a toolset to facilitate the Network App development process. It offers a simplified overlay platform with drivers to the currently most preferred tools in software development tools. It automates multiple steps of the development process under the context of the Network App as a project to include all the required information to deploy a complete and complex service comprised of several different components and the definition of the interactions between them. Throughout the integration with V&V platform OSR's users can test their Network Apps on secure execution environments simulating production conditions in the energy vertical sector.

3.1.1. Network App Descriptor Template

Smart5Grid proposes a Network App that enables developers to build vertical applications with the necessary components to deliver a service with the access performance requirements needed to meet the demanding performance constraints that may be requested by applications in the vertical and to take advantage of the features and performance offered by 5G networks.

In order to define the Network App and each of its features, Smart5Grid defines an Information Model (IM) of the Network App trying to capture all the necessary details. To create the Network App's IM, YANG [12] has been used as the data modelling language. This IM defines the structure of the Network App and lists each of the fields that will compose it, as well as the type of data that each field must follow.

The IM facilitates the developer the creation of the Network App descriptor, showing all the fields that compose it as well as which of them are mandatory and which are optional, thus allowing the creation of a vertical application that covers all the needs. The first version of the Network App IM was introduced in D2.2 [13] which has been updated and improved according to the needs of the individual UCs as you can see below::

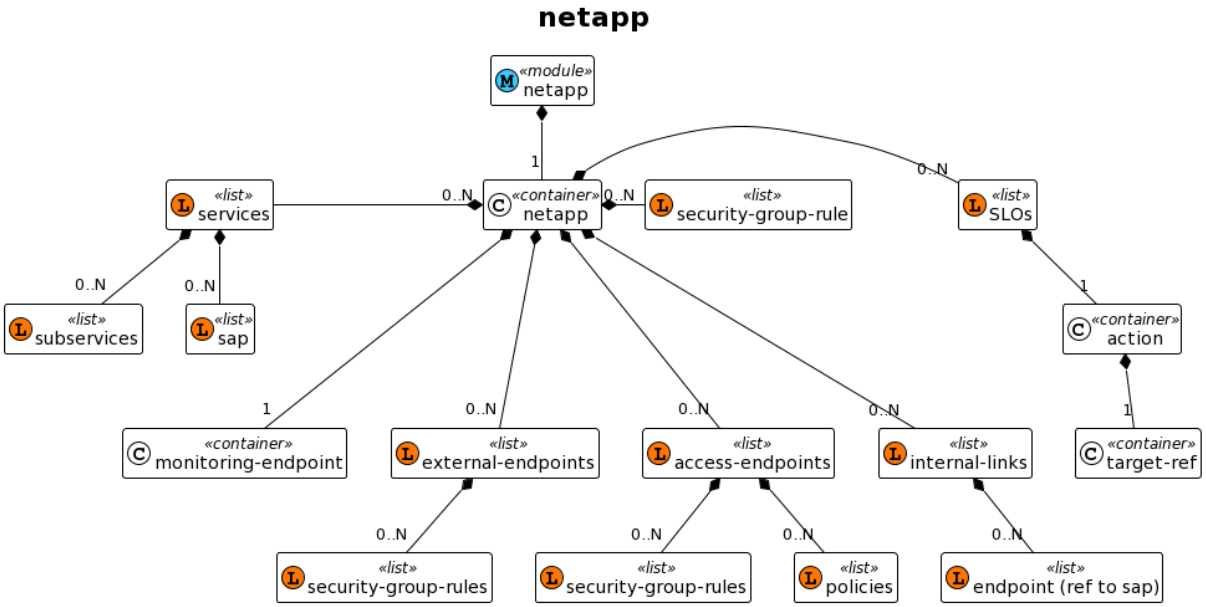

Figure 1 Network App Information Model

As can be seen in Figure 1, the Network App is composed of interconnected and grouped services and sub-services. These services expose the service access points (SAPs) that will be referred to in each of the Network App's endpoints. These endpoints can be of three types depending on their intended use: i) a Monitoring Endpoint, which is the endpoint in charge of communicating with the Network App Controller to monitor the metrics defined in the SLO field, ii) an External Endpoint, which makes a Network App service available externally to interface with a dashboard or APIs, iii) an Access Endpoint, which defines the access policies with the appropriate performance requirements to access each function. In addition, there are other fields that are part of the Network App descriptor, such as SLOs with requirements on the set of metrics exposed by the services, which can be used by the Network App driver to trigger scaling or migration actions, as well as Security-Group-Rule to define certain traffic rules or policies expected for that specific type of traffic and communication should be provided.

Each of the items defined in the final IM are described in the following table:

Item	Type	Content
Network App	Object	Main Network App container object
Network App.IM_version	String	Information Model version of the Network App
Network App.Provider	String	Provider of the Network App
Network App.Name	String	Network App Name
Network App.Version	String	Network App Version
Network App.Description	String	Description of the functionality of the Network App
Network App.Service_Format	String	Format of the Service item. Allowed values are: • OSM: in case of using of packages of services and sub-services using the OSM

Item	Type	Content
		IM • HELM: in case the NearbyComputing Network App driver is used, thus referencing the NBC IM
Network App.Services	List	List of Services type object. The service format is specified in the Network App-Service-Format field.
Network App.Monitoring_endpoint	Object	Monitoring endpoint type object.
Network App.External_endpoints	List	List of External endpoints type object.
Network App.Access_endpoints	List	List of Access endpoints type object. Only supported for OSM IM
Network App.Internal_Link		List of Internal Link type Object. Only supported for OSM service format
Network App.SLOs	List	List of SLO type objects that apply to the Network App
Network App.Security_group_rules	List	List of Security-group-rule type objects applicable to the Access Endpoint. Only supported for OSM IM
Services	Object	Service Object
Services.name	String	Service name
Services.package	String	Service package name. Two types of packages are supported: • OSM service format: NS package as tar.gz • HELM service format: Helm-chart package in tgz format
Services.subservices	List	List of subservices type object to reference from the Services.
Services.values	String	Values of the chart exposed at the Network App level - Values.yaml will be overwritten
Services.sap	List	List of SAPs type Objects
Subservices	Object	Subservices object.
Subservices.name	String	Subservice name. Allows two types of values: • OSM service format: VNF reference name • HELM service format: reference to the Docker image where it is stored in

Item	Type	Content
		the repository
Subservices.package	String	Subservice package name. Two types of packages: • OSM service format: VNF package as tar.gz • HELM service format: Docker image as tar.gz
Sap	Object	Service Access point Object
Sap.name	String	Name of the SAP/SAPs of which the service is composed
Monitoring_endpoint	Object	Monitoring Endpoint container object
Monitoring_endpoint.service_ref	String	Reference to the Service where the Network App monitoring service is reachable
Monitoring_endpoint.sap_ref	String	Reference to the SAP where the Network App monitoring service is reachable.
Monitoring_endpoint.url	String	URL where the SLIs exposed by the Network App are available.
External_endpoints	Object	External Endpoints object
External_endpoints.name	String	Name of the External Endpoint container object
External_endpoints.service_ref	String	Reference to the Service where the Network App service reachable through this external endpoint is available
External_endpoints.sap_ref	String	Reference to the SAP where the Network App service reachable through this external endpoint is available
External_endpoints.security_group_rules	List	List of Security_group_rules type objects applicable to the External Endpoint. Only supported for OSM service format
Security_group_rules	Object	Security group rules object.
Security_group_rules.id_ref	String	Id reference of the security group rule defined in the Security Group Rule list of type Object
Access_endpoints	Object	Access Endpoint object. Only supported for OSM service format
Access_endpoints.name	String	Name of the Access Endpoint container object

Item	Type	Content
Access_endpoints.service_ref	String	Reference to the Service where the Network App service reachable through this access endpoint is available
Access_endpoints.sap_ref	String	Reference to the SAP where the Network App service reachable through this access endpoint is available
Access_endpoints.security_group_rules	List	List of Security_group_rule type objects applicable to the Access Endpoint
Access_endpoints.policies	List	List of Policies type objects applicable to the Access Endpoint
Security_group_rules	Object	Security group rules container object
Security_group_rules.id_ref	String	Id reference of the security group rule defined in the Security Group Rule list of type Object
Policies	Object	Policies container object
Policies.key	String	Name of the specified Access Policy. Allowed values: • Latency: Maximum Latency (ms) • Jitter: Maximum Jitter (ms) • Bandwith_UE: Minimum Bandwidth per UE (Kbps) • Bandwith_aggr: Minimum Bandwidth aggregate (kbps) • Availability: Minimum Availability (number of nines) • Reliability: Minimum Reliability (number of nines) • Density: Minimum Device Density (UE/km2)
Policies.value	Integer	Value of the specified Access Policy
Internal_links	Object	Access Link Object. Only supported for OSM service format
Internal_link.name	String	Name of specific internal link
Internal_link.endpoints	List	List of endpoint type objects applicable to the Internal link
Endpoints	Object	Endpoint Object
Endpoints.service_ref	String	Reference to the Service where the Network App service reachable through this internal link is available
Endpoints.sap _ref	String	Reference to the SAP where the Network App service reachable

Item	Type	Content
		through this Internal link is available
SLOs	Object	Service Level Objective object
SLO.name	String	Name of the SLO
SLO.expression	String	Time series data aggregation expression. Either the field metric or expression must exist in an SLO object.
SLO.metric	String	Reference to the metric when presented as already aggregated. Either the field metric or expression exist in an SLO object.
SLO.threshold	Integer	If the value of the SLO is GREATER Than or LOWER than (see "threshold_type" field) this value, it constitutes a violation of the SLO.
SLO.threshold_type	String	Type of the threshold. Allowed values are GT (GREATER THAN) or LT (LOWER THAN)
SLO.Action	Object	Action type object describing the action to be taken if SLO is violated.
SLO.granularity	Integer	Every Number of Seconds that this SLO will be checked. A value of "0" means best effort.
SLO.cycles	Integer	Number of cycles of granularity time that the thresholds must be crossed in order to consider a violation of SLO.
Action	Object	Action container object
Action.target_ref	Object	Target reference type object describing the reference of Subservice and Service to perform the action.
Action.action_step	String	Action to be executed every time the SLO is violated. Allowed values are: • TRIGGER_SCALE_UP • TRIGGER_SCALE_IN • TRIGGER_MIGRATION
Target_ref	Object	Target reference container object
Target_ref.target_service_ref	String	Reference to the Service
Target_ref.target_subservice _ref	String	Reference to the subservice. Supported for OSM service format. In the case of HELM service format it is optional.
Security_group_rule	Object	Security Group Rule container object

Item	Type	Content
		in the format of ETSI Standard descriptor defined in SOL006. Only supported for OSM service format
Security_group_rule.id	Integer	Id of the Security Group Rule
Security_group_rule. description	String	Description of the Security Group Rule
Security_group_rule. direction	String	Direction of the Security Group Rule. Allowed values: • Ingress • Egress
Security_group_rule.ether_type	String	Type of the Security Group Rule. Allowed values: • IPV4 • IPV6
Security_group_rule.protocol	String	Protocol of the Security Group Rule. Allowed values: • TCP • UDP • Any
Security_group_rule.port_range_min	Integer	Minimum port number of the range applicable to the Security Group Rule.
Security_group_rule.port_range_max	Integer	Maximum port number of the range applicable to the Security Group Rule.

Table 2 Smart5Grid Network App Information Model

3.2. OSR Data Model

This Section describes the data model of the data elements that are utilized in the OSR.

User

A User represents any user that has access to the OSR.

Group

A Group describes a group of users. One user can be part of many groups and one group can have many users.

User Role

A User Role defines an authority level on a given subject or a set of subjects.

Permissions

Permissions are a rule (or restrictions) to perform actions such as view, create, update, delete to objects for a specific user or a group of users.

Network App

A Network App is a high-level entity that provides an abstraction on a combination of lower-level entities (Network Services and VNFs) in order to provide a service. A Network App can have multiple Network App descriptors, each of which represent a different version of the Network App.

Network App Descriptor

A Network App descriptor is the entity that contains the actual information that describe in detail how each sub-component is connected and interacts with each other. It important to note that while the OSR's Network App descriptor stores all the information of the Network App descriptor that will be applied to the Network App Controller, the OSR Catalogue can be agnostic to the detailed information and it provides flexibility on the formats of the descriptors that can be stored. However, the OSR expects certain fields to exist and uses them to better represent the information through the OSR API.

Network App Test

A Network App Test contains the information returned by the V&V Platform.

Network Service

A Network Service represents collection of lower-level components such as VNFs and VDUs that are linked and configured in a way that they provide a network functionality of joined cause. It is linked to a code repository in the Code Versioning Service to store and keep track of changes performed to the various version of its descriptors.

Network Service Descriptor

A Network Service Descriptor contains the information that determine exactly how the Network Service is comprised in terms of required VNFs and their associated VNFFGD (VNF Forwarding Graph Descriptor). As it is mentioned in the Network App descriptor, the NS descriptor also provides flexibility to the structure of the descriptors data.

VNF

Virtual Network Functions (VNFs) are virtualized services, formerly carried out by proprietary, dedicated hardware technology. Common VNFs include virtualized routers, firewalls, WAN optimization, and network address translation (NAT) services. In the context of the OSR the VNF is a generic entity being a subcomponent of the Network Service. Similarly, to the Network Service that has a link to a code repository in the Code Versioning Service.

VNF Descriptor

The VNF Descriptor contains the connection points and the virtual links of the referenced VDUs that comprise the VNF. Again, the OSR is agnostic to the format of the descriptor allowing maximum flexibility.

VDU

Virtual Deployment Unit is a basic part of VNF. In the case of the Smart5Grid project it is the container that hosts a network function. VDUs are mapped to both the Code Versioning Service hosting the software code in a linked code repository and the Image Registry hosting the packaged software images in a linked image repository.

4. Architectural Components and Interfaces

The main component of the OSR that provides external interfaces is the OSR Network App Catalogue (OSR NCAT). In order to provide the necessary functionality OSR internal components interact with each other. We define the internal interfaces as they are depicted in Figure 2 and Figure 3.

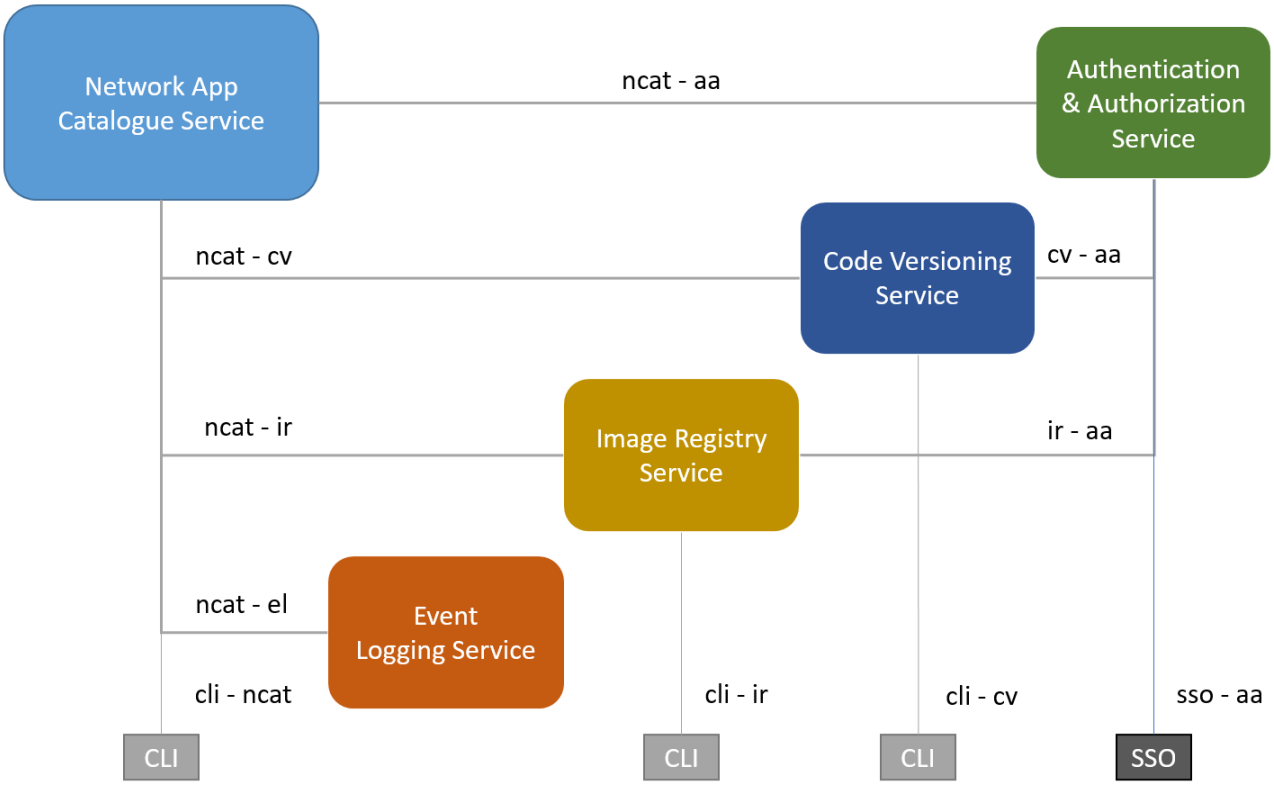

Figure 2: OSR internal and provided interfaces

4.1.OSR Authentication and Authorization Service

sso – aa interface

The Authentication and Authorization Service provides the "sso – aa" interface which implement the authentication and authorization to the users or systems on all of the OSR's components leveraging the OpenID Connect protocol. It includes user registration, user authentication and authorization. The authentication takes place by providing the user credentials, then the A&A service returns a token which validates the authentication of the user and provides authorization to perform specific actions on the OSR.

Method Name	Description	Protocol	Provides Interface to
Get Access Token	Obtain SSO OpenID Connect Access Token for a user	HTTPS	CLI, UI, Network App Catalogue, Image Registry
Create User	Method that allows User Registration	HTTPS	UI

	using the User Interface, or directly the A&A Service.		

Table 3: sso - aa interface methods

ncat – aa interface

This interface provides all the methods required by the OSR NCAT to manage in the OSR A&A Service in order to provide access to users when the OSR NCAT resources are being created, modified or deleted.

Method Name	Description	Protocol
Get User(s)	Return a list of the Users registered in the Smart5Grid project, or single user if an id is specified	HTTPS
Create/Update User	Create a User based on provided user details, if the User exists, the details will be updated	HTTPS
Get Group(s)	Return a list of the Groups, or single group if an id is specified	HTTPS
Create/Update Group	Creates a Group based on provided user details, if the Group exists, the details will be updated	HTTPS
Delete Group	Delete an existing Group	HTTPS
Add User to Group	Add an existing User to a Group	HTTPS
Remove User from Group	Remove a User from a Group	HTTPS

Table 4: ncat – aa interface methods

4.2. OSR Network App Catalogue

cli – ncat interface

This interface is the main interface provided by the OSR to the clients (users or systems). It provides the management of the OSR NCAT's components (Network Apps, NSs, VNFs) and initiates most of the internal component interactions.

Method Name	Description	Protocol
Show User Detailed View	Get a specific User's detailed fields.	HTTPS
List User Repository Keys	List a specific User's public ssh keys.	HTTPS
Create User Repository Key	Add User's public ssh key, needed for underlying Code Repository Service actions.	HTTPS
Delete User Repository Key	Delete User's public ssh key	HTTPS
List Network Apps	Get a list of all visible Network Apps across the OSR for the authenticated user.	HTTPS
Create a Network App	Create a new Network App. Available only for users who can create Network Apps.	HTTPS
Update a Network App	Update an existing Network App's details.	HTTPS
Delete a Network App	Delete a new Network App. Available only for users who can delete the specified Network App.	HTTPS
Fork Network App	Create a new Network App being the exact duplicate of the selected Network App. The new Network App will be owned by	HTTPS

	the initiator user.	
Sync Network App from Code Versioning Service	Synchronize the content of the code existing in the related repository in the Code Versioning Service to the data presented by the OSR Catalogue Service.	HTTPS
Upload Network App Descriptor	Upload the Network App descriptor files archived and compressed in "tar.gz" format. Available only for users who can alter the content of the specified Network App.	HTTPS
Download Network App Descriptor	Download the Network App descriptor archive file, compressed in "tar.gz" format.	HTTPS
Perform V&V test on a Network App	Perform a test via the V&V Platform on a specific version of a Network App. Available only for users who can alter Network App content.	HTTPS
Get V&V test results	Retrieve the test results from V&V Platform on a requested test.	HTTPS
List Network Services	Get a list of all visible Network Services for the authenticated user.	HTTPS
Show Network Service Detailed View	Get a specific Network Service's detailed fields.	HTTPS
Create a Network Service	Create a new Network Service. Available only for users who can create Network Services.	HTTPS
Update a Network Service	Create a new Network Service.	HTTPS
Delete a Network Service	Delete a new Network Service. Available only for users who can delete the specified Network Service.	HTTPS
Fork Network Service	Create a new Network Service being the exact duplicate of the selected Network Service under a specified existing Network App. The Network App must be owned by the initiator user.	HTTPS
List VNFs	Get a list of all visible VNF s for the authenticated user.	HTTPS
Show VNF Detailed View	Get a specific VNF's detailed fields.	HTTPS
Create a VNF	Create a new VNF. Available only for users who can create VNFs.	HTTPS
Update a VNF	Update an existing VNFs details.	HTTPS
Delete a VNF	Delete an existing VNF. Available only for users who can delete the specified VNF.	HTTPS
Fork VNF	Create a new VNF being the exact duplicate of the selected VNF under a specified existing Network App. The Network App must be owned by the initiator user.	HTTPS
List VDUs	Get a list of all visible VDUs s for the authenticated user.	HTTPS
Show VDU Detailed View	Get a specific VDU's detailed fields.	HTTPS
Create a VDU	Create a new VDU. Available only for users who can create VDUs.	HTTPS
Update a VDU	Update an existing VDU's details.	HTTPS
Delete a VDU	Delete a new VDU. Available only for users who can delete the specified VDU.	HTTPS
Get Event Logs	Generate a list of logs that match the provided filters (time period, action performed, object ID etc.)	HTTPS

Table 5: cli – ncat interface methods

4.2.1. OSR Code Versioning Service

ncat – cv interface

The OSR NCAT communicates with the OSR Code Versioning (OSR CV) service via the "ncat – cv" interface. This interface is used to perform all the interactions for the management of the Code Versioning Service resources that map to OSR NCAT entities. For example, a Network App maps to a repository in the OSR CV service, where the Network App Descriptor files can be stored. Interactions with the mapped objects include CRUD operations, and syncing information from the Code Versioning repositories to the OSR NCAT instances.

Method Name	Description	Protocol
List Users	Get a list of all the User instances of the Code Versioning Service.	HTTPS
Show User Details	Show User's detailed information.	HTTPS
Create User	Create User in the Code Versioning Service that maps to the User instance in A&A Service	HTTPS
Delete User	Delete an existing User.	HTTPS
Create User Code Repository Key	Add User's public ssh key, needed for underlying Code Repository Service actions.	HTTPS
Delete User Repository Key	Delete User's public ssh key.	HTTPS
List Groups	Get a list of all the Group instances of the Code Versioning Service.	HTTPS
Show Group Details	Show Group's detailed information.	HTTPS
Create Group	Create Group in the Code Versioning Service that maps to the Group instance in A&A Service	HTTPS
Delete Group	Delete an existing Group.	HTTPS
Get Users in Group	Get a list of all the users that are members in a Code Versioning Group.	HTTPS
List Code Repositories	List existing Code Repositories in the Code Versioning Service.	HTTPS
Show Code Repository Details	Show details of a Code Repository.	HTTPS
Create Code Repository	Create a new Code Repository.	HTTPS
Add Code Repository Member	Add an existing User to a Code Repository.	HTTPS
Remove Code Repository Member	Remove a User from a Code Repository.	HTTPS
Delete Code Repository	Delete a Code Repository.	HTTPS
Get Code Repository Filepath List	Get a Code Repository's directory and file tree in a list.	HTTPS
Sync Files to Code Repository	Synchronize files uploaded from the Network App Catalogue to the Code Repository.	HTTPS

Table 6: ncat – cv interface methods

cli – cv interface

This is a publicly accessible interface, that provides clients direct access to the OSR CV service. The interactions available consist of file and code management in the repositories generated by the OSR NCAT service based on the underlying software used. This interface is directly provided by the underlying open-source software and it was not developed in the context of OSR. It contains all the typical GIT server functionality (eg. Push, Pull etc.). The protocols supported are HTTPS and SSH.

4.2.2. OSR Image Registry

ncat – ir interface

The OSR NCAT communicates with the OSR Image Registry (OSR IR) via the "ncat – ir" interface. The main functionality of the interface is the mapping and management of container images and Helm charts to the corresponding OSR NCAT VDU instances.

Method Name	Description	Protocol
List Users	Get a list of all the User instances of the Image Registry.	HTTPS
Show User Details	Show User's detailed information.	HTTPS
Create User	Create User in the Image Registry that maps to the User instance in A&A Service	HTTPS
List Groups	Get a list of all the Group instances of the Image Registry.	HTTPS
Show Group Details	Show Group's detailed information.	HTTPS
Create Group	Create Group in the Image Registry that maps to the Group instance in A&A Service	HTTPS
Delete Group	Delete an existing Group.	HTTPS
Get Users in Group	Get a list of all the users that are members in a Image Registry Group.	HTTPS
List Image Repositories	List existing Image Repositories in the Image Registry.	HTTPS
Show Image Repository Details	Show details of an Image Repository.	HTTPS
Create Image Repository	Create a new Image Repository.	HTTPS
Get Image Repository Artifacts	Get an Image Repository's Artifacts	HTTPS
Get Image Repository Helm Charts	Get an Image Repository's Helm Charts.	HTTPS
Delete Image Repository Helm Chart	Get a Helm Chart belonging to an Image Repository.	HTTPS
Add Image Repository Member	Add an existing User to an Image Repository.	HTTPS
Remove Image Repository Member	Remove a User from an Image Repository.	HTTPS
Delete Image Repository	Delete an Image Repository.	HTTPS

Table 7: ncat – ir interface methods

cli – ir interface

This interface is provided publicly to clients (users or systems). It offers direct access to images management like upload and download operations.

4.2.3. OSR Event Logging Service

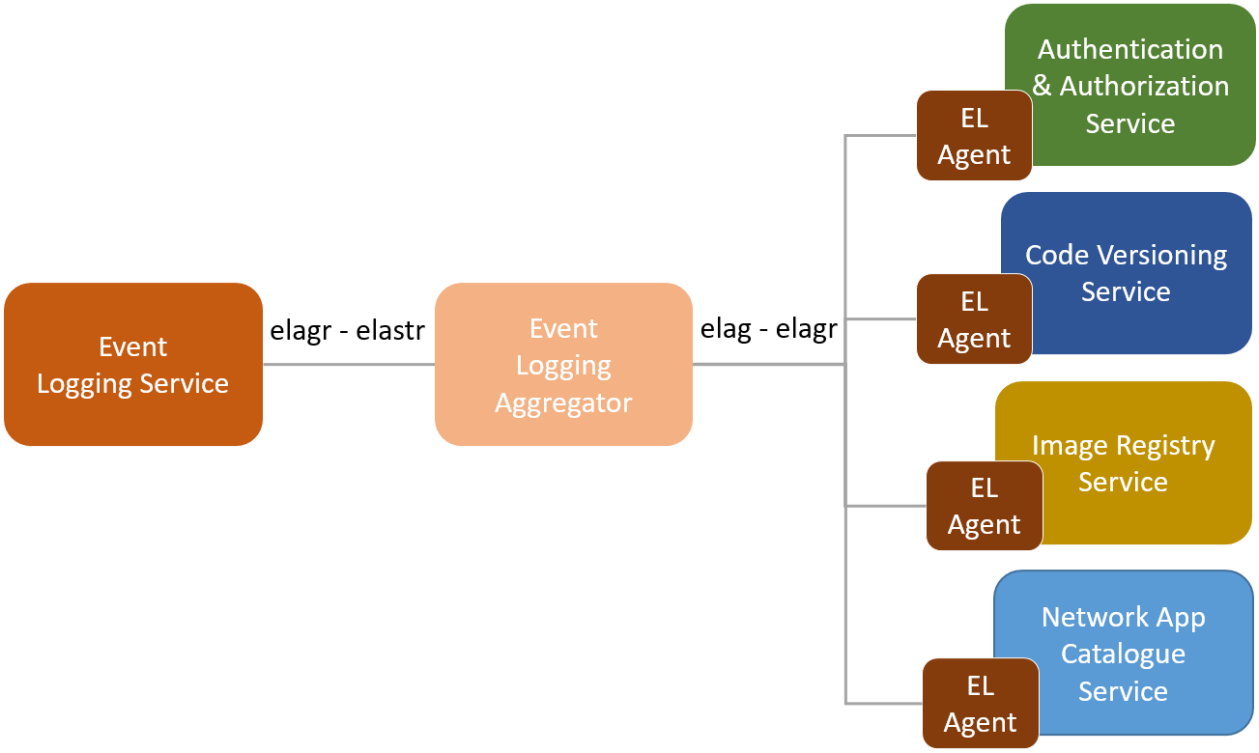

Figure 3: OSR Event Logging internal interfaces

ncat – el interface

This interface interconnects the OSR NCAT with the OSR Event Logging (OSR EL) service for the purpose of retrieving and filtering the stored event logs in the Event Logging Service. The OSR NCAT to OSR EL service interaction includes another interface "elag - ncat" which we consider as separate because it has a very different implementation.

Method Name	Description	Protocol
Get Logs	Generate a list of logs that match the provided filters (user ID, time period, action performed, object ID etc.)	HTTPS

Table 8: ncat – el interface methods

elag – elagr interface

This interface provides the event log gathering functionality from an Event Logging Agent to the Event Logging Aggregator component. The Event Logging Agents are collocated with the monitored service (Network App Catalogue, A&A, Image Registry, Code Versioning services) but they are elements of the Event Logging Service.

elagr – elastr interface

This interface provides the event log storage to the Event Logging Storage after the are aggregated and modified in the Event Logging Aggregator component.

4.3. External Component Interfaces

4.3.1. V&V Interface

The V&V Request Handler is used by the OSR to trigger the verification and/or validation of a Network App when it is uploaded to the OSR. This interface can also be used by external components and third-party users to verify and/or validate their Network Apps. The V&V will return a Unique Identifier (UID) of its run, which can then be used to retrieve information on the verification and validation process.

4.3.2. NAC Interface

The NAC interface of the OSR is used by the NAC to download the Docker images and the Helm charts included in a Network App. And since the OSR embeds a Harbor instance – which implements the OCI distribution specifications – every library and tool compliant with such specification can be used to interact with the OSR. For instance, Docker images can be pulled using the *docker pull* command. Similarly, Helm charts can be pulled from the OSR using the *helm cm-push* command and public libraries e.g., Go Helm Client [14]

For what regards UC1, UC3, and UC4, the NAC, NearbyONE, uses the Go Helm Client to pull the Helm charts, render them, and deploy its releases to the target Kubernetes clusters in the edge nodes. In turn, Kubernetes will use its built-in mechanism to pull Docker images from the OSR; exactly like it would be from Docker Hub or any other registry.

5. Technological Assessment and Implementation

5.1.Technological Assessment

For the **OSR Authentication and Authorization Service**, the **Python** programming language has been used for the driver software implementation.

Python is one of the most famous and widely used programming languages. It is a general-purpose language and it is often used to build websites and software, automate tasks, and conduct data analysis and visualization. Its strength lies to the freedom that it offers to the developer. It is easy to begin with, but it gets complex the more engaged one gets. This comes in contrast with most other famous languages like Java, C and Go that have more complex syntaxes, are less human-readable and present a greater overhead when initiating a new project. On the other hand, Python is beginners-friendly, while it also constitutes a great means for advanced software tasks. Moreover, it is open source, it has a vast and continuously growing archive of modules and libraries and it has a large and active community. Therefore, it creates a great place for discussing possible issues that may come up, it makes development easier and faster and enables developers to find and use the best practices (since everyone is using Python, the best practices may be found implemented in Python!) thus developing more efficient and robust solutions.

Position	PYPL ranking
#1	Python
#2	Java
#3	JavaScript
#4	C#
#5	C/C++

Table 9: Most popular programming languages according to the PYPL index (September 2022). [15]

Table 9: A huge advantage of Python is the wide selection of libraries and frameworks it offers. The development time-to-market will improve by leverage them, since there won't be a need for coding features manually.

One of the biggest criticisms of Python is the runtime, which is relatively slow when compared to other languages.

For **all OSR components** and the **Smart5Grid Platform User Interface**, we have used **Keycloak** [16] open-source software for user management and single-sign-on (based on OpenID connect).

Keycloak is maybe the most popular Identity and Access Management (IAM) solution, among other alternatives like Auth0 [17] and AWS IAM Identity Center [18]. Keycloak is an open-source service that has a large and active community. It helps Software and DevOps Engineers guarantee their platform's security without having to worry about it themselves. Therefore, it saves them quite a lot of time, while it also ensures that the best practices with regards to User Management and Authentication are applied. Moreover, it adds to the system's component-based architecture in contrast with the old-fashioned monolithic architectures and it can also be implemented in an isolated Docker Environment, thus making

it a portable and easy to deploy tool. Keycloak positions its design as primarily for applications and services. The emphasis on third-party application identity security enables monitoring and securing third-party programs with little coding. Yet Keycloak also provides out-of-the-box user authentication and federation. Furthermore, it provides standard protocols, centralized management, password policies, and even social login for Customer Identity and Access Management (CIAM) needs. Keycloak supports SSO "Single-Sign-On", several protocols like OpenID Connect, OAuth 2.0, SAML 2.0, Social media login and supports LDAP and Active directory. It also supports custom password policies. Basically, there is no good reason not to use Keycloak in order to perform IAM. For the above reasons, Keycloak was deemed as the best fit in the framework of the Smart5Grid project.

An alternative that was also considered (but not selected) is OpenIAM [19]. This stands as perhaps one of the most well-known open-source identity management tools; it features Single Sign-On, user and group management, flexible authentication, and automated provisioning—a major component of identity governance and administration. Moreover, OpenIAM aims to help reduce enterprise operational costs and improve identity audits via a centralized control station. The community version doesn't enforce a time limit on subscriptions and benefits from community forum support. The reason we have chosen Keycloak was wider adoption and community support.

For the OSR Network App Catalogue we have used the Python programming language and the Django framework.

Frameworks provide basic infrastructure in order to develop a robust software application. A Python framework, like any framework in other languages, cushions the software development project with a foundation to build on top of it. To be more specific, they render the generic functionality of the program, so developers don't have to start from scratch. Also, they automate standard application building steps, using the respective programming language, thus saving more time on development.

Python frameworks come in various shapes and sizes. Factors that may affect which framework to choose include scalability, expertise, and business-specific goals.

Full-stack frameworks are suitable for both back-end and front-end development. Web development consists of front-end tools for graphic user interface (UI) design and back-end services like databases, security protocols, and business logic. A full-stack Python framework will carry all the equipment needed to facilitate full-stack development. Django is a full-stack framework and is the second most popular Python framework.

Microframeworks are by definition lightweight. In some ways, it is the opposite of a full stack framework. Python's official Flask documentation explains that the "micro" in microframework signifies that the framework's "core [is] simple, but extensible". The components that are fundamental to a full stack framework like a database management system and certain security measures do not come naturally to a microframework. While this might seem like a bad thing, it actually encourages flexibility for developers who want to leverage control over their software, only adding in the relevant third-party libraries when they're completely necessary.

Asynchronous Frameworks: Asynchronous programs are event-driven. Rather than line by line operational handling where one function runs after the other, asynchronous code is non-blocking and doesn't wait for one event to execute before starting another. Because of this parallel programming technique, asynchronous frameworks allow for a profusion of high-performance concurrent connections via running on an async-capable server.

Django is one of the most popular full-stack frameworks in Python and is used in many platforms such as YouTube and Drobox. In comparison with other Python frameworks, Django is backwards compatible, meaning that it offers the provision of working with its older versions and makes use of its older formats and features. It is constantly kept up to an elevated standard, following the most recent patterns in website development and security. It is regularly updated with security fixes, and regardless of whether the developer utilizes an older version of the system, its security is as intact as the new one. Additionally, Django has a large community of developers, making the search for answers to problems much easier.

Django is a good fit for bigger projects, where extensive backend and frontend support is required or in cases where time plays a crucial role, as Django offers a large number of ready components. Coding in Django mostly relies on customizing generic parts of code. The developer must follow a set of rules that come with given element. For projects where a lot of code flexibility is desired, Django might not be the best choice.

Other framework alternatives considered for use in this project are:

Flask: Flask is one of the most popular microframework (meaning it does not require particular tools or libraries) that does not hold any tools or functions that a third-party library can fulfill as essential to its packaging. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for form validation, upload handling, various open authentication technologies and several common framework related tools.

Flask is mainly used for small and medium projects. Setting Flask for a bigger project from the beginning can be tricky.

BlueDream: BlueDream is a Python framework that is most optimal for building medium and large applications. More than a framework, Blue Bream is a server and library too. Some of its best features are its component architecture, transactional object database, and integrated security protocols.

For the OSR Network App Catalogue the PostgreSQL database has been used.

PostgreSQL is a common and well-regarded open-source relational database. It supports a huge number of data types including common database primitives and more importantly network addresses. Network-related types like CIDR addresses, addresses with subnet masks, and MAC addresses, both for IPv4 and IPv6, are useful for a network related application such as the Smart5Grid Platform. Other advantages of PostgreSQL include robust authentication, access control, and privilege management systems suitable for organizations of any size. PostgreSQL has mature user authentication and authorization functionality to define who can use the system and what each user is allowed to see or do. It offers Write-Ahead Logging to provide point-in-time recovery, failover, and streaming replication: these technologies help ensure that the database remains consistent even if the software crashes, and helps copy data between systems for scaling and backing up.

PostgreSQL, together with SQLite and MySQL, and are the three most popular open-source relational database management systems in the world. Each has its own unique features and limitations, and excels in particular scenarios. There are quite a few variables at play when deciding on an RDBMS, and the choice is rarely as simple as picking the fastest one or the one with the most features.

PostgreSQL	SQLite	MySQL
Advantages		
• SQL compliance • Extensible • Open-source and community-driven	• Small footprint • User-friendly • Portable	• Popularity and ease of use • Security • Speed • Replication
Disadvantages		
• Memory performance • Popularity	• Limited concurrency • No user management • Security	• No full SQL compliance • Slowed development • Licensing and proprietary features
When To Use		
• Data integrity is important • Integration with other tools • Complex operations	• Embedded applications • Disk access replacement • Testing	• Distributed operations • Websites and web applications • Expected future growth
When Not To Use		
• Speed is imperative • Simple setups • Complex replication	• Working with lots of data • High write volumes • Network access is required	• SQL compliance is necessary • Concurrency and large data volumes

Table 10: Comparison of PostgreSQL, SQLite and MySQL [21]

For the **OSR Code Versioning Service** the **Python** programming language has been used for the driver software implementation. Furthermore, integration with **GitLab** open-source software has been performed to store object descriptors and keep track of their different versions.

Git versioning is a crucial part of the continuous integration and continuous development process. There is no engineer in the world today that does not use some git versioning tool to keep track of code modifications, safely integrate new features, collaborate with their team, plan, monitor and verify new releases and all these, in a simple and automated way. The most popular git tools undoubtedly are GitHub and Gitlab. In the framework of the Smart5Grid project, Gitlab was chosen as it is an extremely reliable, open-source tool that constitutes a complete DevOps platform and has Continuous Integration/Continuous Delivery (CI/CD) and DevOps workflows already built-in. Apart from being a complete software development solution by its own, Gitlab does offer integrations with some widely used third-party programs and platforms such as Jira, Microsoft Teams, Slack, Gmail, etc., making it more powerful and flexible. Moreover, Gitlab does offer an Enterprise Edition that has even more features than the standard one and comes with user support.

For the **OSR Images Registries** the driver software has been implemented in the **Python** programming language.

Integration with **Harbor** open-source software has been performed to store container images and Helm charts. Harbor is an open-source cloud native registry that stores, signs, and scans container images for vulnerabilities. Harbor is used to solve common challenges by delivering compliance, performance, and interoperability. It is considered as an extension to the open-source Docker Distribution because it has added functionalities usually required by users, for instance security, identity and management. A registry closer to the build and run environment is crucial for the efficiency of image transfer. Harbor supports replication of images between registries, and also offers advanced security features such as user management, access control and activity auditing.

There are other replacements for Harbor for a variety of platforms, including Online / Web-based, Linux, Self-Hosted solutions, SaaS and Docker. A popular alternative is Dedicated Container Registry (a non-free solution in contrast to Docker Hub or Portus).

The **OSR Event Logging Service** is heavily based on **Elasticsearch** [22], **Logstash** [23], and **Filebeat** [24] agents open-source software:

- Elasticsearch as a distributed log database and search engine
- Logstash to modify logs to a unified format
- Filebeat agents in each logged component to gather and push logs to Logstash

The **ELK stack**, comprising of **Elasticsearch, Logstash, Filebeat** (also Kibana [25], while deployed for internal use, is not considered part of the OSR), provides a complete solution for collecting, aggregating, storing, securing, indexing, querying and visualizing data. It allows the developer to easily transform the collected data to a common representation format before storing it to the database for fast querying and retrieval. It is a widely used stack of tools/components, that may be deployed in isolated Docker containers, enhancing even more the component-based architecture of the Smart5Grid platform. Moreover, the ELK stack is quite popular for log management which is critical for diagnosing and troubleshooting issues for optimal application performance, as well as for system monitoring.

There are also other individual components alternative to the sub-components of the ELK stack, like Kafka (for the data collection), scripting languages like bash, or even programming languages like Python (for the data transformation) and other NoSQL databases like MongoDB (for the data storage), but there is no single service that integrates them all in one, complete service able to cover all requirements of the *OSR Event Logging Service*, like the ELK stack does.

5.2. Technological Decisions and Configuration

5.2.1. OSR A&A

As it is mentioned in section 5.1, the underlying open-source software used to implement the OSR A&A was Keycloak and the Single-Sing-On technology that links all OSR subcomponents is OpenID Connect. The OSR consists of several different components, each of which requires secure user authentication to provide users with access to the application's features. An example of this challenge is having a user

owning resources on the OSR Code Versioning service and also, on the OSR Image Registry service. Having multiple separate user management points and methodologies that need to be synchronized would result in a highly complex, hard-to-manage, and error-prone procedure. The solution adopted using Keycloak and OpenID Connect helps us overcome this challenge.

The authentication protocol enabled in Keycloak is OpenID Connect (OIDC). OIDC is an authentication protocol that is an extension of OAuth 2.0. OIDC is a complete authentication and authorization protocol. It leverages Json Web Token (JWT) set of standards which define an identity token in JSON format and ways to digitally sign and encrypt that data in a compact and web-friendly manner. In order, to better explain the OIDC process of authentication and authorization we present the definition of its basic concepts. [22]

"Client" is an OAuth 2.0 Client using OpenID Connect, can also be referred to as Relying Party (RP).

"Claim" is a piece of information asserted about an Entity. An "Entity" is something that has a separate and distinct existence and that can be identified in a context.

"End-User" is a human participant and is one example of an Entity.

"Claim Type" is the syntax used for representing a Claim Value. This specification defines Normal, Aggregated, and Distributed Claim Types.

"Claims Provider" is a server that can return Claims about an Entity.

"Credential" is data presented as evidence of the right to use an identity or other resources.

"Subject Identifier" is a locally unique and never reassigned identifier within the Issuer for the End-User, which is intended to be consumed by the Client. The subject Identifier type we are using is "pairwise". This provides a different sub value to each Client, so as not to enable Clients to correlate the End-User's activities without permission.

"ID Token" is a JSON Web Token (JWT) that contains Claims about the Authentication event. It may contain other Claims.

In our case for the authentication, we use the "Authorization Code Flow" which can be described by the following steps:

1. Client prepares an Authentication Request containing the desired request parameters.
2. Client sends the request to the Authorization Server.
3. Authorization Server Authenticates the End-User.
4. Authorization Server obtains End-User Consent/Authorization.
5. Authorization Server sends the End-User back to the Client with an Authorization Code.
6. Client requests a response using the Authorization Code at the Token Endpoint.
7. Client receives a response that contains an ID Token and Access Token in the response body.
8. Client validates the ID token and retrieves the End-User's Subject Identifier and the exposed Claims.

The figure below Figure 4 depicts the authentication interactions of a user login to the Platform User Interface.

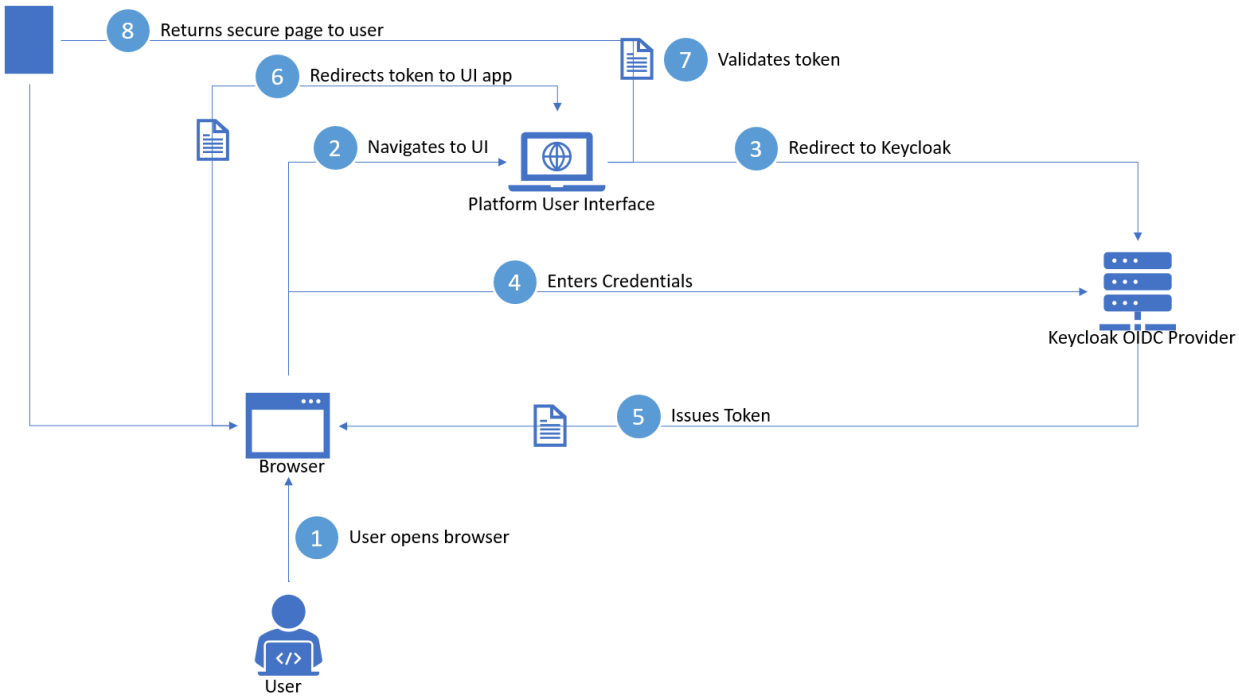

Figure 4: OpenID Connect Platform UI interactions

The exposed Claims on the OSR Network App Catalogue context include the "groups" information which determine which groups the users can access and on what level based on the role included in the group claim. The group ids in the OSR A&A claim are mapped to the Group instances created in the OSR Network App Catalogue, the OSR Code Versioning, and the OSR Image Registry services. User access to these instances is determined by this mapping. As defined in deliverable 2.2 [13] in section 3.3 "Smart5Grid User Roles and Scenarios" the roles a user can have on an instance are "default", "developer", and "admin". "default" role mean read access only, "developer" role means access to modify instances and "admin" full access. Additionally, a user that is the creator (or "owner") of a n instance has full control over this instance.

The OSR A&A service is deployed as a Helm chart on the AXON on premises Kubernetes cluster. It consists of a Keycloak deployment based on the "Codecentric AG" open-source release of the Keycloak chart and the overlay OSR A&A driver application that was necessary for the integration with the OSR Catalogue Service. Data persistence was provided by a database in the common PostgreSQL cluster. Public network communication was made available using the common Load Balancer (HAProxy) and Kubernetes NodePort service type. The software application was uploaded to AXON's internal Harbor repository in Docker image format and a Helm chart for the definition of all the necessary Kubernetes manifests. This Helm chart includes an existing open-source Helm chart which provides the Keycloak Kubernetes manifests. The configuration of the Helm chart was uploaded to AXON's internal GitLab repository, which was linked to an ArgoCD application manifest, responsible to deploy the complete stack of manifests to the Kubernetes cluster when changes git merged on GitLab repository's master branch. This deployment setup was used for all the OSR components with minor adjustments when needed or for simplicity reasons.

5.2.2. OSR Network App Catalogue

The OSR Network App Catalogue is a service in which the main logic of the OSR is implemented. It communicates with and coordinates all the other OSR components. For the OSR CV service and the OSR IR, the OSR NCAT communicates with the corresponding developed driver for each component. All the actions initiated by the OSR NCAT are propagated to the corresponding component as a REST API request. As they are defined in section 4.2 most of the cli – ncat methods that serve requests that originate from external actors (users or services) are being translated to methods supported by ncat – cv and ncat – ir interfaces. In the case of the OSR EL service, the OSR NCAT communicates directly with the Elasticsearch API but limits the exposure of logs only to the permitted for each user. The integration with the OSR A&A service is implemented using the "mozilla-django-oidc" authentication and access management library. The OSR NCAT is set up as an OpenID Connect "Client" in the OSR A&A context with almost administrative access to all of OSR A&A resources.

The OSR NCAT fully uses the data model described in section 3.2. In order to provide a comprehensive explanation of how the models are linked and interact with each other we describe the internal actions and communications occurring in the main functionalities offered by the OSR:

- The user registers a new user directly to the OSR A&A service, this functionality is only provided by a web browser interaction. This causes a user to be created at the OSR A&A service. Then the user requests an access token from the OSR NCAT. The OSR NCAT by making use of the registered "Client" retrieves the token and returns it to the user. This token can be used for several requests until it is expired.
- The user creates a VDU on the OSR NCAT. A VDU object that contains the information on the actual software image that is going to be used. The user is assigned the "developer" role for this object (the VDU). Via the ncat – ir interface a group and a project are created in the OSR IR the user is also assigned the "developer" role for these resources in the OSR IR context. Moreover, via the ncat – cv interface a new group and a new repository are created in the OSR CV service. Then the user can upload the VDU application's code to the newly created code repository. After the development process has reached a point of publishing or testing the VDU the user creates an image from the code and uploads a docker image or a Helm chart directly to the created registry. Creating and uploading the image can be done either using automation features provided by the underlying GIT server software GitLab such as the GitLab-CI or by other means of user preference. In either case, this step requires some manual configuration from the part of the user. By following this process multiple VDUs can be created at this stage.
- Then the user creates a Network App object in the OSR NCAT causing a new repository to be created at the OSR CV via the ncat – cv interface. The user is assigned the "developer" role in both the context of the OSR NCAT and the OSR CV. The Network App has no content at this point. To add content the user can add content by uploading a Network App descriptor. The uploaded data is then parsed and stored in the repository. If the descriptor contains descriptors of Network App subcomponents such as Network Services or VNFs, these are also created as OSR NAC objects with references to their corresponding committed versions.
- Once the Network App is created, the user can use an alternative method to upload the descriptors. Using directly the OSR CV the user can add or modify the repository contents and when a new version needs to be known to the OSR NAC the user can use the "Sync from the Code Versioning Service" action to upload it. This creates a new instance of descriptor objects in the OSR NAC using the contents of the chosen commit.

All Network App sub-components are meant to be reusable. The OSR NCAT provides this functionality via the "fork" action. By "forking" a component OSR NCAT creates a duplicate component of the version specified a new object is created in the OSR database and a new repository is created in the OSR CV having the contents of the repository specified.

Deleting objects follows the opposite process of, firstly, removing entities from the OSR CV and OSR IR components and then from the OSR NCAT, using ncat – cv and ncat – ir interfaces respectively.

During this whole process, we have omitted to mention that in each action performed on every OSR component logs have been generated for each of the events. Such events can be queried by the user. A query in the events triggers an OSR NCAT to translate it and send an expanded query to be performed to the OSR EL through the ncat – el interface.

The OSR-NAC is developed as a Python web application using the Django framework using the common PostgreSQL cluster as the database. It is deployed on the AXON on-premises Kubernetes cluster as a Kubernetes service using the required manifests (service, deployment, pod, configmap, secret).

5.2.3. OSR Code Versioning Service

The OSR Code Versioning service is an application that provides a REST API for GIT server operations in order to be compatible with the OSR Catalogue workflows. In section 4.2.1 "OSR Code Versioning Service Interfaces" all the provided functionality of the OSR CV service is described. To accomplish this, we have developed these interfaces as a Python application that translates the requested action into actions to be performed to the GIT server to perform the required result. As the GIT server we have chosen GitLab open-source software. This GitLab instance differs from the internal GitLab instance that we use for our code development purposes. The OSR CV service is not bound with the GitLab software, a different GIT server software could be used with the development of a new driver. However, we leverage GitLab functionality by providing direct access to the users of the OSR. Instances of GitLab resources that are created by the OSR such as projects and repositories can be accessed directly from the GitLab user interface. While, the functionality is limited, in order not to intervene with the OSR workflows, this provides an additional tool in the hands of the OSR user and specifically to the developers of Network Apps. For the integration with the OSR A&A, we have enabled OmniAuth GitLab feature which allows multiple authentication providers. We have configured OmniAuth with OSR A&A OpenID Connect provider, in that way GitLab resources can be mapped to users created in the OSR A&A and they can access the GitLab instance directly. In the figure below we can see the additional button for the OSR A&A single-sign-on option in the GitLab login page.

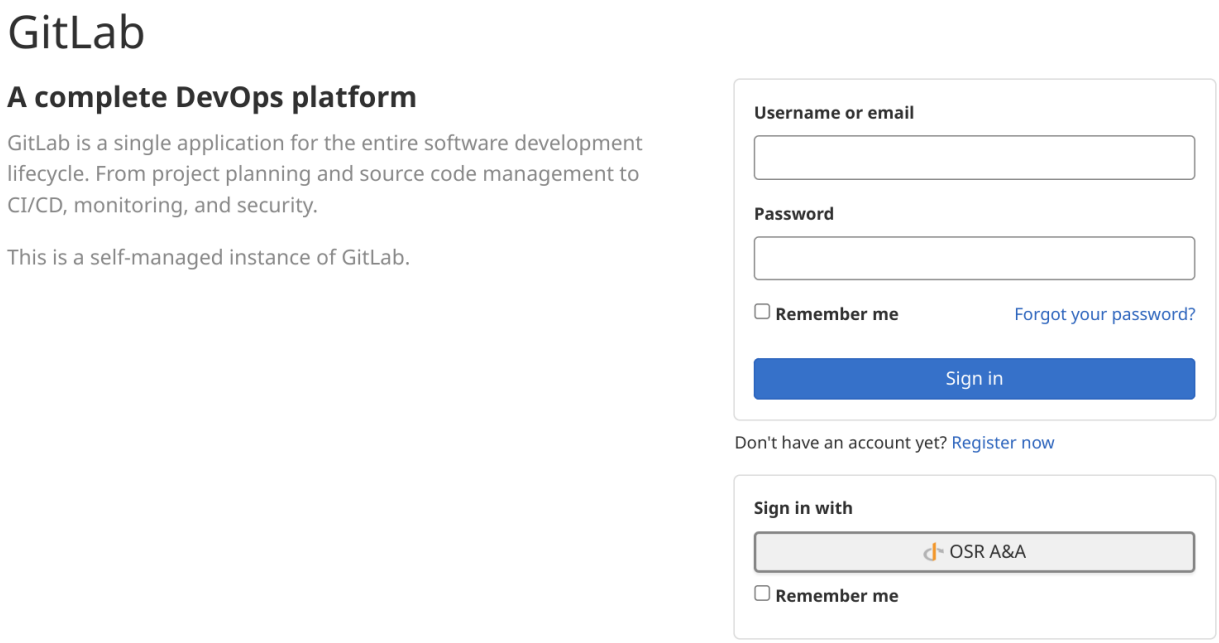

Figure 5: GitLab OSR A&A Single-Sign-On Login

When a Network App is created, a corresponding repository is created in GitLab by the OSR and the initiator user is granted access to this repository. When a Network App Descriptor is uploaded, a commit is merged to the master branch of the repository. Different versions of the Network App descriptors correspond to different commits in the repository.

5.2.4. OSR Image Registry

Given that in all of the Smart5Grid Use Cases the Network Apps are implemented in either Docker containers or Helm charts, the OSR Image Registry are exactly these two: a Docker container registry and a Helm chart registry. The chosen backend to host such registries is Harbor open-source software as it supports both of them. In this case, again, we have developed an overlay driver application to facilitate the integration of the OSR NCAT with the Harbor interfaces. The OSR NCAT communicates with the Harbor APIs via the OSR IR driver software. It transforms the OSR requests to a series of Harbor API requests that result in the desirable outcome. While, this logic could have been implemented directly to the OSR Catalogue code, we have added this layer to provide modularity in the underlying Image Registry backend (in our case Harbor). The users of the OSR have direct access to the Harbor software as image access and manipulation directly from the registry is the most popular way developers interact with them and often required by applications, for example in order to deploy these containers or charts to a Kubernetes cluster the developer must setup a direct, secure connection to the registry. For the single-sign-on integration with the OSR A&A we have configured its OpenID Connect provider to Harbor. This allows users to manage their own images in Harbor registries created by the OSR Catalogue service. Having the OSR A&A provider configured provides the "LOGIN VIA OIDC PROVIDER" button option in the Harbor login view.

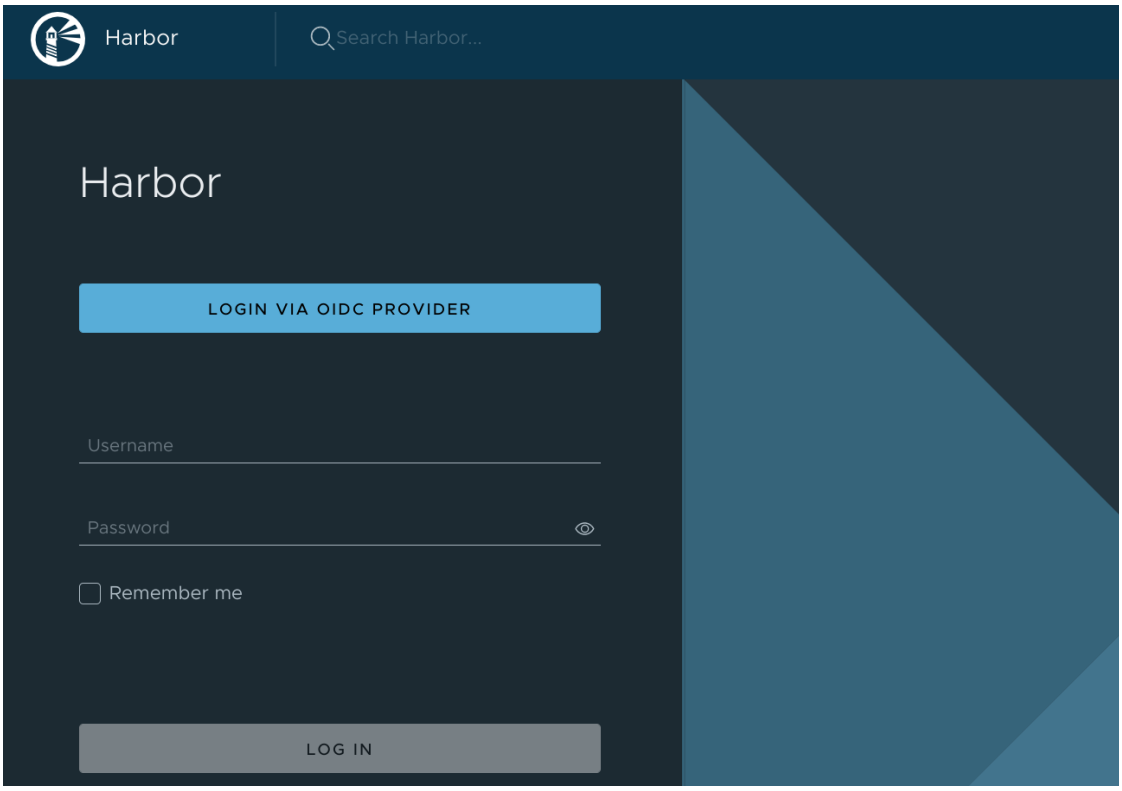

Figure 6: Harbor OSR A&A Single-Sign-On Login

The OSR IR utilizes the common PostgreSQL database cluster for data persistence and it is deployed as a Helm chart on the AXON on premises Kubernetes cluster.

5.2.5. OSR Event Logging Service

The OSR Event Logging service is based on the underlying open-source software components used. Log persistence was implemented on Elasticsearch, log Aggregation on Logstash, and log pushing from the containers toward the Elasticsearch-Logstash by the Filebeat Agents. The Filebeat agent is deployed as a Kubernetes Daemonset. This means that an instance of Filebeat exists on all the Kubernetes cluster nodes and has all the necessary permissions to access the logs of any container from which we need to pull logs. Using filters in the configuration we define the label of the pods that belong to the Kubernetes Services that we want to monitor. The services that we monitor are the OSR A&A, the OSR NCAT, the OSR CV service, and the OSR IR. The Filebeat is also configured to send the logs to the Logstash service. Logstash is also deployed as a Kubernetes service and receives and parses all the logs coming from the Filebeat agents. Using conditions and filter in the Logstash configuration we filter only the logs occurring as a result of OSR users' actions. In this step logs also get formatted in a unified format for better information clarity. Logstash is, also, configured to propagate the modified and filtered logs to the Elasticsearch service. The Elasticsearch service receives and stores the logs arriving from the Logstash service. Direct access to the Elasticsearch cluster is not allowed to OSR's external users. OSR exposes a limited set of logs to the users with the ability to query the logs that relate to only the resources that are available for each user. All the components of the OSR EL service are deployed as separate Helm charts on the AXON on premises Kubernetes cluster.

5.2.6. Platform User Interface

The web application was developed using React [27], no middleware server is provided, and the client directly calls the exposed OSR API. The web app is designed as follows, if you are not authorized you access a page where there is a button that allows login, there is no direct interface because for login as required, we use the OIDC authentication system. After logging in you access the dashboard, where 4 cards are shown related to available resources and list of events (Event Logs), also, there is a side menu that allows access other features. The structure of the web application is shown in the figure "Components of the web application"

The generic architecture is shown in the following figure.

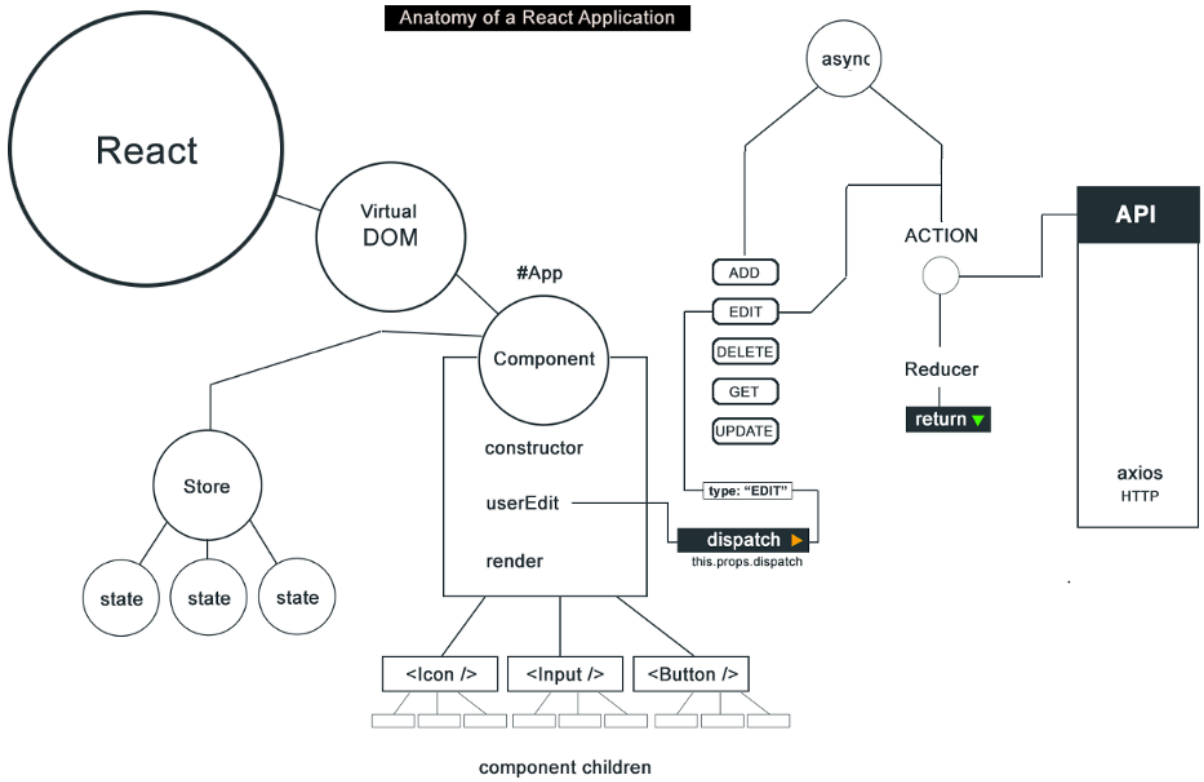

Figure 7: Platform UI ReactJS Architecture

The following figure shows the directory structure of the application

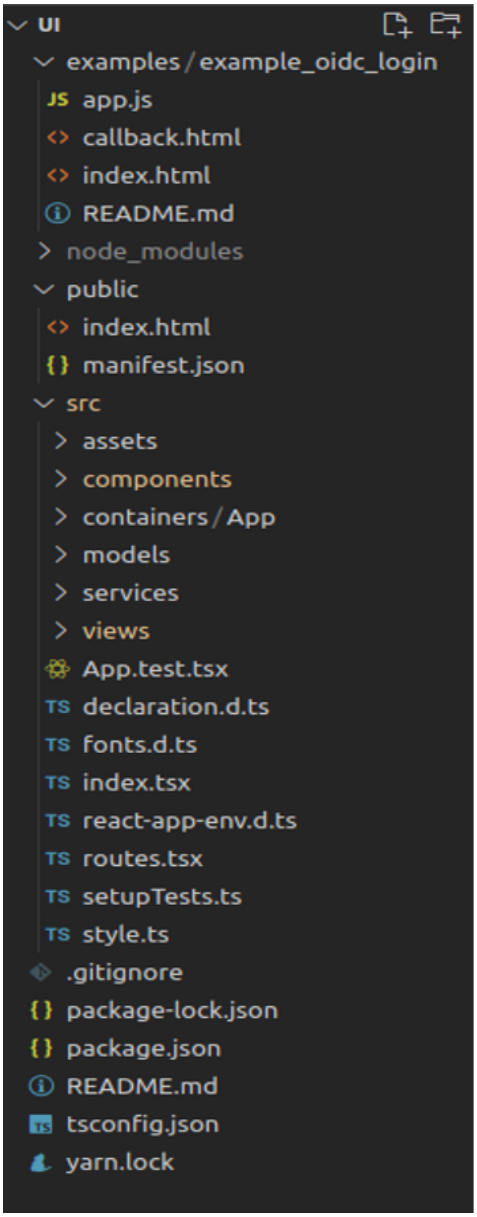

Figure 8: Platform UI code directory structure

In the following figure we can see how the web application is structured in detail

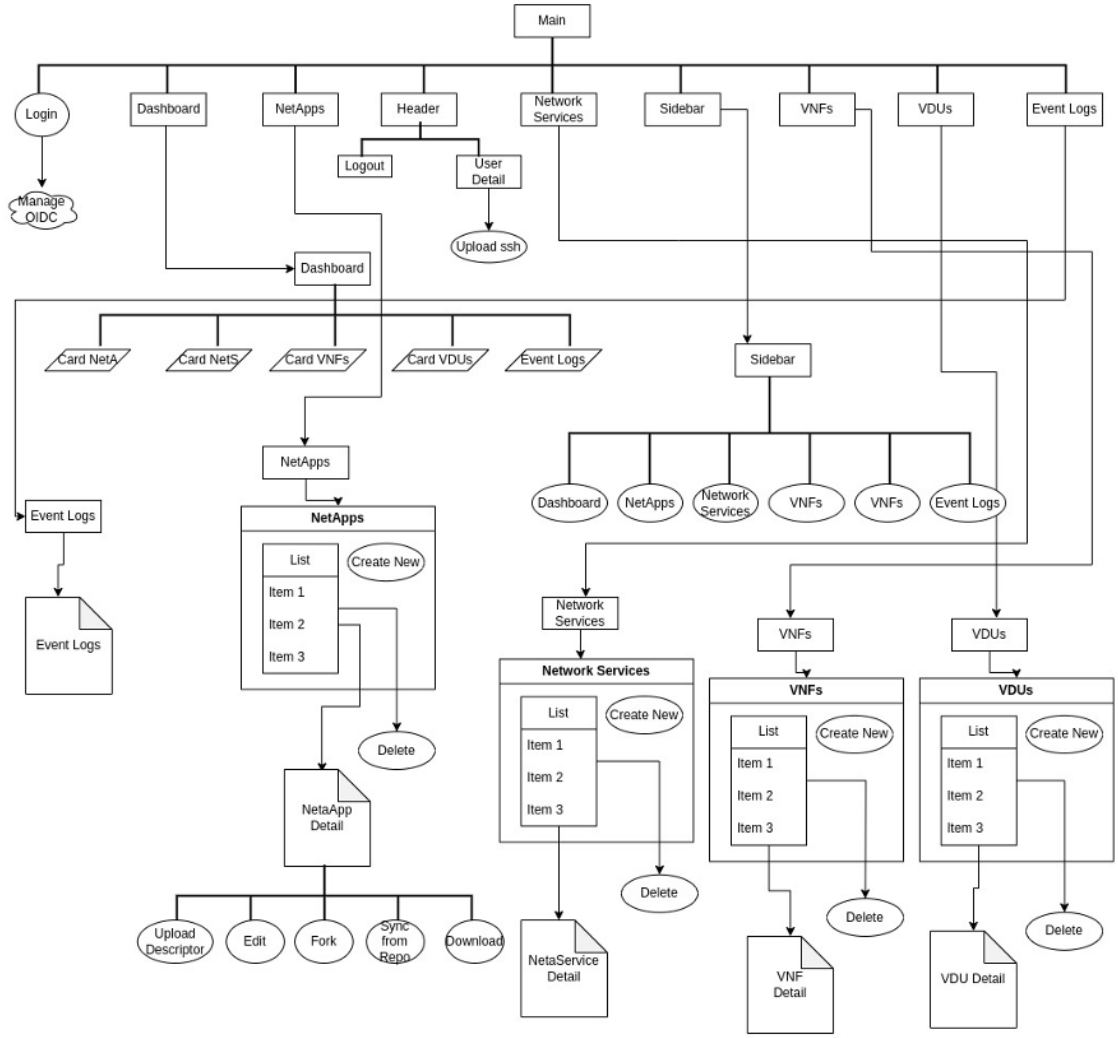

Figure 9: Platform UI tree structure

The main view is nothing more than the login button container, this is done by calling the *MainButtonsWrapper* component, which in turn calls the *Login* component. The user is presented with the Login view when first accessing the platform.

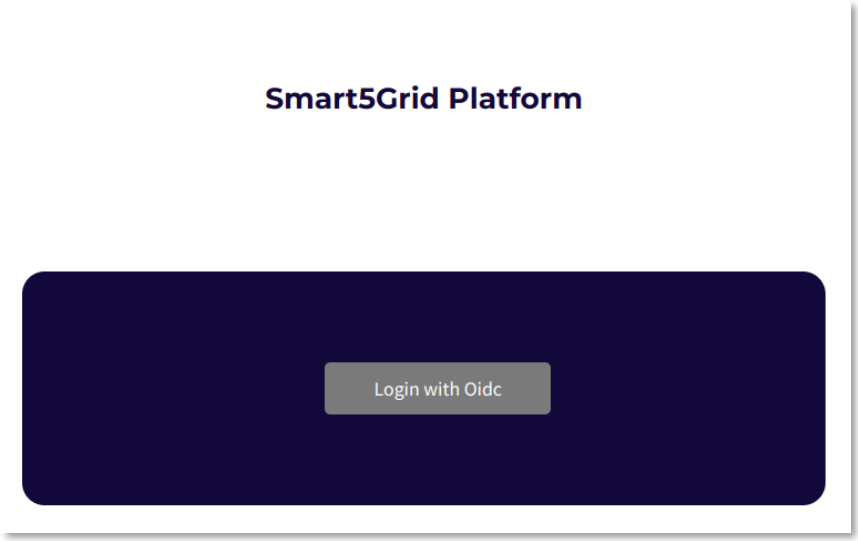

Figure 10: Platform UI Login View

There is no login interface provided, the *Login* component was developed using the *react-oidc-context* module, this module requires configuration of some parameters as provided by the specification, i.e. authority, client_id, redirect_uri, client_secret, response_type, scope, post_logout_redirect_uri. The logic is implemented as follows: after clicking on the login button the user is redirected, thanks to the oidc module, to the existing login interface If the user is already registered, he/she can login. Otherwise, the user can register via the sign-up button provided in that interface. After inputting the authentication data, OIDC redirects to a callback page, such a component was developed so that it redirects to the dashboard, eliminating unnecessary parameters from the URL bar.

The header is displayed only if the user is successfully logged in. It provides a logo, information about the logged in user and a logout button. Also, if the user clicks on the displayed username (top right corner), he/she can access the user details page. All SSH Keys will be shown here and the user can also upload new public keys here.

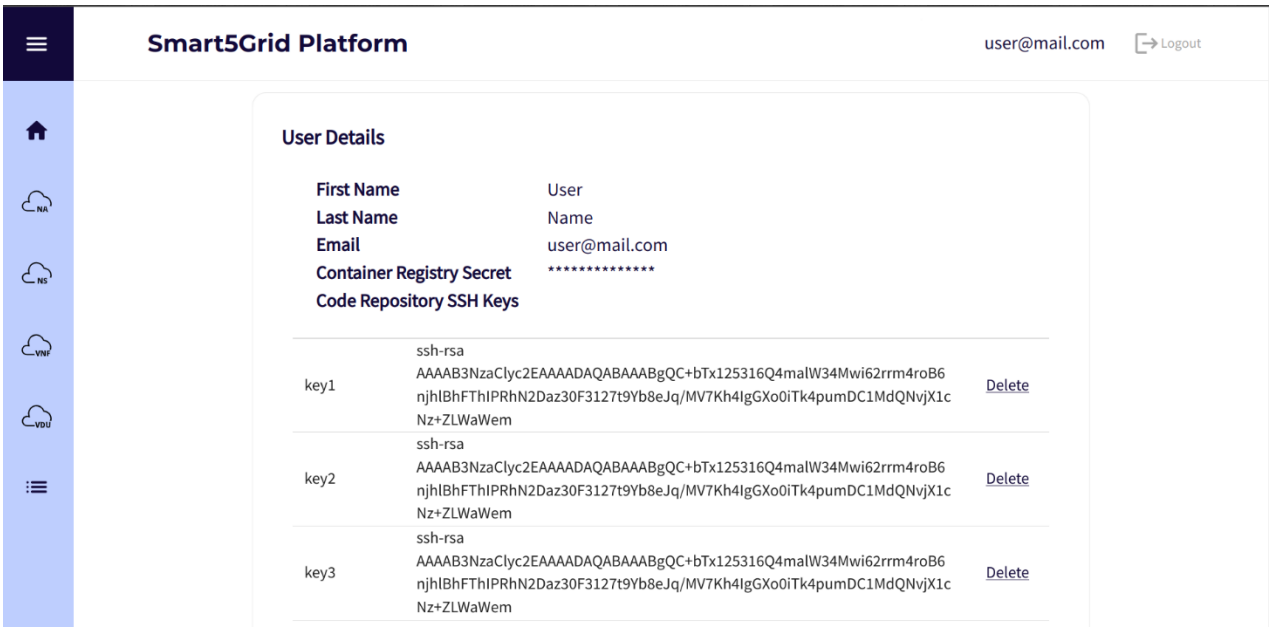

Figure 11: Platform UI User Details View

The public key upload modal will appear. After filling in the required info, the new public key will be added and displayed in the Code Repository SSH Keys table.

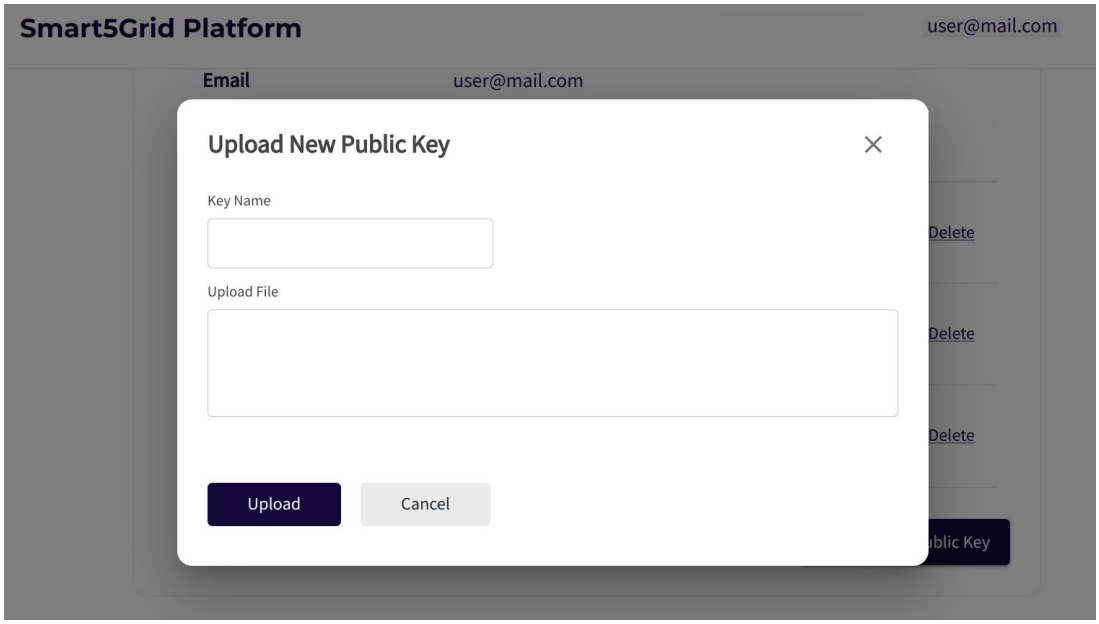

Figure 12: Platform UI Upload Repo Key Modal

After logging in, the user is redirected to the dashboard, which shows all the instances available to this user, as well as the most recent event logs. In the dashboard view the UI calls the OSR API to enumerate Network Apps, Network Services, VNFs, and VDUs. This allows the different cards to be built. The view at the bottom of the screen corresponds the returned event logs.

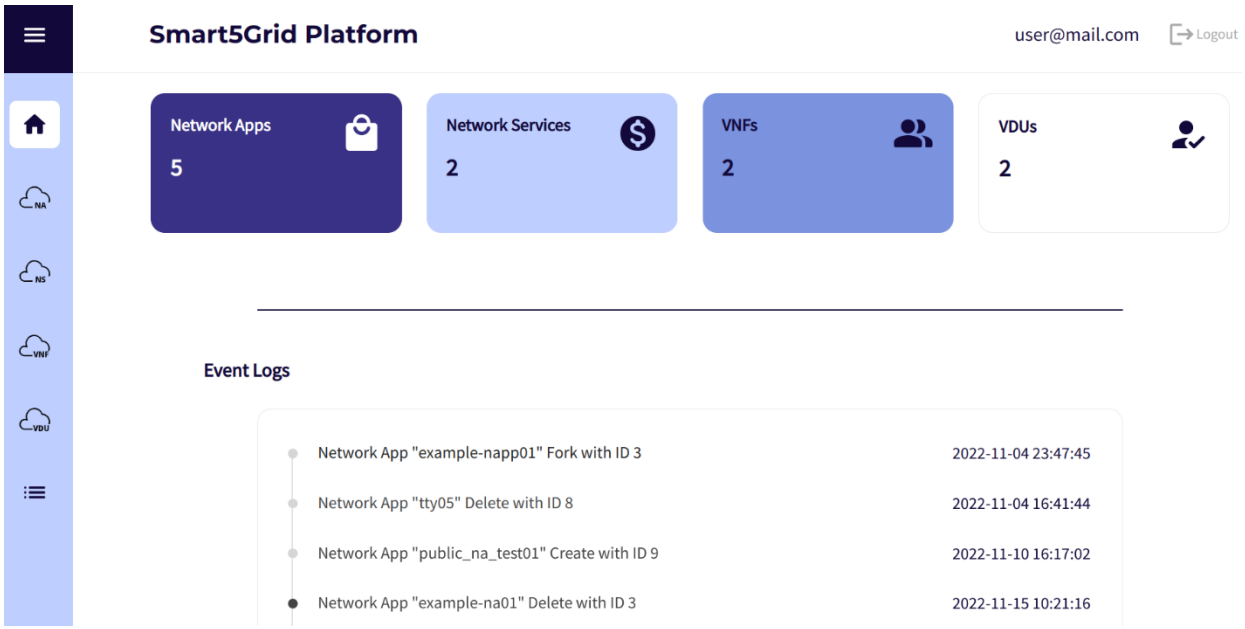

Figure 13: Platform UI Dashboard View

A more detailed navigation sidebar can be displayed by clicking on the hamburger button at the top of this navigation.

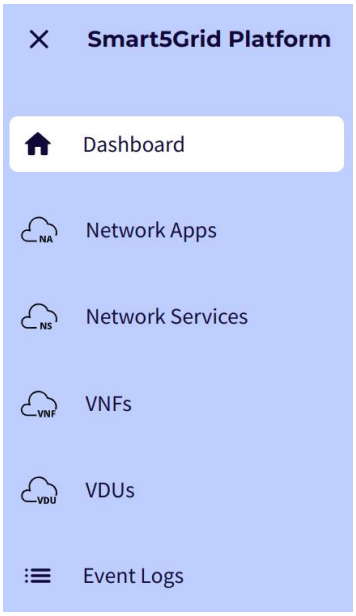

Figure 14: Platform UI Sidebar

The Network App List View displays all the Network Apps visible to the user. At the end of each row there is a "Delete" button that deletes the Network App (after confirmation), if the user has the necessary permissions.
The "Create New" button at the top right brings forward a modal window to create a new Network App.

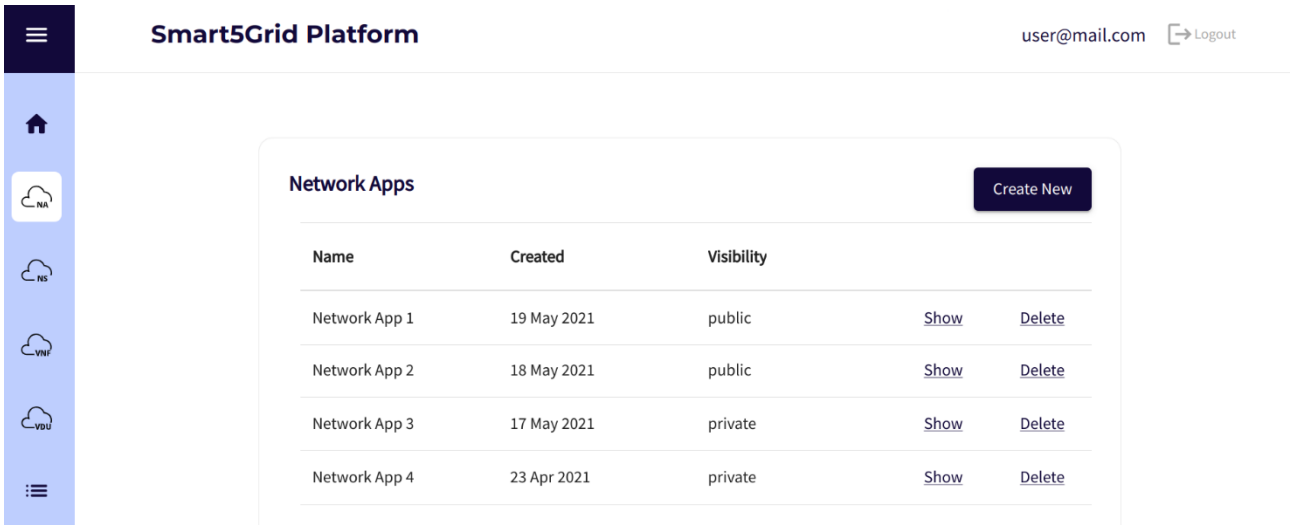

Figure 15: Platform UI Network App List View

Network App Details view shows the details of a Network App. The drop-down menu right next to the Network App Name is showing all the descriptor versions of the Network App. By selecting a different Network App descriptor version, the "Descriptor" section of the view gets updated with the specified descriptor's YAML representation. On the top right there is a "Select Action" drop down menu showing the available actions on the Network App. "Upload Descriptor" displays a modal window to upload a new Network App Descriptor version. "Edit" shows a modal window to modify main Network App fields (eg. Description, Visibility etc). "Fork" creates a duplicate Network App owned by current user (after confirmation). "Sync from repo" creates a new Network App descriptor from the current version of the master branch of the corresponding OSR CV code repository (Descriptor Link repository). "Download" downloads the Network App descriptor. "Delete" deletes the Network App (after confirmation). The Event Logs section at the bottom of the view shows the event logs related to this Network App.

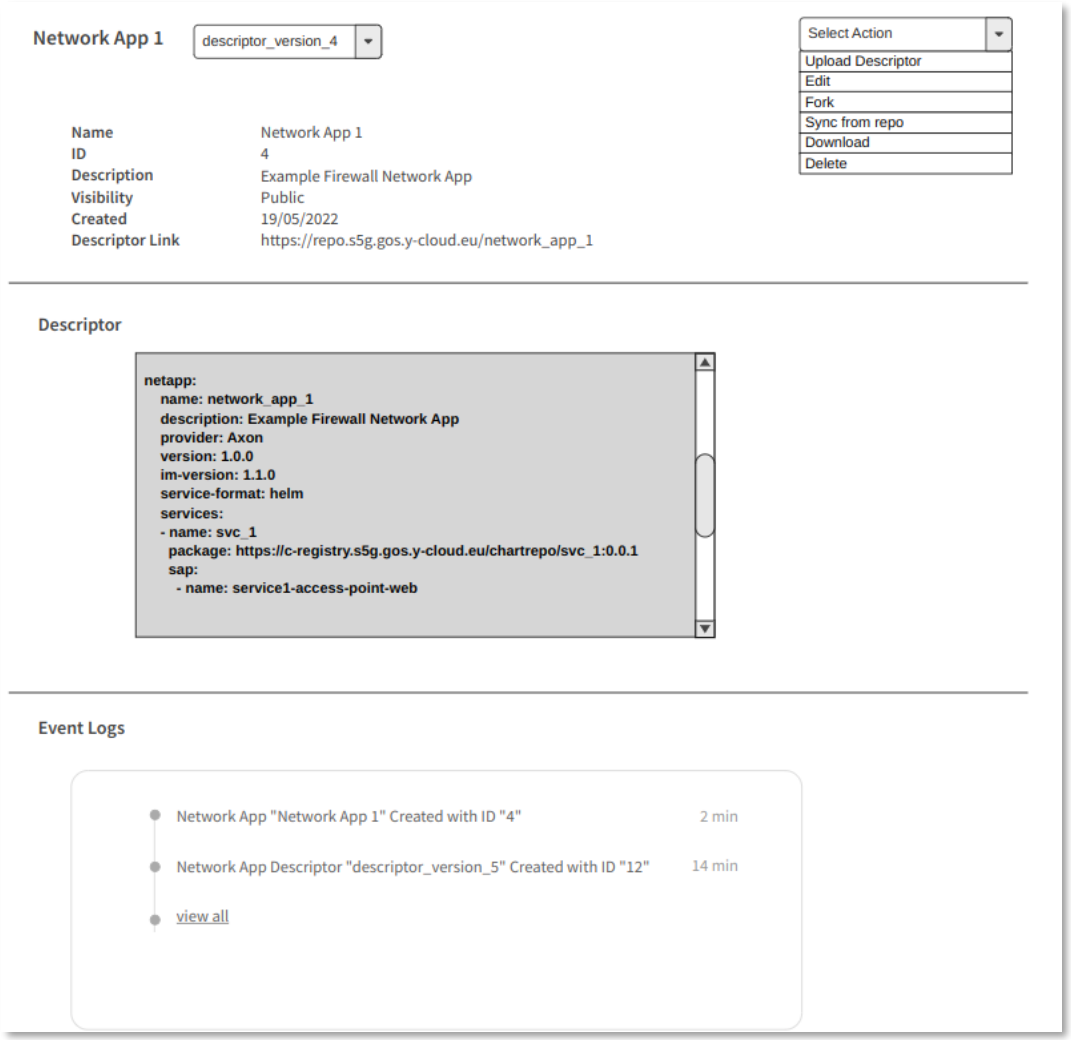

Figure 16: Platform UI Network App Details View

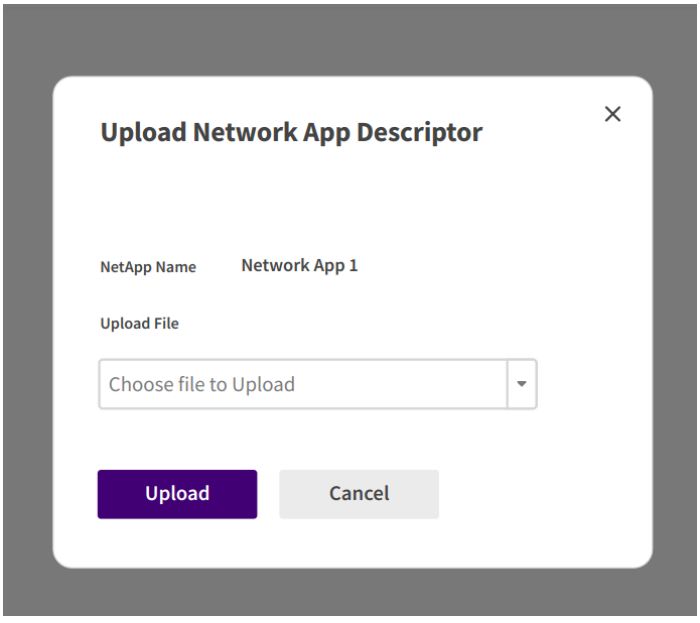

Figure 17: Platform UI Upload Network App Descriptor Modal

Views handling of the Network Service, VNF, and VDU instances are similar in design with the Network App Views. Aesthetic changes to the Platform UI are likely to occur until the public release of the Platform.

5.2.7. External Interfaces Integration

After a new Network App is created, the V&V can be triggered to ensure the Verification and/or Validation of a Network App. The Network App descriptor is then retrieved from the OSR by the V&V and propagated to the NAC along with the required testing parameters. The NAC retrieves the referenced Network App data from the OSR.

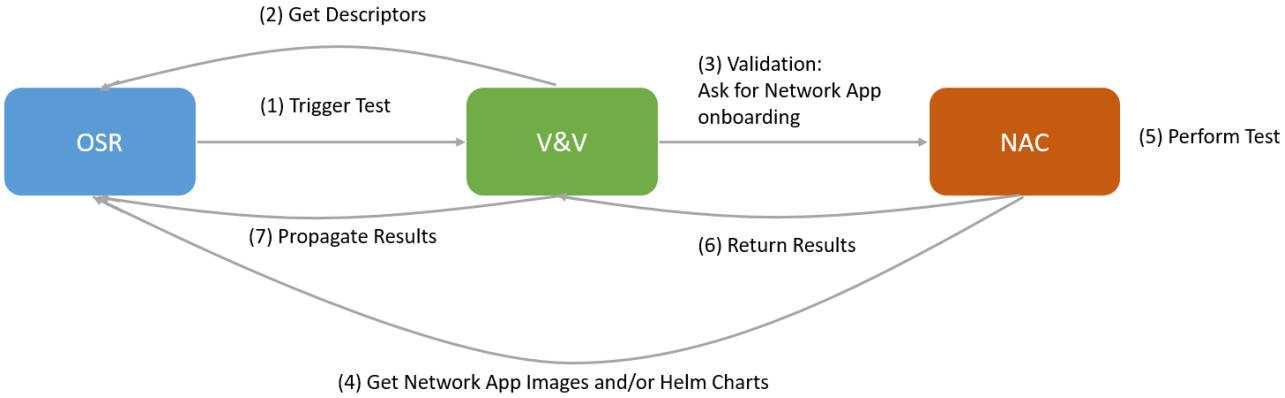

Figure 18: OSR - V&V - NAC workflow

The designed and defined architecture of a V&V framework considers the NFV automatic testing & verification framework as a continuous integration, while including UI and Results Manager to simplify the usage of the V&V by anyone. A Request Handler / API, V&V Engine, Results Manager, Verification Engine was developed and integrated with the OSR (to download the Network App) and with the Network App Controller for Helm Chart based Network Apps to onboard and validate the Network App.

This V&V flow encompasses the following steps:

1. The V&V is triggered by the OSR
2. The V&V runs its verification and validation engine
    a. Downloads Network App – OSR integration
    b. Verifies the Network App – ATOS Verification Engine integration
    c. Validates the Network App – NAC integration
    d. Stores results – Results Manager integration

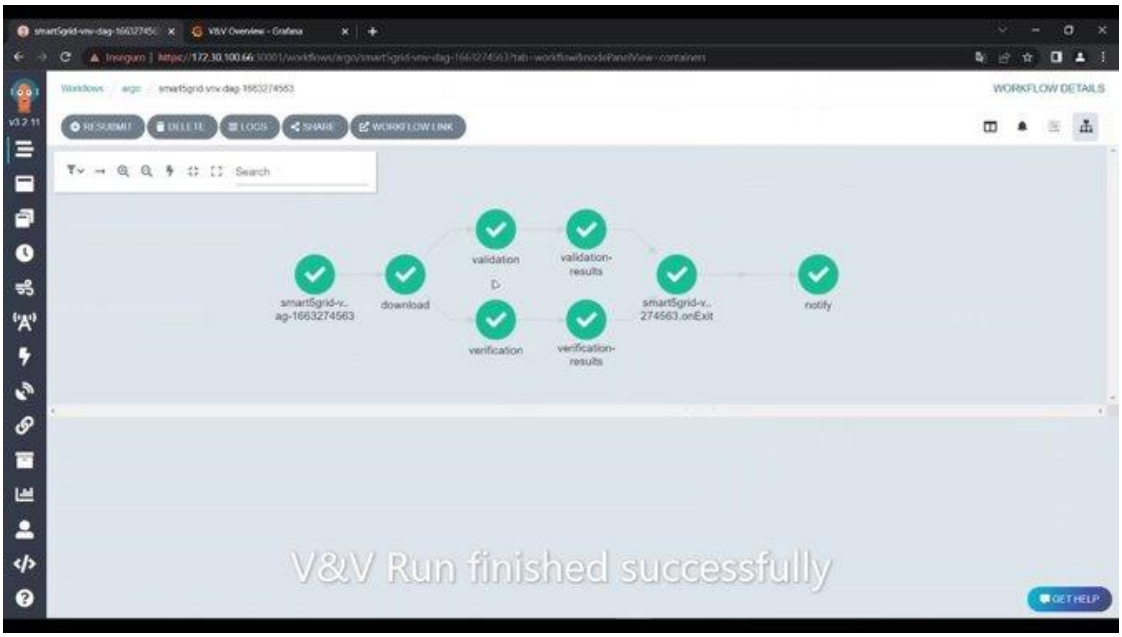

Figure 19: V&V test job execution

3. The V&V makes the results available to others in a UI

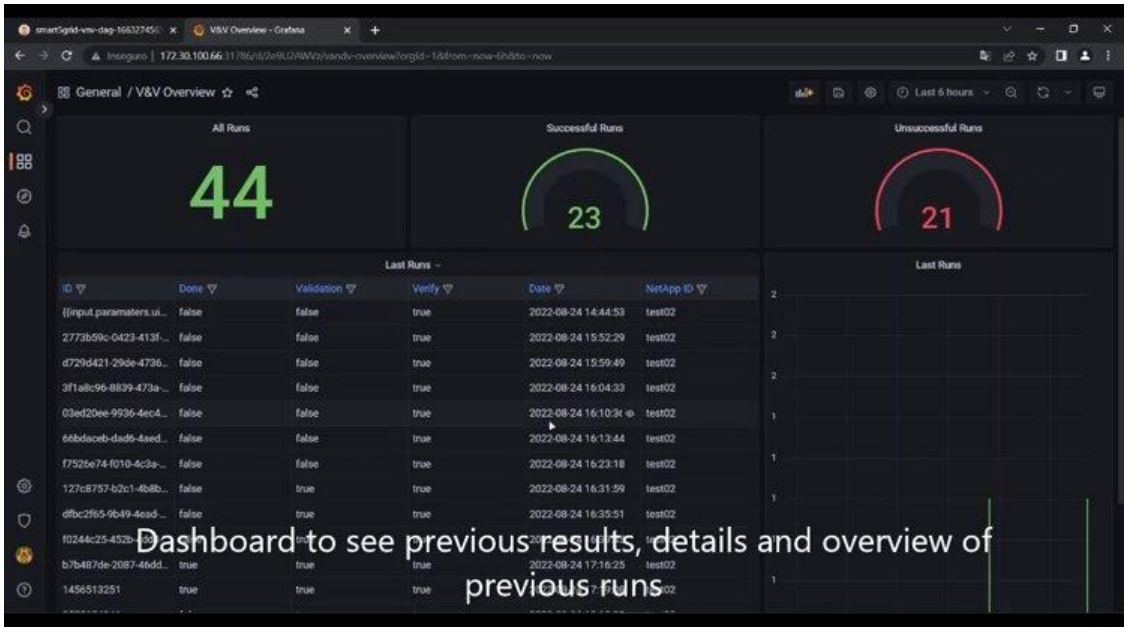

Figure 20: V&V test result dashboard

4. And in a database accessible programmatically.

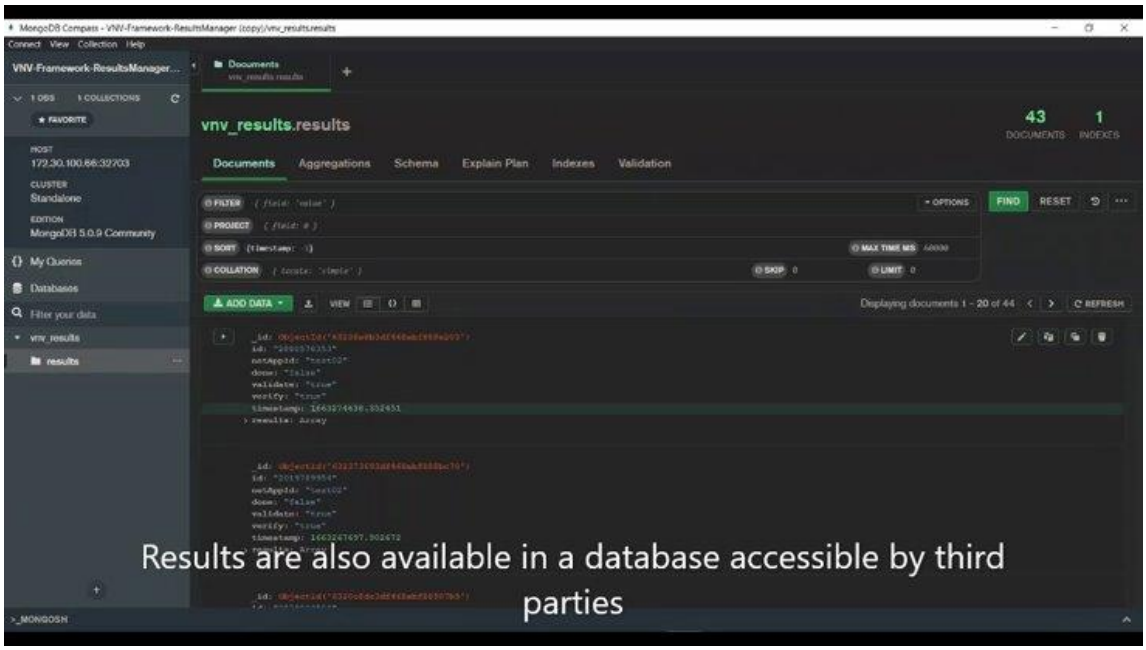

Figure 21: V&V test results in the database

The NAC receives a request for the Network App onboarding from the V&V, step 3 in Figure 18. In order to retrieve the Network App data, the NAC pull the image or Helm chart from the OSR. This functionality is offered by the OSR Image Registry service. It consists of a PULL action targeting the Network App related Docker or Helm repository in the OSR Image Registry.

Finally, test results returned to the V&V by the NAC (step 6) are propagated by the V&V to the OSR and stored in the Network App descriptor information (step 7).

5.3. Laboratory Deployment

For the purposes of the project, we had to deploy a new laboratory environment with different layers of virtualization that would allow us to deploy software in a flexible manner. We needed automated ways in order to be able to provision baremetal nodes, virtual machines and containers, as well as a reliable network storage solution.

For the baremetal node provisioning we chose MAAS (Metal As A Service) open-source tool that offers server discovery, commissioning, deployment and configuration. MAAS includes a PXE/preseed service, stores the operating system boot images and is able to automatically deploy nodes according to the given configuration. Moreover, MAAS offers deploying simple KVM hosts for virtual machine provisioning.

To have better control over the laboratory's local network we deployed a local DNS service. This allowed us to configure network-level domain mappings to better simulate public network environment before. The open-source software we used is dnsmasq.

For the storage needs we used a simple QNAP appliance storage unit having 3 x 4TB HDD drives in a RAID 5 array for disk stripping with parity and a 256GB SSD drive for cache to improve performance. This provided an NFS network storage for all project storage needs.

To follow the microservices approach in our developed software we chose to host a Kubernetes cluster. Kubernetes is an open-source tool for open-source system for automating deployment, scaling, and management of containerized applications. The variation of Kubernetes we deployed is called K3S and is preferred when using lower end servers as a lightweight alternative. We used 3 control nodes for control plane high availability and 3 worker nodes for application side high availability. For the k3s cluster persistence a three node PostgreSQL cluster was used one in every control node. In order to use the NFS storage in an automated way we deployed the Kubernetes NFS Subdir External Provisioner. This allows requesting PVCs (Persistent Volume Claims) of a storage class related to the existing NFS network storage mentioned in the previous paragraph.

For the Container Network Interface (CNI) we have chosen the default option for a K3S cluster, Flannel network fabric with VXLAN backend for packet encapsulation. In order to provide connectivity to our Kubernetes services from external networks we followed a hybrid approach using a NGINX ingress controller with Kubernetes ClusterIP services and an external HAProxy Load Balancer with Kubernetes Node Port type services. A Service is a Kubernetes object that acts as an endpoint for enabling the communication between various components within and outside the application. In other words, a service is a stable address for pods. The three important Service types in Kubernetes are ClusterIP, Node Port, and Load Balancer. ClusterIP is used to group pods together and provide a single interface to access them. For example, an incoming request by another service will be forwarded to one of the pods in the ClusterIP randomly. NodePort is a Kubernetes service type that listens on a port on the node and forwards requests on that port to a pod on the node. When a request is received by a specific node, this node acts as a built-in load balancer and sends the request to one of the pods at random. The decision to use both Node Port with external Load Balancer and ClusterIP with Ingress Controller services was to facilitate setting up the different services when one of the options was causing issues during the assessment period of choosing our technology stack.

For simplicity and higher agility in the development and testing phase we also needed a simple KVM host for the various virtual machines deployed. The HAProxy external Load Balancer was deployed as a Docker container in the KVM host.

The services we deployed for the software development process of the OSR and the Smart5Grid Platform UI are a GitLab deployment for the code repositories, Harbor as a private Docker registry to host the created images, and ArgoCD for deployment automation following the GitOps paradigm.

5.4. CI/CD Pipelines

5.4.1. Development Workflow

Figure 22 depicts an example CI/CD workflow during of development of the OSR services. The developers write new code and commit source code changes, implementing new features or integrating endpoints, and push their code to GitLab, the central source code repository. The commit automatically triggers GitLab CI service to pull, build and test the committed source code from the specific feature branch of the code repository. If the unit tests finish successfully, GitLab CI uses Docker to create a Docker image, which then is pushed to the private Harbor Docker Registry with an appropriate tag. Once components have been built and their images have been pushed to the Docker Registry, the GitLab CI optionally deploys the OSR components new images to a development namespace as containerized services for further manual testing.

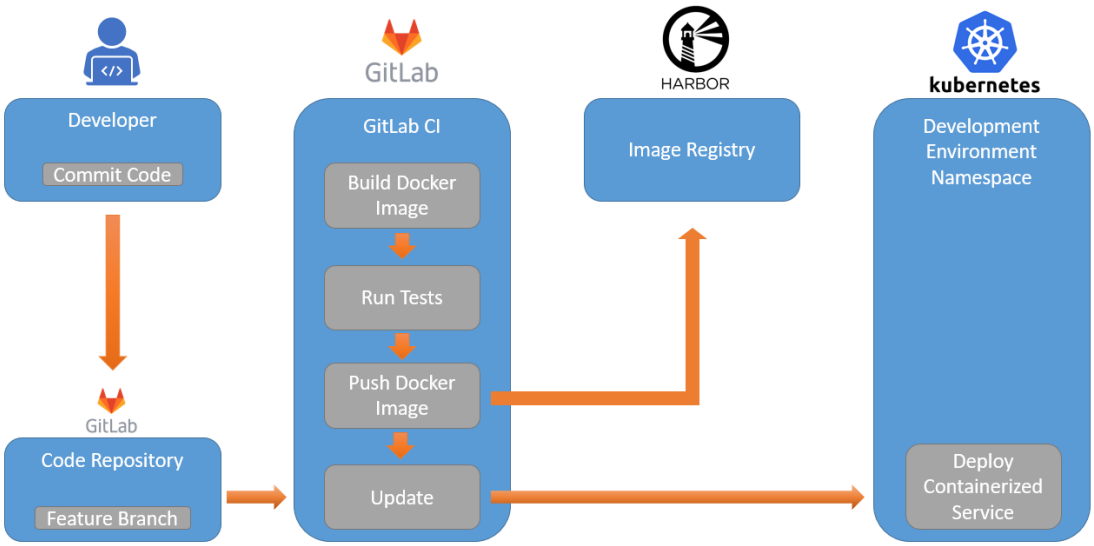

Figure 22: OSR Development Workflow

5.4.2. Deployment Workflow

Figure 23 depicts the deployment workflow that is executed when the development of a new feature is completed and a new version of the software application is ready to be released. In this scenario, the developer creates a new merge request to merge all the changes in the source code of the feature branch to the master branch of the repository. When the merge request gets approved, GitLab triggers ArgoCD to update container image tag in the deployed Application object. This causes ArgoCD to automatically sync the actual deployed service in Kubernetes production namespace (Kubernetes deployment object) to the new version.

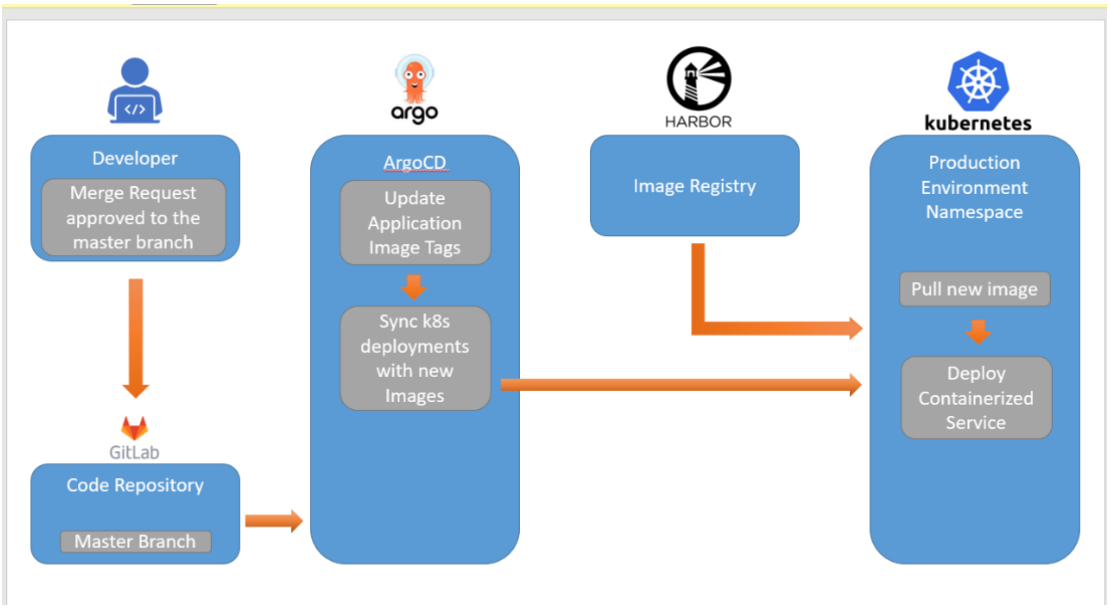

Figure 23: OSR Deployment Workflow

5.5. Testing

For the OSR components testing we have created unit tests to ensure that individual units of the code keep performing the required functionality successfully after the changes being merged to the code during the development process. The OSR components are all developed in Python programming language so the tool we have used to perform the unit tests is "tox", currently on of the most used test tool for Python programming. Tox is a generic virtual environment management and test command line tool, it aims to automate and standardize testing Python and easing the packaging, testing and release process of Python software. We add tox commands in our GitLab CI pipelines to perform the required tests. The pipeline can only continue to the next stage if all tox tests have completed successfully.

We can summarize the tox process in the following steps:

- tox generates a series of virtual environments
- it generates dependencies for each environment, as defined in the tox configuration
- it runs the setup commands for each environment
- finally, it returns the results from each environment to the user

In order to assure that the high-level functionality of the OSR backend is provided as described we have created a series of functional tests. This testing procedure is based on the open-source Robot Framework tool. Robot Framework has an easy syntax, utilizing human-readable keywords. For out tests we have mainly used the "HTTP RequestLibrary" which greatly facilitates performing HTTP requests and compare the returned response with the expected one. Moreover, Robot Framework supports adding a "teardown" stage at the end of each test to remove any objects or artifacts created during the test execution.

6. Conclusions and Future Work

This document presented the implementation details of the Open Service Repository and the Platform User Interface in order to support Network App development in a collaborative environment and storage within the Smart5Grid framework. It explains interfaces, architectural design, and integration with external components.

The official development period for the Smart5Grid OSR is completed by the release of this document. The OSR will be used in the future both for the realisation of the Smart5Grid Use Cases showcasing and experiments as well as for other activities beyond Smart5Grid project. During the operational phase of the OSR we will collect and analyse feedback and use it to create a loop of information that will be used to provide suggestions to improve the performance, functionality and user experience of the OSR. In the future additional enhancements can be introduced by implementing more drivers for open-source DevOps tools that can further automate and facilitate the development process of Network Apps and by providing more interfaces with respect to deploying Network Apps on production infrastructure and managing their lifecycle.

7. References

[1] Open Container Initiative Distribution Specification
https://github.com/opencontainers/distribution-spec/blob/main/spec.md

[2] Harbor, the trusted cloud native registry for Kubernetes
https://goharbor.io/

[3] Docker Hub
https://hub.docker.com/

[4] RedHat Quay
https://quay.io/

[5] Artifact Hub
https://artifacthub.io/

[6] VMware Bitnami
https://bitnami.com/

[7] Direito, R., Gomes, D., Trantzas, K., & Tranoris, C. (2022, September). 5GASP's approach to the onboarding, deployment and validation of 5G NetApps. In 2022 IEEE International Mediterranean Conference on Communications and Networking (MeditCom) (pp. 78-81). IEEE.

[8] EVOLVED-5D Deliverable D2.2 "Design of the NetApps development and evaluation environments"

[9] 5G-IANA – Deliverable D2.1 Specifications of the 5G-IANA architecture

[10] VITAL-5G Deliverable D1.2 system specifications and architecture

[11] 5G-MediaHUB Deliverable D2.1 Northbound APIs, NetApps and NetApps Repository development – Initial

[12] The YANG Data Modeling Language RFC
https://www.rfc-editor.org/rfc/rfc7950

[13] Smart5Grid Project (2021), Smart5Grid deliverable D2.2 "Overall Architecture Design, Technical Specifications and Technology Enablers"

[14] Go Helm Client
https://pkg.go.dev/github.com/mittwald/go-helm-client#section-readme

[15] PYPL PopularitY of Programming Language
https://pypl.github.io/PYPL.html

[16] Keycloak, open source identity and access management solution
https://www.keycloak.org/

[17] Auth0, authentication and authorization platform
https://auth0.com/

[18] AWS IAM Identity Center, centrally managed access to multiple AWS accounts and applications
https://aws.amazon.com/iam/identity-center/

[19] OpenIAM, converged identity platform
https://www.openiam.com/

[20] Django Framework, open-source, Python-based web framework
https://www.djangoproject.com/

[21] SQLite vs MySQL vs PostgreSQL: A Comparison of Relational Database Management Systems https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems

[22] Elastisearch, distributed, search and analytics engine https://www.elastic.co/what-is/elasticsearch

[23] Logstash, data processing pipeline https://www.elastic.co/logstash/

[24] Filebeat, lightweight shipper for logs https://www.elastic.co/beats/filebeat

[25] Kibana, user interface for Elasticsearch data https://www.elastic.co/kibana/

[26] OpenID Connect Core 1.0 specification - Section 1.2. Terminology https://openid.net/specs/openid-connect-core-1_0.html#Terminology

[27] React, a JavaScript library for building user interfaces https://reactjs.org/

8. Appendix A: OSR REST API

OSR Authentication and Authorization Interfaces

Get OpenID Connect Access Token

Get an OpenID connect access token bearing the user's claims.

POST - "/auth/realms/s5g/protocol/openid-connect/token"

Headers:

- Content-Type: application/x-www-form-urlencoded

Attribute	Type	Required	Description
grant_type	String	Yes	OIDC grant type (use "password" for cli access).
client_id	String	Yes	OIDC client ID.
client_secret	String	Yes	OIDC client secret.
scope	String	Yes	OIDC scope. (default: "openid")
username	String	Yes	OIDC username.
password	String	Yes	OIDC user password.

Returns:

- 200 OK – On success

Example output:

```
{
  "access_token":"....ACCESS TOKEN REDACTED.....",
  "expires_in":600,
  "refresh_expires_in":0,
  "token_type":"Bearer",
  "id_token":"....ID TOKEN REDACTED.....",
  "not-before-policy":0,
  "session_state":"####-####-####",
  "scope":"openid profile email"
}
```

Create User

Creates a user in the A&A Service. This request requires elevated permissions, available only to admin users and to the Network App Catalogue Service.

POST - "/auth/realms/s5g/protocol/openid-connect/token"

Headers:

- Content-Type: application/x-www-form-urlencoded
- Authorization: Bearer $oidc_access_token

Attribute	Type	Required	Description
firstName	String	Yes	User First Name.
lastName	String	Yes	User Last Name.
email	String	Yes	User email.
enabled	String	Yes	Specifies whether the user will be enabled.
username	String	Yes	User username.

Returns:

- 201 Created – On success

OSR Catalogue Interfaces

User REST API:

Show User Detailed View

Get a specific User's detailed fields.

GET - "/users/details/"

Headers:

- Content-Type: application/json
- Authorization: Bearer $oidc_access_token

Returns:

- 200 OK – On success

Example output:

```
{
  "username": "name@mail.com",
  "enabled": true,
  "emailVerified": true,
  "firstName": "John",
  "lastName": "Doe",
  "email": "name@mail.com",
  "keys": [
    {
      "id": 5,
      "title": "demo-ssh-key",
      "created_at": "2022-05-13T01:38:51.676Z",
      "expires_at": null,
      "key": "ssh-rsa ######"
    }
  ],
  "container_registry_secret": "******",
  "code_versioning_service_activation": "Done",
  "image_registry_activation": "Done"
}
```


List User Repository Keys

List a specific User's public ssh keys.

GET - "/users/repokeys/list"

Headers:

- Content-Type: application/json
- Authorization: Bearer $oidc_access_token

Returns:

- 200 OK – On success

Example output:


```
[
  {
    "id": 5,
    "title": "demo-key1",
    "created_at": "2022-05-13T01:38:51.676Z",
    "expires_at": null,
    "key": "REDACTED"
  },
  {
    "id": 6,
    "title": "demo-key2",
    "created_at": "2022-06-19T02:17:46.292Z",
    "expires_at": null,
    "key": "REDACTED"
```

```
    }
]
```

Create User Repository Key

Add User's public ssh key, needed for underlying Code Repository Service actions.

POST - "/users/repokeys/create"

Headers:

- Content-Type: application/json
- Authorization: Bearer $oidc_access_token

Attribute	Type	Required	Description
key_title	String	Yes	The title for the uploaded public SSH key.
key	String	Yes	Content of the public SSH key.

Returns:

- 200 OK – On success

Delete User Repository Key

Delete a User's public ssh key.

DELETE- "/users/repokeys/delete/:id"

Headers:

- Content-Type: application/json
- Authorization: Bearer $oidc_access_token

Attribute	Type	Required	Description
id	Integer	Yes	The ID of the public SSH key.

Returns:

- 200 OK – On success

Network App REST API:

List Network Apps

Get a list of all visible Network Apps across the OSR for the authenticated user.

GET - "/netapp/list/"

Headers:

- Content-Type: application/json
- Authorization: Bearer $oidc_access_token

Example output:


```
[
   {
      "id": 1,
      "name": "example-np",
      "description": "Network App description",
      "latest_version": "",
      "public": false,
      "repository_url": "git@repo_url:example-np/example-np.git",
      "user": 1,
      "created_at": "2022-10-25T17:19:35.996474Z",
      "updated_at": "2022-10-25T17:19:35.996528Z",
   }
]
```


Show Network App Detailed View

Get a specific Network App's detailed fields.

GET - "/netapp/detail/:id"

Headers:

- Content-Type: application/json
- Authorization: Bearer $oidc_access_token

Attribute	Type	Required	Description
id	Integer	Yes	The id of the NetApp.

Returns:

- 200 OK – On success
- 404 NetApp not found - if the NetApp does not exist or cannot be accessed by the requester.

Example output:


```
{
  "id": 1,
  "latest_version": "1.0",
  "name": "example-np",
  "description": "",
  "public": false,
  "repo_project_id": 127,
  "repo_group_id": 288,
  "repository_url": "git@repo_url:example-np/example-np.git",
  "identity_group_id": "c6cc7534-2724-43ef-a058-c44be9cbd878",
  "user": 1,
  "created_at": "2022-10-25T17:53:46.055379Z",
  "updated_at": "2022-10-25T17:53:46.055403Z",
  "descriptors": [
    {
      "id": 1,
      "commit": "0e1d0aae24b59013fd16520bd4a43bb142e05ec3",
      "version": "1.0",
      "validated": false,
      "verified": false,
      "created_at": "2022-10-25T18:20:41.748032Z",
      "updated_at": "2022-10-25T18:20:41.748058Z",
      "netapp_tests": [],
      "data_formated": {
        "netapp": [
          {
            "name": "demo_np",
            "description": "A demo network app",
            "provider": "Axon",
            "version": 1,
            "service": {
              "type": "nbc",
              "nsd": [
                {
                  "nsd-id-ref": "demo_ns_nsd",
                  "vnf-ref": [
                    {
                      "vnf-id-ref": "demo_vnf_vnfd"
                    }
                  ]
                }
              ]
            }
          }
        ]
      },
      "data": "REDACTED: same as above but single line with new line
characters"
    }
  ]
```

```
}
```

Create a Network App

Create a new Network App. Available only for users who can create Network Apps.

POST - "/netapp/create/"

Headers:

- Content-Type: application/json
- Authorization: Bearer $oidc_access_token

Attribute	Type	Required	Description
name	String	Yes	The name of the Network App.
description	String	No	The description of the Network App.
public	Boolean	No	Specifies if the new Network App will be publicly available or not. Default value is "false".

Returns:

- 200 OK – On success

Example output:

```
{
  "id": 1,
  "name": "example-np",
  "description": "Network App description text",
  "public": false,
  "name": "example-np",
  "repository_url": "git@repo_url:example-np/example-np.git",
  "created_at": "2012-10-12T17:04:47Z",
  "updated_at": "2012-10-12T17:04:47Z"
}
```

Update a Network App

Create a new Network App. Available only for users who can update the specified Network App.

PUT - "/netapp/update/:id"

Headers:

- Content-Type: application/json
- Authorization: Bearer $oidc_access_token

Attribute	Type	Required	Description
id	Integer	Yes	The ID of the Network App.
description	String	No	The description of the Network App.
public	Boolean	No	Specifies if the new Network App will be publicly available or not. Default value is "false".

Returns:

- 200 OK – On success

Example output:

```
{
  "id": 1,
  "name": "example-np",
  "description": "Network App description text",
  "public": false,
  "name": "example-np",
  "repository_url": "git@repo_url:example-np/example-np.git",
  "created_at": "2012-10-12T17:04:47Z",
  "updated_at": "2012-10-12T17:12:24Z"
}
```

Delete a Network App

Delete a new Network App. Available only for users who can delete the specified Network App.

DELETE - "/netapp/delete/:id"

Headers:

- Content-Type: application/json
- Authorization: Bearer $oidc_access_token

Attribute	Type	Required	Description
id	Integer	Yes	The ID of the Network App.

Returns:

- 200 OK – On success

Fork Network App

Create a new Network App being the exact duplicate of the selected Network App. The new Network App will be owned by the initiator user.

GET - "/netapp/fork/:id"

Headers:

- Content-Type: application/json
- Authorization: Bearer $oidc_access_token

Attribute	Type	Required	Description
id	Integer	Yes	The ID of the Network App.
fork_name	String	Yes	The name of the forked Network App.

Returns:

- 200 OK – On success

Sync Network App from Code Versioning Service

Synchronize the content of the code existing in the related repository in the Code Versioning Service to the data presented by the OSR Catalogue Service.

GET - "/netapp/sync/:id"

Headers:

- Content-Type: application/json
- Authorization: Bearer $oidc_access_token

Attribute	Type	Required	Description
id	Integer	Yes	The ID of the Network App.

Returns:

- 200 OK – On success

Upload Network App Descriptor

Upload the Network App descriptor files archived and compressed in "tar.gz" format. Available only for users who can alter the content of the specified Network App.

POST - "/netapp/upload-descriptor/:id"

Headers:

- Content-Type: multipart/form-data
- Authorization: Bearer $oidc_access_token

Attribute	Type	Required	Description
id	Integer	Yes	The ID of the Network App.
descriptor_file	File	Yes	Compressed archive containing Network App descriptor files in yaml format. It can also include Network App sub-component descriptor files (Network Services, VNFs). Network App descriptor yaml file must contain a "version" field.

Returns:

- 200 OK – On success

Download Network App Descriptor

Download the Network App descriptor archive file, compressed in "tar.gz" format.

GET - "/netapp/download-descriptor/:id"

Headers:

- Content-Type: application/json
- Authorization: Bearer $oidc_access_token

Attribute	Type	Required	Description
id	Integer	Yes	The ID of the Network App.
version	String	Yes	Network App descriptor version.

Returns:

- 200 OK – On success

Perform V&V test on a Network App

Perform a test via the V&V Platform on a specific version of a Network App. Available only for users who can alter Network App content.

POST - "/netapp/perform-test/"

Headers:

- Content-Type: application/json
- Authorization: Bearer $oidc_access_token

Attribute	Type	Required	Description
netapp_id	Integer	Yes	The ID of the Network App.
netapp_version	String	Yes	The version of the Network App descriptor.
validate	Boolean	Yes	Request V&V validation test.
verify	Boolean	Yes	Request V&V verification test.

nac_to_use	String	No	Specify the Network App Controller to be used for the deployment of the Network App resources for the test. Available choices “osm” and “nbc”.

Returns:

- 200 OK – On success

Example output:

```
{
  "id": 1,
  "nac_to_use": "nbc",
  "netAppVersion": "1.0",
  "validate": true,
  "verify": true,
  "vnv_test_id": 89901102,
  "created_at": "2012-10-12T17:04:47Z",
  "updated_at": "2012-10-12T17:12:24Z"
  "netapp_descriptor": 1
}
```

Get V&V test results

Retrieve the test results from V&V Platform on a requested test.

GET - “/netapp/get-test-results/:id”

Headers:

- Content-Type: application/json
- Authorization: Bearer $oidc_access_token

Attribute	Type	Required	Description
id	Integer	Yes	The ID of the Network App Test.

Returns:

- 200 OK – On success

Example output:

```
{
  "results": [
```

```
    {
      "_id": "63064ed03b513ec068aae027",
      "done": "true",
      "id": pk,
      "netAppId": "test02",
      "netAppVersion": "1.0",
      "results": [
        {
          "date": "2022-09-18T18:55:16.329Z",
          "details": "None",
          "errorCode": "0",
          "errorDescription": "None",
          "failed": "false",
          "netAppId": "test02",
          "netAppVersion": "1.0",
          "type": "verification"
        },
        {
          "date": "2022-09-18T18:55:25.21Z",
          "details": "None",
          "errorCode": "500",
          "errorDescription": "error un-marshaling the Network App
Descriptor: error converting YAML to JSON: yaml: control characters are
not allowed",
          "failed": "true",
          "netAppId": "test02",
          "netAppVersion": "1.0",
          "type": "validation"
         }
      ],
      "validate": "true",
      "verify": "true"
    }
  ]
}
```


Network Service API:

List Network Services

Get a list of all visible Network Services for the authenticated user.

GET - "/ns/list/"

Headers:

- Content-Type: application/json
- Authorization: Bearer $oidc_access_token

Returns:

- 200 OK – On success

Example output:

```
[
  {
    "id": 1,
    "ns_id": "demo_ns_nsd",
    "public": false,
    "latest_version": "1.0"
  }
]
```

Show Network Service Detailed View

Get a specific Network Service's detailed fields.

GET - "/ns/detail/:id"

Headers:

- Content-Type: application/json
- Authorization: Bearer $oidc_access_token

Attribute	Type	Required	Description
id	Integer	Yes	The id of the Network Service.

Returns:

- 200 OK – On success
- 404 Network Service not found - if the Network Service does not exist or cannot be accessed by the requester.

Example output:

```
{
  "id": 1,
  "ns_id": "demo_ns_nsd",
  "public": false,
  "latest_version": "1.0",
  "descriptors": [
    {
      "id": 1,
      "data_formated": {
        "nsd": {
          "nsd": [
            {
```

```
              "description": "Network Service k8s for Network App",
              "designer": "Axon",
              "id": "demo_ns_nsd",
              "name": "demo_ns_nsd",
              "version": 1,
              "virtual-link-desc": [
                {
                  "id": "ns_virtual_link",
                  "mgmt-network": "true",
                  "vim-network-name": "mgmtnet"
                }
              ],
              "vnfd-id": [
                "demo_vnf_vnfd"
              ]
            }
          ]
        }
      },
      "data": "REDACTED",
      "commit": "0e1d0aae24b59013fd16520bd4a43bb142e05ec3",
      "version": "1.0",
      "created_at": "2022-10-25T18:20:41.788315Z",
      "updated_at": "2022-10-25T18:20:41.788341Z",
      "ns": 1
    }
  ]
}
```


Create a Network Service

Create a new Network Service. Available only for users who can create Network Services.

POST - "/ns/create/"

Headers:

- Content-Type: application/json
- Authorization: Bearer $oidc_access_token

Attribute	Type	Required	Description
name	String	Yes	The name of the Network Service.
description	String	No	The description of the Network Service.

public	Boolean	No	Specifies if the new Network Service will be publicly available or not. Default value is “false”.

Returns:

- 200 OK – On success

Example output:

```
{
  "id": 1,
  "name": "example-ns",
  "description": "NS description text",
  "public": false,
  "name": "example-ns",
  "created_at": "2012-10-12T17:04:47Z",
  "updated_at": "2012-10-12T17:04:47Z"
}
```

Update a Network Service

Create a new Network Service.

PUT - “/ns/update/:id”

Headers:

- Content-Type: application/json
- Authorization: Bearer $oidc_access_token

Attribute	Type	Required	Description
id	Integer	Yes	The ID of the Network Service.
description	String	No	The description of the Network Service.
public	Boolean	No	Specifies if the new Network Service will be publicly available or not. Default value is “false”.

Returns:

- 200 OK – On success

Example output:

```
{
  "id": 1,
  "name": "example-ns",
  "description": "Network Service description text",
  "public": false,
  "name": "example-ns",
  "created_at": "2012-10-12T17:04:47Z",
  "updated_at": "2012-10-12T17:12:24Z"
}
```

Delete a Network Service

Delete a new Network Service. Available only for users who can delete the specified Network Service.

DELETE - "/ns/delete/:id"

Headers:

- Content-Type: application/json
- Authorization: Bearer $oidc_access_token

Attribute	Type	Required	Description
id	Integer	Yes	The ID of the Network Service.

Returns:

- 200 OK – On success

Fork Network Service

Create a new Network Service being the exact duplicate of the selected Network Service under a specified existing Network App. The Network App must be owned by the initiator user.

GET - "/ns/fork/:id"

Headers:

- Content-Type: application/json
- Authorization: Bearer $oidc_access_token

Attribute	Type	Required	Description
id	Integer	Yes	The ID of the Network Service.
netapp_id	Integer	Yes	The ID of the target Network App Service, on which we want to fork the Network Service.

Returns:

- 200 OK – On success

Virtual Network Function API:

List VNFs

Get a list of all visible VNF s for the authenticated user.

GET - "/vnf/list/"

Headers:

- Content-Type: application/json
- Authorization: Bearer $oidc_access_token

Returns:

- 200 OK – On success

Example output:

```
[
  {
    "id": 1,
    "vnf_id": "demo_vnf_vnfd",
    "public": false,
    "latest_version": "1.0"
  }
]
```

Show VNF Detailed View

Get a specific VNF's detailed fields.

GET - "/vnf/detail/:id"

Headers:

- Content-Type: application/json
- Authorization: Bearer $oidc_access_token

Attribute	Type	Required	Description
id	Integer	Yes	The id of the VNF.

Returns:

- 200 OK – On success
- 404 VNF not found - if the VNF does not exist or cannot be accessed by the requester.

Example output:


```
{
  "id": 1,
  "vnf_id": "demo_vnf_vnfd",
  "public": false,
  "latest_version": "1.0",
  "descriptors": [
    {
      "id": 1,
      "data_formated": {
        "vnfd": {
          "description": "Virtual Network Function for Network App",
          "ext-cpd": [
            {
              "id": "vnf-connection-point"
            }
          ],
          "id": "demo_vnf_vnfd",
          "mgmt-cp": "vnf-connection-point",
          "product-name": "demo_vnf_vnfd",
          "provider": "Axon",
          "vdu": [
            {
              "name": "demo_vdu",
              "helm-chart":
"https://registry_url/chartrepo/demo_vdu/demo_vdu:0.1.0"
            }
          ],
          "version": 1
        }
      },
      "data": "REDACTED",
      "commit": "0e1d0aae24b59013fd16520bd4a43bb142e05ec3",
      "version": "1.0",
```

```
      "created_at": "2022-10-25T18:20:41.773368Z",
      "updated_at": "2022-10-25T18:20:41.773391Z",
      "vnf": 1
    }
  ]
}
```


Create a VNF

Create a new VNF. Available only for users who can create VNFs.

POST - "/vnf/create/"

Headers:

- Content-Type: application/json
- Authorization: Bearer $oidc_access_token

Attribute	Type	Required	Description
name	String	Yes	The name of the VNF.
description	String	No	The description of the VNF.
public	Boolean	No	Specifies if the new VNF will be publicly available or not. Default value is "false".

Returns:

- 200 OK – On success

Example output:


```
{
  "id": 1,
  "name": "example-vnf",
  "description": "VNF description text",
  "public": false,
  "name": "example-vnf",
  "created_at": "2012-10-12T17:04:47Z",
  "updated_at": "2012-10-12T17:12:24Z"
}
```

Update a VNF

Create a new VNF.

PUT - "/vnf/update/:id"

Headers:

- Content-Type: application/json
- Authorization: Bearer $oidc_access_token

Attribute	Type	Required	Description
id	Integer	Yes	The ID of the VNF.
description	String	No	The description of the VNF.
public	Boolean	No	Specifies if the new VNF will be publicly available or not. Default value is "false".

Returns:

- 200 OK – On success

Example output:

```
{
  "id": 1,
  "name": "example-vnf",
  "description": "VNF description text",
  "public": false,
  "name": "example-vnf",
  "created_at": "2012-10-12T17:04:47Z",
  "updated_at": "2012-10-12T17:12:24Z"
}
```

Delete a VNF

Delete a new VNF. Available only for users who can delete the specified VNF.

DELETE - "/vnf/delete/:id"

Headers:

- Content-Type: application/json
- Authorization: Bearer $oidc_access_token

Attribute	Type	Required	Description
id	Integer	Yes	The ID of the VNF.

Returns:

- 200 OK – On success

Fork VNF

Create a new VNF being the exact duplicate of the selected VNF under a specified existing Network App. The Network App must be owned by the initiator user.

GET - "/vnf/fork/:id"

Headers:

- Content-Type: application/json
- Authorization: Bearer $oidc_access_token

Attribute	Type	Required	Description
id	Integer	Yes	The ID of the VNF.
netapp_id	Integer	Yes	The ID of the target Network App Service, on which we want to fork the VNF.

Returns:

- 200 OK – On success

Virtual Deployment Unit API:

List VDUs

Get a list of all visible VDUs s for the authenticated user.

GET - "/vdu/list/"

Headers:

- Content-Type: application/json
- Authorization: Bearer $oidc_access_token

Returns:

- 200 OK – On success

Example output:

```
[
  {
    "id": 3,
    "registry_url": "https://c-registry.s5g.gos.y-
cloud.eu/harbor/projects/85/repositories",
    "name": "demo-vdu",
    "description": "a test vdu",
    "public": false,
    "image_type": "helm",
    "repository_url": "git@repo_url:demo-vdu/demo-vdu.git",
    "identity_group_id": "6f7c9d4b-db0b-4f91-b650-007c585822ee",
    "user": 1
  }
]
```

Show VDU Detailed View

Get a specific VDU's detailed fields.

GET - "/vdu/detail/:id"

Headers:

- Content-Type: application/json
- Authorization: Bearer $oidc_access_token

Attribute	Type	Required	Description
id	Integer	Yes	The id of the VDU.

Returns:

- 200 OK – On success
- 404 VDU not found - if the VDU does not exist or cannot be accessed by the requester.

Example output:


```
{
  "id": 3,
  "registry_url": "https://c-registry.s5g.gos.y-
cloud.eu/harbor/projects/85/repositories",
  "name": "demo-vdu",
  "description": "a test vdu",
  "public": false,
  "image_type": "helm",
  "repository_url": "git@repo_url:demo-vdu/demo-vdu.git",
  "identity_group_id": "6f7c9d4b-db0b-4f91-b650-007c585822ee",
  "user": 1
}
```


Create a VDU

Create a new VDU. Available only for users who can create VDUs.

POST - "/vdu/create/"

Headers:

- Content-Type: application/json
- Authorization: Bearer $oidc_access_token

Attribute	Type	Required	Description
name	String	Yes	The name of the VDU.
description	String	No	The description of the VDU.
public	Boolean	No	Specifies if the new VDU will be publicly available or not. Default value is "false".
image_type	String	Yes	Image type can be either "docker" or "helm".

Returns:

- 200 OK – On success

Example output:

```
{
  "id": 1,
  "name": "example-vdu",
  "description": "VDU description text",
  "public": false,
  "name": "example-vdu",
  "created_at": "2012-10-12T17:04:47Z",
  "updated_at": "2012-10-12T17:12:24Z",
  "registry_url": "https://registry_url/harbor/projects/86/repositories",
  "repository_url": "git@repo_url:example-vdu/example-vdu.git",
}
```


Update a VDU

Create a new VDU.

PUT - "/vdu/update/:id"

Headers:

- Content-Type: application/json
- Authorization: Bearer $oidc_access_token

Attribute	Type	Required	Description
id	Integer	Yes	The ID of the VDU.
description	String	No	The description of the VDU.
public	Boolean	No	Specifies if the new VDU will be publicly available or not. Default value is "false".

Returns:

- 200 OK – On success

Example output:


```
{
  "id": 1,
  "name": "example-vdu",
  "description": "VDU description text",
  "public": false,
  "name": "example-vdu",
```

```
  "created_at": "2022-10-12T17:04:47Z",
  "updated_at": "2022-10-12T17:12:24Z",
  "registry_url": "https://registry_url/harbor/projects/86/repositories",
  "repository_url": "git@repo_url:example-vdu/example-vdu.git",
}
```


Delete a VDU

Delete a new VDU. Available only for users who can delete the specified VDU.

DELETE - "/vdu/delete/:id"

Headers:

- Content-Type: application/json
- Authorization: Bearer $oidc_access_token

Attribute	Type	Required	Description
id	Integer	Yes	The ID of the VDU.

Returns:

- 200 OK – On success