D4.1 First report on 5G-IANA nApp Toolkit and VNFs Repository development

Dissemination level:	Public (PU)
Work package:	WP4
Task:	T4.1
Deliverable lead:	NXW
Version:	1.5
Submission date:	03/03/2023
Due date:	28/02/2023
Partners:	

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101016427

5g-iana.eu

Authors

Authors		
Name	**Organisation**	**Email**
Matteo Andolfi	NXW	m.andolfi@nextworks.it
Manuel Fuentes	5COMM	manuel.fuentes@fivecomm.eu
Miriam Ortiz	5COMM	miriam.ortiz@fivecomm.eu
David Martín-Sacristán	5COMM	david.martin-sacristan@fivecomm.eu
Thanos Xirofotos	UBI	txirofotos@ubitech.eu
Dimitris Klonidis	UBI	dklonidis@ubitech.eu
Marios Zinonos	HIT	m.zinonos@hit-innovations.com

Control sheet

Version History			
Version	**Date**	**Modified by**	**Summary of changes**
v0.1	10/11/2022	Matteo Andolfi (NXW)	ToC
v0.2	17/01/2023	Marios Zinonos (HIT)	First draft for Chapter 5
v0.3	06/02/2023	NXW	First draft for Chapter 4
V0.4	10/02/2023	Marios Zinonos (HIT)	Second draft for Chapter 5
v0.5	10/02/2023	Matteo Andolfi (NXW), UBI	Formatted all the contributions that the partners provided in the previous version of the document, UBI provided the first draft for Chapter 3
v0.6	10/02/2023	Marios Zinonos (HIT)	Review and changes on chapter 5
v0.8	16/02/2023	Matteo Andolfi (NXW)	First review and formats
V0.9	20/02/2023	Edoardo Bonetto (LINKS)	First review
V1.0	22/02/2023	Matteo Andolfi (NXW)	Revision and contributions to all sections
v1.1	23/02/2023	Marios Zinonos (HIT)	Editorial Review and comments
v1.4	24/02/2024	Matteo Andolfi (NXW)	Final Review
v1.5	28/02/2023	Matteo Andolfi (NXW)	Additional revisions

Peer review		
	Reviewer name	**Date**
Reviewer 1	Edoardo Bonetto (LINKS)	24/02/2023
Reviewer 2	Markus Wimmer (NOKIA)	15/02/2023
Reviewer 3	Dimitris Klonidis (UBI)	28/02/2023
Reviewer 4	Eirini Liotou (ICCS)	02/03/2023

Legal disclaimer

TABLE OF CONTENTS

TABLE OF CONTENTS 4

1. INTRODUCTION 11

1.1. Purpose of the deliverable 11
1.2. Intended audience 11

2. NAPP TOOLKIT 12

2.1. High-level architecture 14
2.1.1. Vertical App Composition & Customization 14
2.1.2. NApp Catalogue 15
2.2. Composition of nApps and Vertical Services 15
2.2.1. Atomic components chaining and composition of a nApp 17
2.2.2. Procedures for nApps DevOps 17
2.2.3. DevOps Pipeline 20
2.3. Release roadmap 23

3. VERTICAL SERVICE COMPOSITION AND CUSTOMIZATION 24

3.1. Vertical Service Composition 24
3.2. Vertical Service QoS Parameter Definition and Editing 25
3.3. Policy Management 26
3.4. Software Design 27
3.5. Workflows and APIs 27
3.5.1. NApp Graph REST Endpoints 29
3.5.2. Policies REST Endpoints 29
3.5.3. AF/NF Onboarding REST Endpoint 30

4. NAPP CATALOGUE 32

4.1. NApp Template & Information model 32
4.1.1. Extensions from baseline information model 34
4.2. Software Design 39
4.3. Workflows and APIs 40
4.3.1. On-boarding 40
4.3.2. Query 42
4.3.3. getBlueprint 42
4.3.4. GetAllnAppPackages 43
4.3.5. Delete 43

5. APPLICATION AND NETWORK FUNCTIONS DEVELOPMENT 44
6. CONCLUSION 50
REFERENCES 51
ANNEX – NAPP TEMPLATE 52

List of Figures

Figure 1 5G-IANA AOEP overall architecture 12
Figure 2 nApp Toolkit: High Level Architecture 13
Figure 3 nApp package 16
Figure 4 Atomic Component Structure 19
Figure 5 An example of Dockerfile for AFs/NFs 20
Figure 6 DevOps pipeline 21
Figure 7 High-level workflow for onboarding and deploying a nApp 22
Figure 8 Vertical App Composition & Customization 24
Figure 9 Vertical Service Composition and Customization workflow 28
Figure 10 On-boarding workflow 41

List of Tables

Table 1 nApp Graph REST Endpoints 29
Table 2 Policies REST Endpoints 29
Table 3 AF/NF Onboarding REST Endpoints 30
Table 4 nApp Information Model 32
Table 5 ComponentNode information model 34
Table 6 LinkNode information model 34
Table 7 Link information model 35
Table 8 AtomicComponent information model 35
Table 9 ExposedInterface information model 36
Table 10 RequiredInterface information model 37
Table 11 Requirement information model 37
Table 12 HealthCheck information model 37
Table 13 FiveGServiceSpec information model 38
Table 14 SoftwareLicense information model 38
Table 15 RequiredEquipment information model 39
Table 16 On-board Endpoint 40
Table 17 Query Endpoint 42
Table 18 getBlueprint Endpoint 43
Table 19 GetAll nApp packages Endpoint 43
Table 20 Delete Endpoint 43
Table 21: Baseline AFs offered by 5G-IANA 44
Table 22: Baseline NFs offered by 5G-IANA 47
Table 23: 5G or ITS communications related NFs offered by 5G-IANA 48

ABBREVIATIONS

Abbreviation	Definition
5G-IANA	5G Intelligent Automotive Network Applications
5G-PPP	5G Public Private Partnership
AI	Artificial Intelligence
AF	Application Function
AOEP	Automotive Open Experimental Platform
API	Application Programming Interface
CE	(Gitlab) Community Edition
CI	Continuous Integration
CD	Continuous Delivery
CPU	Central Processing Unit
DL	Downlink
DML	Decentralized Machine Learning
DoA	Description of Action
E2E	End-to-End
EC	European Commission
FOV	Field-of-View
GPS	Global Positioning System
HMD	Head Mounted Display
IPC	Inter-Process Communication
IPS/IDS	Intrusion Detection Systems/Intrusion Prevention Systems
KPI	Key Performance Indicator
LDM	Local Dynamic Map
MANO	Management and Orchestration
MEC	Multi-access Edge Computing

ML	Machine Learning
MOS	Mean Opinion Score
NF	Network Function
NW	Network
OS	Operating System
PU	Public
OBU	On-Board Unit
QoE	Quality of Experience
QoS	Quality of Service
RAN	Radio Access Network
RTT	Round-Trip-Time
SME	Small and Medium Sized Enterprise(s)
UC	Use Case
UDP	User Datagram Protocol
UE	User Equipment
UHD	Ultra-High Definition
UL	Uplink
UPF	User Plane Function
VAO	Vertical Application Orchestrator
VBT	Virtual Bus Tour
VNF	Virtual Network Function
WP	Work Package

Executive Summary

This deliverable has the objective to provide a first report regarding the implementation and the activities performed in Work Package (WP) 4 “5G-IANA nApps toolkit development”. These activities include the design and the development of several components which compose the nApp toolkit. The nApp toolkit offers functionalities to the developers to create, to store and to manage network applications (nApps) inside the 5G-IANA Automotive Open Experimental Platform (AOEP) using a Graphical User Interface (GUI).

A network application (nApp) is a composition of atomic components, which are Application and Network functions (AFs/NFs). A component is a virtualizable function that can be deployed in a container. The use of a GUI helps the nApp developer to define the requirements and to compose a standalone network application using the available atomic components which are stored in the toolkit. The functionalities of the nApp toolkit can be divided in two main functional components: the Vertical App Composition & Customization and the nApp Catalogue. The Vertical App Composition & Customization offers the functionalities to graphically compose a network application, while the nApp Catalogue is used to store and manage several network applications which a nApp developer can use.

Moreover, this deliverable offers a starting point for defining the components used in the 5G-IANA platform, how they should be composed and how they should be used. A CI/CD pipeline is described, which provides the developers with the ability to compile the various components to obtain docker images to be pushed on the 5G-IANA centralized registry. These atomic components can be linked together with the nApp toolkit, to obtain standalone network applications which can be stored and managed by the nApp catalogue to be reused or to be shared among different stakeholders. There are several atomic components which will be made available in the nApp toolkit and will be composed in different network applications to be used as nApp starter kits from which a stakeholder can start developing his/her vertical service.

1. INTRODUCTION

1.1. Purpose of the deliverable

The purpose of this deliverable is to report on the implementation of the 5G-IANA nApp Toolkit and the VNFs repository.

In Section 2 the nApp toolkit is presented, with the functional components composing it.

Section 3 and Section 4 present the main components of the toolkit, namely, the Vertical Service Composition and Customization and the nApp catalogue respectively.

Finally, in Section 5 the atomic components are presented which will be available in the AOEP platform to be used by the nApp developers.

1.2. Intended audience

The dissemination level of this deliverable is "public" (PU)[1]. It is primarily aimed to be the reference document to be used by the 5G-IANA Consortium Members during the development and integration phases of the atomic components and nApps of the 5G-IANA project. Moreover, it will be a useful document ("manual") for the third-parties, i.e., SMEs, who will experiment on the 5G-IANA platform during the project's Open Calls.

[1] The dissemination level needs to change from Confidential to Public, as requested by the project interim review. Here, it is already declared as Public, while an amendment will specify this.

2. NAPP TOOLKIT

Orchestration techniques have been developed to meet the requirements of specific industries, specifically to allow for various services to be provided on a 5G infrastructure that can handle different types of data. These techniques are designed to simplify the process of providing these services by using information models and APIs that are oriented towards specific industries, as reported in Section 3.1 of [1]. The development of 5G technology, particularly in relation to network slicing, has made service deployment scenarios more complex, highlighting the need for simplified information to be provided to industries so that they can request services and applications autonomously.

For this reason, the 5G-IANA's nApp toolkit enables developers to create brand-new network applications and vertical automotive services which can exploit 5G services with specific requirements and functionalities, and which can be deployed over a 5G infrastructure. The nApp toolkit is one of the main functional blocks of the 5G-IANA architecture as depicted in Figure 1.

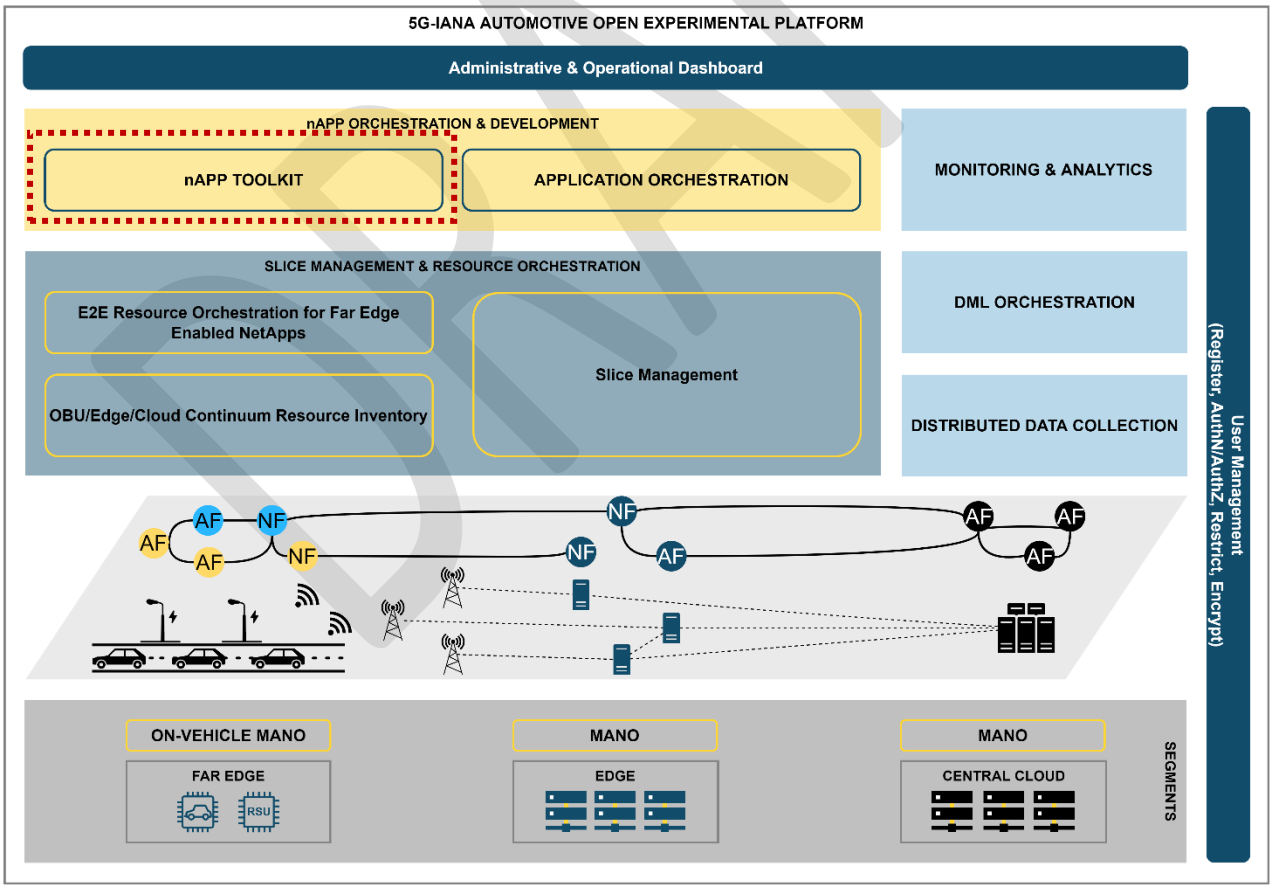

Figure 1 5G-IANA AOEP overall architecture

The goal of the nApp Toolkit, a part of the 5G-IANA nApp Orchestration and Development framework, is to make it easier to chain together and customize 5G-ready vertical services from vertical service providers with the functionalities provided by the Vertical App Composition & Customization (as is depicted in Figure 2) as well as the functionalities provided by the nApp catalogue which enables the on-boarding and updating of nApps Packages and related components from nApps and AFs/NFs Providers.

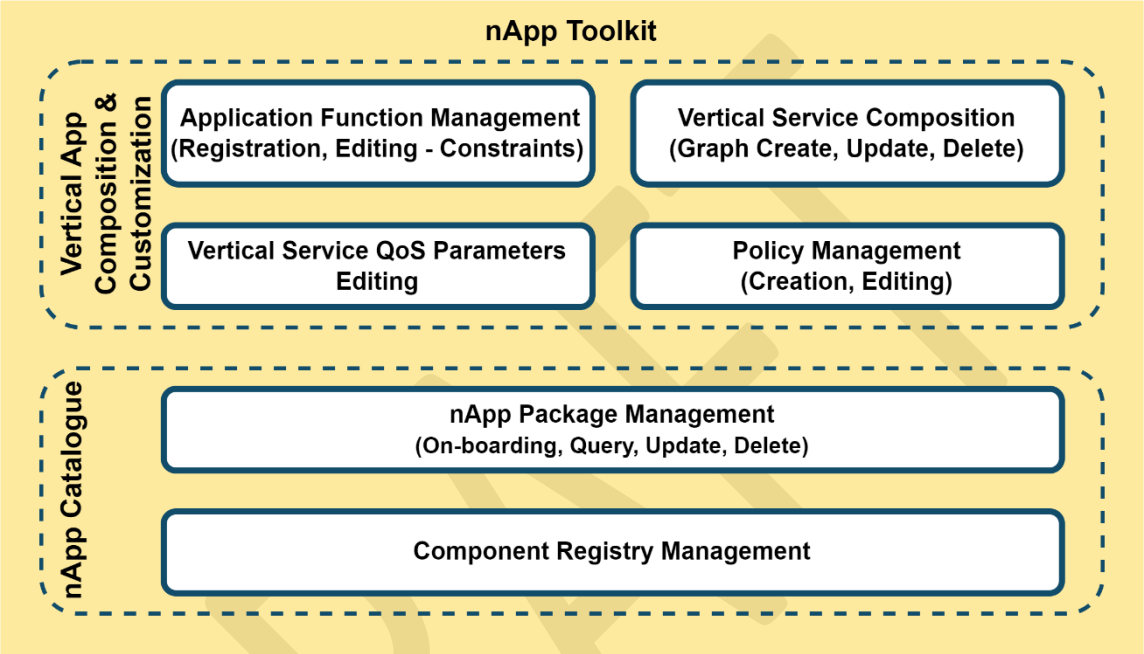

Figure 2 nApp Toolkit: High Level Architecture

The Toolkit communicates from one side with the Application Orchestrator which manages the deployment requests. On the other side, the nApp toolkit exposes its services directly to the nApp and Vertical service developers providing features to:

- register application and network functions as atomic components,
- compose network applications and vertical services in a graphical, intuitive, and simple way,
- onboard nApps and vertical services for future use.

In addition, it provides several starting points, called “nApp starter kits”, to help developers in the development of their applications.

2.1. High-level architecture

Figure 2 shows the nApp Toolkit's internal functional architecture as well as the primary functional building blocks and associated features. During the complete Vertical Service and nApps lifetime, the nApp Toolkit is in charge of creating and exposing the necessary set of features that are connected to the design and modeling stages. In [1] it is explained where this functional block is placed within the complete 5G-IANA architecture.

The main functional blocks in the nApp toolkit are the Vertical App Composition & Customization and the nApp Catalogue.

2.1.1. Vertical App Composition & Customization

The Vertical App Composition & Customization assists the service providers wrapping cloud native nApps in a proper format, so as to be publishable in the nApp Catalogue. This registration mechanism, called “onboarding of a nApp” is being provided through a dedicated user interface that offers several features, to make the overall process less error-prone. Vertical App Composition & Customization provides a graphical way to create a nApp by specifically selecting atomic components and bundling them together in the form of a graph (see Section 3.1). Complementary, it enables declaring cloud-related constraints and non-cloud-related metadata (network requirements - see Section 3.2) that play a significant role in the slice negotiation process.

Another aspect of the Vertical App Composition & Customization is the enabling of Policy authoring. Every nApp that is deployed can be subject to runtime changes/reconfiguration. This reconfiguration aims at the satisfaction of a set of business goals that are bundled in the form of a Service Level Agreement. The Policies’ specific module (also referred to as Policy Management) will be responsible for authoring these instructions in a formal rules format.

For the sake of reader’ clarity, the functionalities the vertical App Composition and Customization provides are:

- the Application Function Management which is the onboarding of atomic components and the Vertical Service Composition which are covered on Section 3.1,
- The vertical Service QoS Parameter Definition and Editing which is covered on Section 3.2,
- The policy management which is covered on Section 3.3.

2.1.2. NApp Catalogue

The nApp Catalogue stores and manages the atomic components and the nApps. It is composed of the two functional blocks described below:

- nApp Package Management

The functionality of nApp Package Management is implemented by the nApp Catalogue component which implements all the functionalities for managing the nApp packages. A description is available in Section 4.

- Component Registry Management

The functionality of Component Registry Management is implemented by a Centralized Registry. This component is the repository that stores and makes the AF/NF images available in 5G-IANA. This AFs/NFs registry will be a docker registry installed and available in the testbed site and accessible via VPN to the nApp developer, Gitlab platform, and Composer platform to upload and retrieve all the images that are needed to compose the network applications. These images are made available for use, according to their license agreements, and can be queried by the composer, so that to be used to graphically compose a network application.

The nApp developer can query directly the AFs/NFs registry, using the Docker Registry API [3]. The Vertical Service Composition and Customization provides a unique entry point to the Service provider. The functionalities supported by this functional block are reported in [1]. More details about this component are available at Section 4.

2.2. Composition of nApps and Vertical Services

From a network application developer's point of view, a nApp is a composition of atomic components that can communicate with each other and can be instantiated separately with different requirements. This definition leads to a scalable vertical service deployment which takes into consideration the availability of the processing resources in the Edge/Cloud server(s), slice resources, and the nApp developer requirements.

A nApp is defined with a **nApp package**, as described in Figure 3.

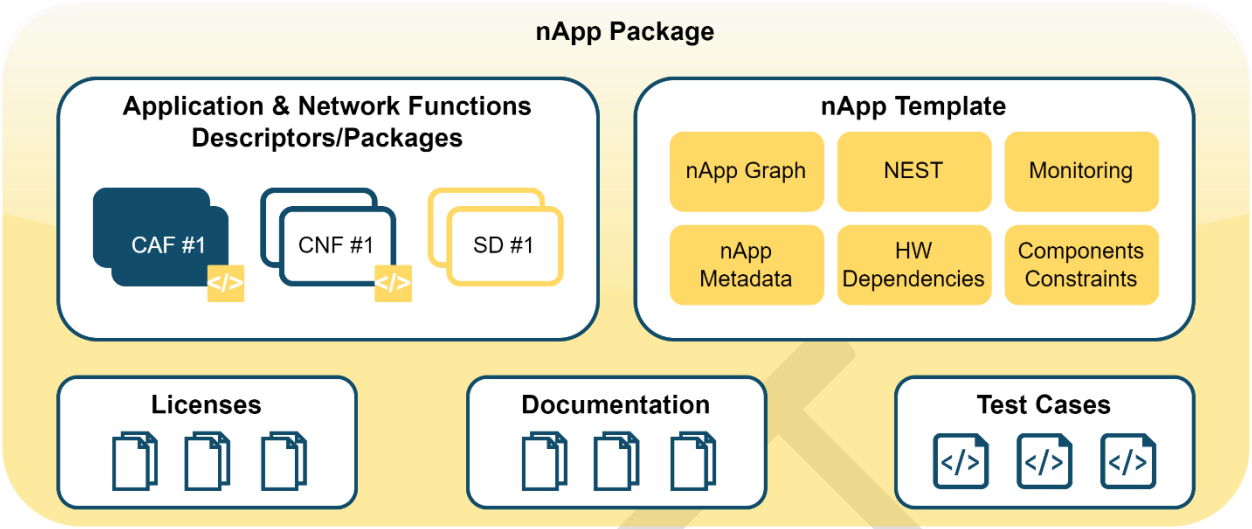

Figure 3 nApp package

A **nApp package** is composed of several items:

- A set of **Application & Network Functions Descriptors/Packages** – These describe the atomic components used in the network application, and which are defined in 4.1.1.
- A **nApp Template** – A description of the network application, with the information model defined in 4.1.
- A **Licenses** folder – which contains the licenses of the atomic components used in the nApp package without a specific format.
- A **Documentation** folder – which contains the documents of the atomic components used in the nApp package.
- A **Test Cases** folder – which contains the test cases of the atomic components, with a specific format, like for example the OpenAPI Specification v3.1 [9].

Each developer can link together several atomic components, specifying the exposed interfaces used to communicate. In the same manner, multiple nApps can be chained together to create a standalone end-to-end vertical service.

An atomic component used in 5G-IANA can be:

- an Application Function (AF): a component that implements the logic of a service,
- a Network Function (NF): a component used for communication and networking tasks.

2.2.1. Atomic components chaining and composition of a nApp

In the terminology of 5G-IANA, atomic components are virtualizable functions that are deployable and operating on top of programmable network infrastructures. They are distinct into two categories, Application and Network functions.

Application functions are composed of one or typically many software components. These application components in containerized deployments are becoming the Application Functions. These AFs are typically developed by vertical service developers for particular vertical use cases and linked together to provide the required service functionality. Therefore, the AFs have practically the same notion as the application components.

Network functions is an umbrella term of the vastly used term VNFs. There are two types of VNFs:

1. The Network Service and Resource-facing Virtualizable Functions, denoted commonly in standards as Virtual Network Functions or VNFs, which refer to the standardised functions provided by the network operator and infrastructure owner for fulfilling the networking and connectivity services (e.g.: User Plane Function (UPF), Access and Mobility Function (AMF), Session Management Functions (SMF), etc.).
2. The Value-added (middleware) VNFs or Non-standardised VNFs, which refer to network-level functions that are added in support of the networking operations and services either to satisfy specific end user network function requests (i.e., intermediately on demand) or as a response to network changes (i.e., automatically). Examples may include encryption and decryption VNFs for secure communications, video processing VNFs, and so on.

In 5G-IANA, the Network Functions are realizing the network and communication requirements of an application and are all belonging to the second type. So, these virtualizable functions are eligible for deployment on top of programmable infrastructures in an identical way as the Application Functions.

Inside the scope of 5G-IANA, a nApp is a mixture of AFs and NFs linked together into a form of graph. Furthermore, a nApp is a cloud native application that uses several AFs/NFs as part of its data plane in a seamless way, encapsulating this way the layer-7 business logic in container format (AFs) accompanied by network requirements that may need to be satisfied (NFs). Once the nApp is deployed, the linked and deployed AFs/NFs are becoming the nApp components.

2.2.2. Procedures for nApps DevOps

The 5G-IANA project uses Gitlab as a DevOps platform. GitLab is an open-source code repository and collaborative software development platform which offers a location for online code storage and capabilities for issue tracking and CI/CD, and it is used by the developer to store and upload the atomic components and the network applications inside the nApp toolkit.

A nApp developer will use several types of components to build network applications. The components' types that can be used in the platform are the following, which can be mapped with the licenses reported in [2]:

- Open-Source atomic components (Open-source). These components are open-source, and thus, the source code of each component is available on the 5G-IANA Gitlab platform and can be used free. A developer can access the source code and some Continuous Integration (CI) metrics, like compilation logs, the number of bugs found analysing the code, etc. The packaged component then, is stored as a Docker image in a shared registry to use. Other developers can get the source code of this type of components and change it to match their requirements. For example, an external component developer can use an open-source component, changing the input and output interfaces to match the other components interfaces in the nApp.
- Protected atomic components (Open-use-NDA, Open-use-platform, Open-use-partners). These components are not open-source. Instead, the binary of each component, built as a Docker image, is available to be used by the stakeholders with licenses limitations. An Open-use-NDA component is used after the interested party and the owner of the AF/NF sign an NDA, and Open-use-platform component can be used only within the 5G-IANA AOEP, and an Open-use-partners component can be used only from the 5G-IANA consortium partners. A nApp developer can use these components as-is and for this reason, he/she has to match the input and output interfaces of each component.
- Private atomic components. These components are private and are not usable by other stakeholders.

The platform provides a way to unify the atomic components' development, in which every atomic component is developed to be used as a Docker image. With this methodology, the Gitlab platform can host every open-source project, developed with the most useful code language and paradigm. The CI pipeline will get a project and will build it using the Dockerfile provided in the root folder.

With this procedure it is possible to upload a stable version on the 5G-IANA shared registry that can be accessed by the Composer to retrieve the list of atomic components to be used for creating a nApp.

- A private atomic component's image can be uploaded on a private registry, using its Dockerfile, preventing the component to be used from external stakeholders

- The protected atomic components are a particular type of components. The source code can be committed on the 5G-IANA Version Control System as a private project, with an access control mechanism to hide source code from other developers. Otherwise, a component developer can push component's docker images on the 5G-IANA registry from a private repository.

In Figure 4 it is possible to view a generic code structure that is used by the 5G-IANA Gitlab CI pipeline. The Dockerfile must include several steps to be accepted and executed by the platform.

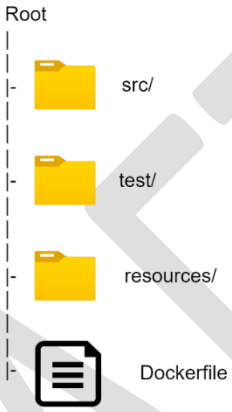

Figure 4 Atomic Component Structure

The steps, called stages, included in the Dockerfile are like the ones shown in Figure 5 and are described as:

- Build stage. Where the code is compiled with specific tools
- Analysis stage. Where the code is analyzed
- Package stage. Where the component is packaged and the entry points are defined
- Deploy stage. Where the container is finally uploaded to the registry

The required steps are the Build and the Deploy stages. The Analysis and the Package stages are not mandatory but should be included to provide a stable and self-contained component.

```
#
# Build stage
#
FROM maven:3.6.0-jdk-11-slim AS build
COPY ./src /home/app/src
COPY ./pom.xml /home/app
RUN mvn -f /home/app/pom.xml clean package
#
# Analyis stage
#
FROM maven:3.6.0-jdk-11-slim AS build
COPY ./src /home/app/src
RUN mvn verify sonar:sonar -f /home/app/pom.xml
    -Dsonar.host.url=<ANALYSIS_URL>
    -Dsonar.login=<SONAR_LOGIN>
    -Dsonar.qualitygate.wait=true
    -Dsonar.projectKey=<SONAR_PROJECT_ID>
    -Dsonar.projectName=<SONAR_PROJECT_NAME>
    -Dsonar.java.source=1.11
#
# Package stage
#
FROM openjdk:11-jre-slim
COPY --from=build /home/app/target/*.jar /usr/local/lib/<AF/NF name>.jar
EXPOSE 8080
ENTRYPOINT ["java","-jar","/usr/local/lib/<AF/nf name>.jar"]
#
# Deploy stage
#
FROM docker-registry-dev.nextworks.it/mvn-jdk8-nextworks:1.0.0 AS deploy
COPY ./src /home/app/src
COPY ./pom.xml /home/app
RUN  export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64 && mvn -f /home/app/pom.xml deploy
```

Figure 5 An example of Dockerfile for AFs/NFs

2.2.3. DevOps Pipeline

As stated in the previous Section, every component/nApp that is open-source should be stored in the 5G-IANA GitLab platform. If the atomic components are not open-source, a developer can upload directly the images on the Centralized Registry, which has an access control mechanism to prevent from showing private images to other developers.

In addition, the composer has an access control mechanism to let the developers use only the intended atomic components. The private images uploaded in the centralized registry are retrieved by the composer only for specific developer profiles. In this way, protected and private atomic components can be stored in the nApp catalogue without being shared to other stakeholders.

The open-source components instead, are stored in the Gitlab platform to be available for use and deployment with the help of two different pipelines that are developed ad-hoc for the project. A DevOps pipeline is a collection of automated procedures and technologies that enables actors to collaborate effectively while creating and deploying code in a test bed.

Figure 6 graphically describes the whole DevOps pipeline for the 5G-IANA project.

As stated in 2.2.1, a nApp developer creates a network application, chaining together several Application and Network functions. After the composition is finished, the nApp is saved in the Vertical Service Composition

and Customization (described below, in Section 3) and the nApp template is committed on the Gitlab repository. This operation triggers a DevOps pipeline that reads the nApp Template, retrieves all the components from the testbed registry and onboards the nApp Template on the Toolkit located in the target testbed.

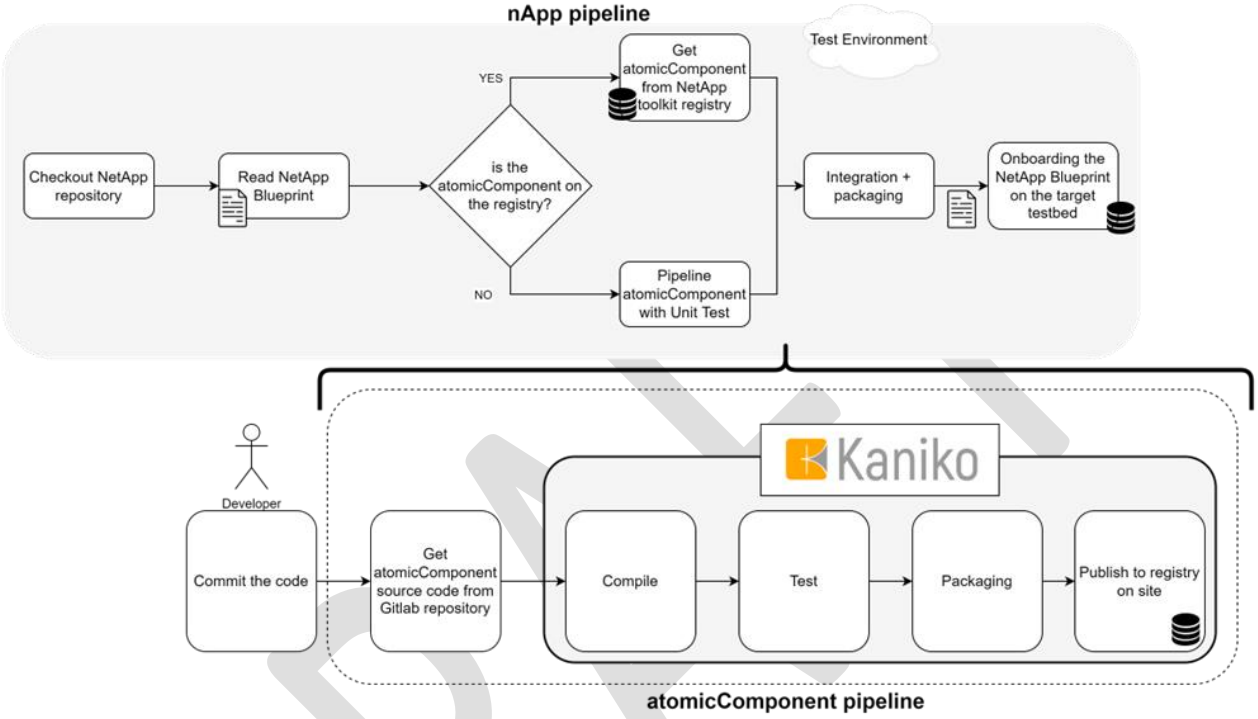

Figure 6 DevOps pipeline

If an atomic component in not present in the testbed registry, the DevOps pipeline will retrieve the source code of the component from its repository, compile it, test it, and publish the image to make it available for the next pipelines. These operations are made using Kaniko [4], a tool to build container images from a Dockerfile, inside a container or Kubernetes cluster. In this way an atomic component is built in a completely standalone container, without the possibility to interact with the 5G-IANA environment, which is secured from malicious actions and from unexpected behavior during the compiling and onboarding operation.

The whole workflow for onboarding a network application is depicted in Figure 7. The component developer develops the AFs and NFs and commits the source code on the Gitlab repository. This action triggers a CI/CD pipeline (Figure 6, atomicComponent pipeline) that compiles and analyzes the code and pushes the images on the Centralized Registry of the nApp catalogue with the other images already provided by the 5G-IANA nApp toolkit.

The nApp developer, from the composer GUI, queries the Centralized Registry to obtain the list of components available in the Registry with their documentation.

With the help of the documentation, and with the list of available components, a nApp developer creates a network application and stores it both on the local database of the composer (for future use) and on the Gitlab repository, that triggers the pipeline in Figure 6.

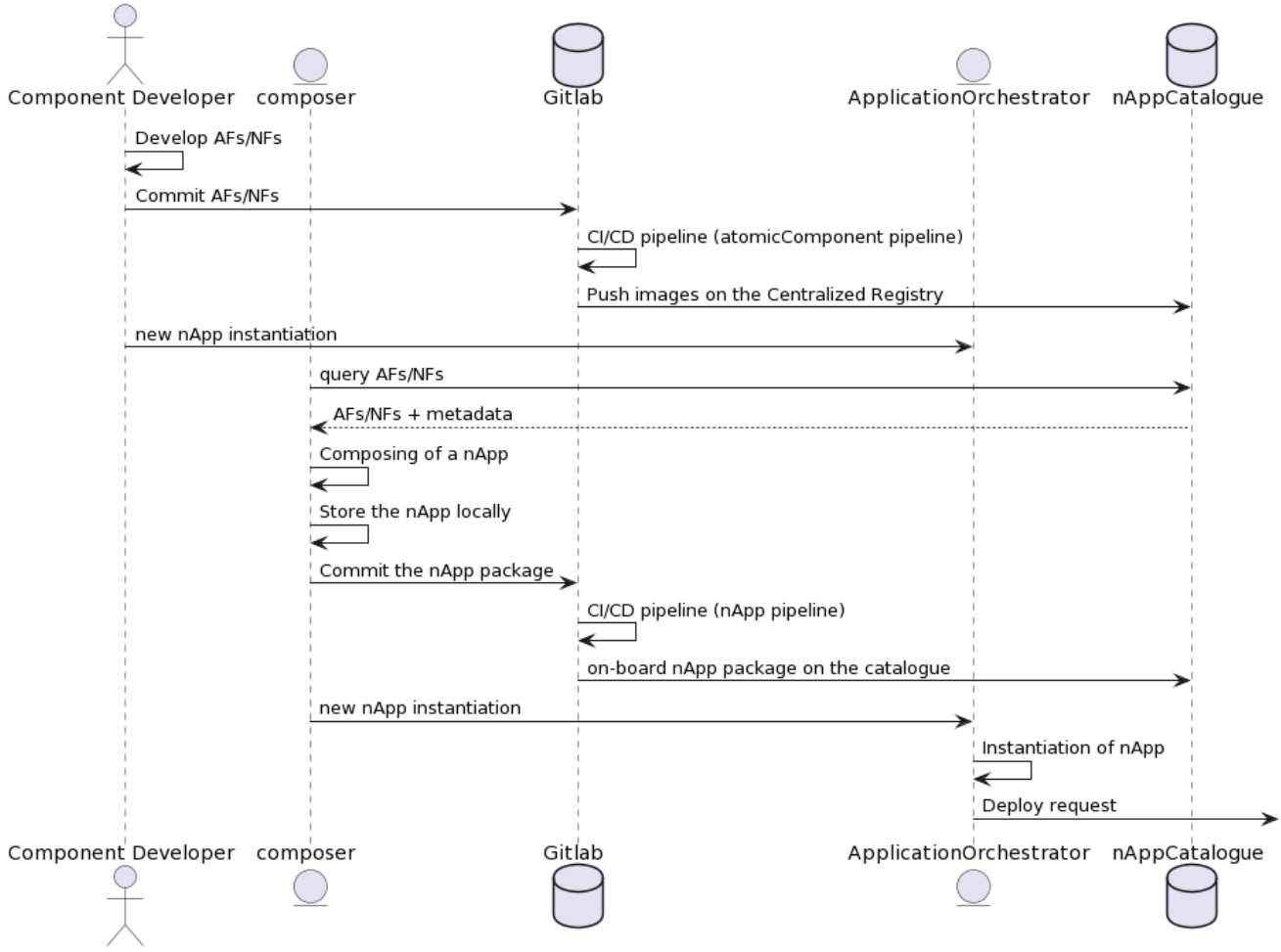

Figure 7 High-level workflow for onboarding and deploying a nApp

The network application is now available on the nApp catalogue, and it can be instantiated and deployed. From the composer GUI, the nApp developer requests a new instantiation, and after the requirements were specified for his/her use case, the application orchestrator requests deployment on the 5G-IANA testbed.

This high-level workflow is intended to be used with open-source atomic components, which can be committed on the Gitlab repository of 5G-IANA.

2.3. Release roadmap

The first release of the vertical composer and the catalogue has been tested and is going to be deployed on the NOKIA testbed. Future development, provided by M33, will involve the integration with the Gitlab repository to deliver the onboarding mechanism of a network application as a continuous flow of actions, from the composition of a network application to the deployment of the vertical service.

3. VERTICAL SERVICE COMPOSITION AND CUSTOMIZATION

The purpose of the Vertical Service Composition and Customization module is to assist and guide the composition and onboarding of a network application with specific methodology and targeted Graphical User Interfaces. To this end, it provides an interfacing layer to the end user (i.e., the vertical service owner and the nApp developer) for constructing, managing the deployable applications and their features, without interfering with the network level functions and processes managed by the involved telecom operators and infrastructure owners. All of the functionalities provided (depicted in Figure 8) by this block are analysed in the rest of this section.

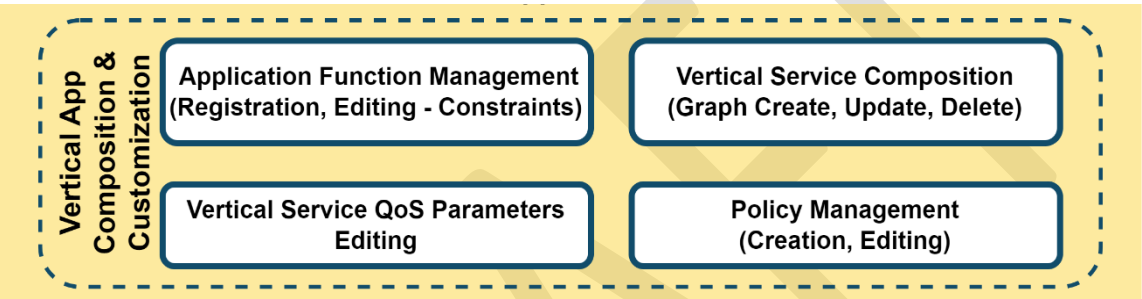

Figure 8 Vertical App Composition & Customization

3.1. Vertical Service Composition

The primary goal of the Vertical Service Composition is to provide the means for the authoring of abstracted representations of cloud-native applications that may be accompanied by several requirements in the form of constraints. To this end, the service providers wrap the microservices that implement the business logic of a service in a proper format, so as to be publishable in the nApp Catalogue. Each nApp consists of multiple containers (each container is an AF or an NF) that are chained in the form of a nApp graph. Every cloud-native component has to comply with a specific metamodel which is called the 12-factor metamodel [7]. 12-factor compliance provides a proper guarantee that a component will be able to be managed/orchestrated during its deployment. The registration process of a nApp component provides design-time validation as far as a component model is concerned. The same process shall guarantee that all cloud-native properties are maintained, specifically:

- metadata regarding minimum infrastructural requirements,
- metadata regarding deployment preferences,
- metadata regarding configuration parameters during component initialization,

- mutable configuration parameters during runtime,
- exposed and required interfaces,
- exposed metrics, link metrics, etc. [see section 4.1].

All the previous will be performed through a specific user interface which offers several design-time features, to make the overall process less error-sensitive.

3.2. Vertical Service QoS Parameter Definition and Editing

Vertical Service Composition and Customization offers a range of functions; one of these functionalities is the Vertical Service QoS Parameter Definition and Editing which allows service providers to create constraints that must be met to ensure that a nApp operates as intended. There are various types of constraints, and the aim of Vertical Composition and Customization is to define and formalize them. Typically, these constraints can be categorized as cloud-related or non-cloud-related.

- The cloud related constraints are split to deployment constraints and resource constraints**.**
    - Resource constraints may refer to amount of memory, CPUs, storage, and IO throughput required
    - Deployment constraints may refer to location requirements, device characteristics (i.e.: FPGA, GPU related).
- The non-cloud-related constraints are:
    - the network & operational constraints which refer to minimum or maximum thresholds from network and operational view (e.g.: e2e latency or deployment time) or desired QoS classifiers that are dependent on the performance of the programmable resources possessed.

The cloud-related constraints are authored through the Vertical Service Composition process described above. However, there are some non-cloud-related metadata that play a significant role in the slice negotiation process. These metadata are taken in consideration by the Slice Manager and Multi-Domain Orchestrator and refer to:

- the required slice capabilities (eMBB (enhanced Mobile Broadband), URLLC (Ultra Reliable Low Latency Communications), mMTC (massive Machine Type Communications))
- the QoS Class Identifier classification

- the guaranteed bit rate and
- the selection of specific location for the deployment (while the actual location of all the deployments will take place inside the domain of the two testbeds involved; these metadata will be used to point to specific OBU / RSU inside each testbed).

This type of constraint will be provided in a formal way through a specified editor. It should be noted that all the previous constraints are expressed in a "as much as possible" user friendly manner and without prior interaction with the Slicing Manager and Multi-Domain Orchestrator of the 5G-IANA Platform [1]. However, for the selection of specific location, an interaction with Resource Inventory module is required (part of Slicing Manager and Multi-Domain Orchestrator layer of the architecture). It is important to note here that since the constraints are declared without the prior interaction there might be the case that these constraints cannot be satisfied. All these situations are handled through the slice intent and slice reply exchange [1].

3.3. Policy Management

Policy Management is a functionality provided by the nApp Toolkit. In the frame of 5G-IANA every nApp that is deployed can be also included to runtime reconfiguration. Every nApp that is deployed can be subject to runtime changes/reconfiguration. In 5G-IANA context, a Policy is this reconfiguration that aims to the satisfaction of a set of business goals that are bundled in the form of a Service Level Agreement [1]. To satisfy these goals, these are "actions" that may need to intervene with the control plane. Indicative actions include allocating more resources, spawning new instances of cloud-native components, migrating live instances, the complete renegotiating of the entire slice, or others.

All these actions are triggered by proper rules that on the one end, take under consideration monitoring aspects and on the other end, produce actions that are offered by the orchestration loop. The actions supported are a compound of:

- the programmability that the virtualized programmable infrastructure can offer and
- the integration of these supported actions in the orchestration's control plane.

Policies inside 5G-IANA are provided in a prescriptive manner, so as to be executed upon instantiation of a slice. A specific module, namely the Policy Editor, is responsible for authoring these instructions in a formal rules format (i.e., a set of rules). Complementary to composition of rules is the validation of them as well as to approve the applicability or not. The applicability is performed by examining the existence of proper "enablers" that will facilitate the execution of a rule.

Currently, the supported actions that the orchestration loop will support are under definition, so further information will be given in future Deliverable 3.3.

3.4. Software Design

Vertical Service Composition and Customization projects the functionalities of the Vertical Service Composition, the Policy Editing and the QoS Parameters editing through dedicated Editors as Graphical User Interfaces. It is important to note that, these functionalities that are provided through GUIs are accompanied with the respective repositories to exploit persistency. There is also a generic entry GUI that integrates a dashboard for nApps components that provides intelligent application-level analytics and visualisation tools for both application-level and infrastructure-level analytics and monitoring.

The technology behind all the views and the GUIs is React.js and there is also usage of VivaGraph.js (for the graph composition). To validate the nApp graph composition java-based routines are used. For the policies, the back-end services are developed in plain Java code and the policy creation is based on Drools. All the back-end services are implemented in plain Java 11 as Quarkus projects [8].

The Vertical Service Editor along with the QoS Parameter editor is packaged as a Docker container along with their respective backend. Another container is used by the Policy Editor functionality with its respective backend and database.

The software base is coming from Int5Gent Project. The extensions that are built on top of this baseline area are a) the nApp catalogue implementation and integration and b) the resource inventory implementation and integration for registration and selection of available edges (including OBU/RSU programmable resources see [1] on section 3.4).

3.5. Workflows and APIs

The workflow depicted in Figure 9, visualizes the process of onboarding and constructing a nApp from the Service Provider perspective.

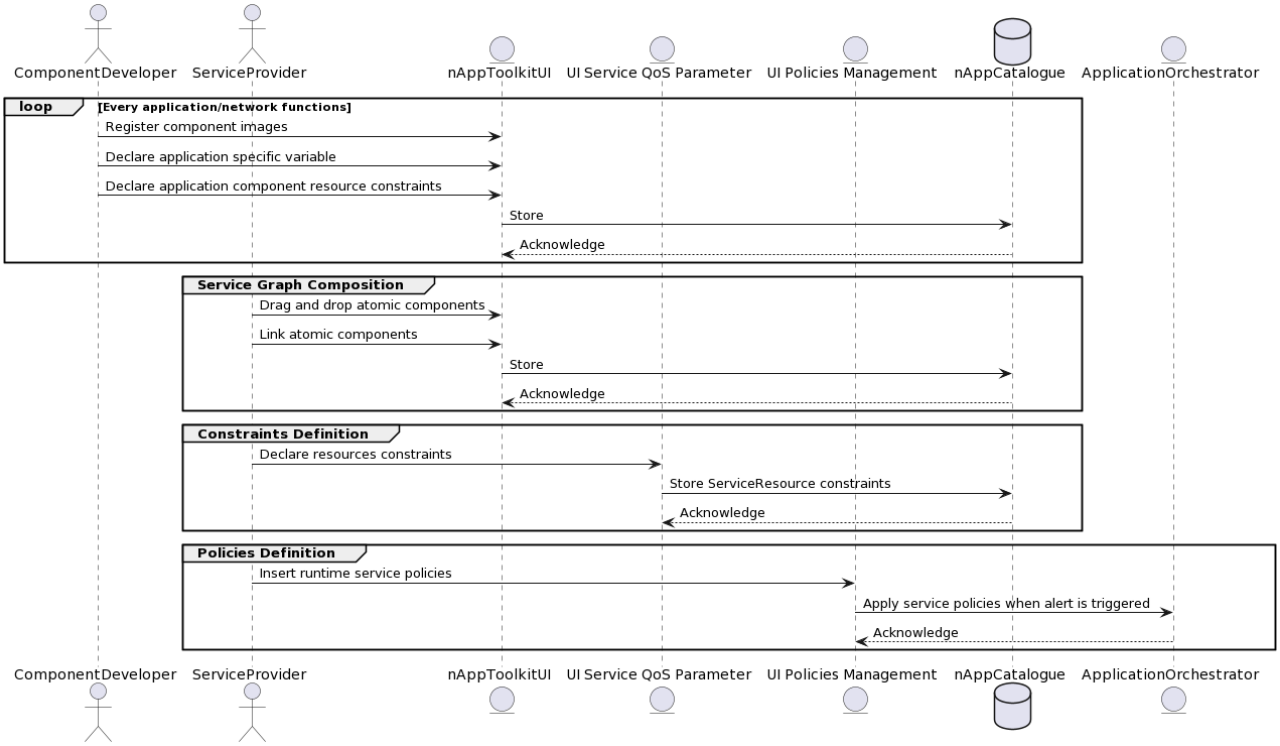

Figure 9 Vertical Service Composition and Customization workflow

For the whole process the service provider is interacting with a series of graphical interfaces that are built to collect specific information. This process is stepwise starting from the uploading of each application function as a docker image into the Centralized Registry of the nApp Catalogue. For the 5G-IANA AOEP the service component (or atomic component) model consists of the image accompanied by the metamodel for its execution. Specifically, the metamodel is comprised by application specific variables (i.e., environmental variables required to be known to the execution environment) and resource constraints for the optimal operation of the component.

Next, the Service provider with the nApp Toolkit GUI's help, retrieves the list of atomic components uploaded and composes a nApp dragging and dropping the components in a canvas and links them together to form a connected graph. Per graph, the service provider can declare constraints regarding the desired state of the deployed graph throughout its lifetime. This is done through QoS identifiers that are translated into KPIs (including network & performance KPIs) that will be requested from the programmable infrastructure provider (see Section 3.2) through the slice negotiation process. Last, the service provider can author custom policies per graph, that based on a threshold for a specific-by-the-service-provider metric or a combination of metrics that are coming from the monitoring engine, will trigger an action for the operational state of the graph or a component of the graph.

3.5.1. NApp Graph REST Endpoints

The Vertical Service Composition and Customization module provides its capabilities through REST interface, listed in Table 1. These capabilities support the four basic operations of persistent storage: Create, Read, Update and Delete (CRUD) operations and specifically the creation of a new nApp graph, the update or deletion of existing ones as well as information retrieval related to one or more graphs, searching capabilities based on specific searching criteria (names, ID etc).

Table 1 nApp Graph REST Endpoints

API	Type	Brief Description
/api/v1/application	POST	Creates an application, if the given field values are valid. The input parameters are: Application Name List of the Atomic Components
/api/v1/application	PUT	Updates an existing application, only if the new values are valid
/api/v1/application/count	GET	Shows the total count of Applications on the Dashboard
/api/v1/application/fetchByInstance	GET	Fetches all applications by instance, to be used for Application filtering field
/api/v1/application/list	POST	Fetches all the applications in pageable format, if the user is authorized
/api/v1/application/search/{name}	GET	Checks if the application name already exists
/api/v1/application/{id}	GET	Fetches application by id
/api/v1/application/{id}	DELETE	Delete an application if the given id is valid

3.5.2. Policies REST Endpoints

This API (listed in Table 2) provides the creation, the modification and deletion of runtime policies.

Table 2 Policies REST Endpoints

API	Type	Brief Description
/runtime	POST	Create a runtime policy Parameters expected: "name": "name of policy",

		"policy": "type of policy", "policyExpression": "expression that will be validated before triggering an action", "policyPeriod": "time to disable the policy", "inertiaPeriod": "waiting time before validating the expression for activation of the policy", "actions": "what is the output when the policy is triggered"
/runtime/{id}	PUT	Update runtime policy by id
runtime/{id}/check	GET	Check if a runtime policy exists
runtime/{id}	DELETE	Delete runtime policy by id

3.5.3. AF/NF Onboarding REST Endpoint

The endpoints described in Table 3 support the CRUD operations regarding the AF/NF onboarding. Specifically, they support the creation, the update, and the deletion of an existing Application or Network function.

Additionally, they support information retrieval related to one or more entities, as well as filtering capabilities upon provided criteria.

Table 3 AF/NF Onboarding REST Endpoints

API	Type	Brief Description
/api/v1/component	POST	Creates a new AF/NF, if it does not exist
/api/v1/component	PUT	Updates an existing AF/NF, only if the new values are valid
/api/v1/component/count	GET	Shows the total count of AF/NF on the Dashboard
/api/v1/component/filtered	POST	Filters out traffic and fetches the AF/NF, if the user is authorized
/api/v1/component/list	POST	Fetches all the AFs/NFs in a pageable format, if user is authorized
/api/v1/component/list/all	POST	Fetches all the AFs/NFs, if the user is authorized

/api/v1/component/{id}	GET	Filters out AF/NF by id and fetches them, if user is authorized
/api/v1/component/{id}	DELETE	Delete an AF/NF only if it exists
/api/v1/component/{id}/candidate/{interfaceID}	GET	Filters out candidate AF/NF by interfaceID and fetches them

4. NAPP CATALOGUE

This section presents the nApp Catalogue, a component in the nApp toolkit that offers a set of functionalities for storing and managing nApp packages. The catalogue provides the whole set of operations to store, update and delete 5G-IANA network applications. This way, the application packages can be reused several times in different context with different requirements. The following sections present the nApp Template information model used to describe and store in a unified manner a network application in 5G-IANA, the software implementation of the nApp catalogue and the APIs which are exposed by the catalogue.

4.1. NApp Template & Information model

The nApp template is the entry point of the nApp package, providing the mappings between the vertical service and the software documentation and software licenses.

An example of nApp template can be found in the [Annex – nApp Template].

The nApp template is a JSON file with the structure depicted in Table 4.

Table 4 nApp Information Model

Name	Type	Mandatory	Brief Description
name	String	Yes	The nApp Package name
description	String	No	A human readable description of the package
version	String	Yes	The version of the nApp package
publicApplication	Boolean	Yes	This identifies if an app is visible to all users or not
hexID	String	No	Internal ID for storing purposes (ID for fast access to specific data)

componentNodes	ComponentNode	Yes	Atomic components of the nApp (see Table 5)
LinkNodes	LinkNode[]	No	The ID of a connection between two components (see Table 6)
type	ENUM (SERVICE, COMPONENT)	Yes	The type of the nApp
specLevel	ENUM (VERTICAL_SPECIFIC, VERTICAL_AGNOSTIC)	Yes	Determines if the nApp is vertical specific or vertical agnostic
accessLevel	ENUM (PRIVATE, RESTRICTED, PUBLIC)	Yes	Determines how this nApp is shared with other stakeholders. PRIVATE nApps are only visible to the account of the nApp developer, RESTRICTED shall only be available to other certain accounts, while PUBLIC ones are available for everyone to see.
requiredEquipments	RequiredEquipment	No	The devices used by the nApp to work
useCase	String	No	The use case to which the nApp is mapped.
testbed	ENUM(NOKIA, TS)	No	The targeted 5G-IANA testbed

softwareLicenses	SoftwareLicense	No	The license terms and attached filed
organization	String	No	Label for a nApp. It is used for searching criteria based on who possess this nApp
serviceCategory	ENUM(HAZARD_NOTIFICATION, VEHICLE_MOVEMENT, SMART_TRAFFIC_PLANNING, INFOTAINMENT)	No	Determines the category of the nApp, based on the service that offers
Required5GCoreService	FiveGServiceSpec	No	The 5G services required by the nApp

4.1.1. Extensions from baseline information model

Inside the nApp information model, there are several complex types which are used to define the Cloud Application Function and Cloud Network Function descriptors which have the information model showed in Table 5 to Table 15. Each table describes a subcomponent (a child) of the nApp information model.

Table 5 ComponentNode information model

Name	Type	Mandatory	Brief Description
componentNodeID	Long	Yes	An identifier for the componentNode
hexID	String	No	Internal ID for storing purposes (ID for fast access to specific data)
name	String	Yes	The name of the component
component	AtomicComponent	Yes	The object which represents the component

Table 6 LinkNode information model

Name	Type	Mandatory	Brief Description

LinkNodeID	Long	Yes	An identifier for the linkNode
componentNodeFrom	ComponentNode	Yes	The source componentNode of the link
componentNodeTo	ComponentNode	Yes	The destination componentNode of the link
graphLink	Link	Yes	The link between the 2 components

Table 7 Link information model

Name	Type	Mandatory	Brief Description
LinkID	Long	Yes	An identifier for the Link
friendlyName	String	No	A link's human-readable name
interfaceObj	ExposedInterface	Yes	The interface used by the link

Table 8 AtomicComponent information model

Name	Type	Mandatory	Brief Description
Id	Long	No	An identifier for the atomicComponent
name	String	Yes	Name of the Component defined by the end user
hexID	String	No	Internal ID for storing purposes (ID for fast access to specific data)
publicComponent	Boolean	Yes	This identifies if an atomic component can be visible to all users or not
architecture	String	No	Defines the supported system that the Atomic Component can be deployed. This is an Optional field
iconBase64	String	No	Icon transformation with Base 64 algorithm

dockerImage	String	Yes	The name of the Docker Image to be referred
dockerRegistry	String	Yes	The URL of the Docker Registry where the Docker Image of the component is uploaded
dockerCredentialUsing	Boolean	Yes	If the Docker Registry needs and authentication
dockerCustomRegistry	Boolean	Yes	If the registry that stores the Docker images is the 5G-IANA registry or a custom Registry
dockerUsername	String	Yes	The Username to access the Docker registry
dockerPassword	String	Yes	The Password to access the Docker registry
exposedInterfaces	ExposedInterface	No	Services that are being exposed inside the container
requiredInterfaces	RequiredInterface	No	Services that are required for the container to be functional
requirement	Requirement	No	Resource requirements for the container to be operational
healthCheck	HealthCheck	No	Endpoint for checking the container lifecycle during runtime. This endpoint id being visited by the orchestrator to validate if the container is running or not

Table 9 ExposedInterface information model

Name	Type	Mandatory	Brief Description
interfaceID	Long	No	An identifier of the interface
name	String	Yes	Name of the interface in a human readable format
port	String	Yes	Specific port the interface exposes

interfaceType	String	Yes	String used from the orchestrator to identify the connectivity requirements of each Atomic Component. It is metadata used to point an EDGE or CORE deployment.
transmissionProtocol	String	Yes	UDP / TCP

Table 10 RequiredInterface information model

Name	Type	Mandatory	Brief Description
graphLinkId	Long	No	An identifier of the interface
friendlyName	String	Yes	Name of the interface in a human readable format
intefaceId	Long	Yes	Points to an ID of an Exposed Interface

Table 11 Requirement information model

Name	Type	Mandatory	Brief Description
requirementId	Long	No	An identifier for the Requirement
CPU	Integer	No	Required Number of CPUs (either physical or virtual)
Ram	Float	No	Required RAM
Storage	Float	No	Required Storage
gpuRequired	Boolean	no	Identifies the deployment requires GPU

Table 12 HealthCheck information model

Name	Type	Mandatory	Brief Description
healthCheckID	Long	No	An identifier for the HealthCheck
name	String	No	Name of the interface in a human readable format
httpURL	String	No	The Endpoint the healthcheck exposes. This is

			being visited in intervals by the orchestrator to verify the operational state of each atomic component
args	String	No	This field complements the httpURL field
interval	Float	No	The time interval of the heartbeat, i.e.: querying the endpoint

Table 13 FiveGServiceSpec information model

Name	Type	Mandatory	Brief Description
fiveGServiceSpecId	String	No	An identifier for the 5G service
Version	String	No	The version of the 5G service
function	ENUM(NWDAF, LCS)	No	The 5G service description
mandatory	Boolean	No	Tells if the 5G service described is mandatory for the nApp
name	String	No	The name of the 5G Service

Table 14 SoftwareLicense information model

Name	Type	Mandatory	Brief Description
id	Long	No	An identifier for the software license record on the DB
softwareLicenseId	String	Yes	The software license identifier in the nApp package
openLicense	boolean	Yes	If the License is open or not
licenseFile	String	Yes	The path to the license file
validationURL	String	No	An external URL to be used while validating the license during the

			service instantiation (if required)
type	ENUM(APPLICATION_PROPRIETARY, INSTANCE_BASED, TIME_BASED, FLAT)	No	Determines the general licensing terms of the license

Table 15 RequiredEquipment information model

Name	Type	Mandatory	Brief Description
requirementEquipmentId	Long	No	An identifier for the Requirement
Type	ENUM(SENSOR, IOT_GW, ACTUTOR, OTHER)	Yes	The type of the Equipment that is required by the nApp
Protocol	String	Yes	The protocol used to exchange message with the equipment
accessLevel	ENUM(READ, READ_WRITE, WRITE)	No	The type of access required to the hardware device.
Location	String	No	A geographical identifier of the place where the equipment should be placed i.e.: the position of the OBU
Description	String	No	A brief description of the hardware
Mandatory	Boolean	Yes	If the hardware is required for the execution of the nApp

4.2. Software Design

The nApp catalogue is a Java application created with Spring Boot, an open-source framework used to build standalone and production ready applications. The catalogue will interact with two components:

- the Vertical Service Composition and Customization to compose a new network application or to update and existing network application,
- the Gitlab platform to store and onboard a stable network application.

The catalogue exposes several APIs to perform all operations to the nApp package management from the two platforms already mentioned.

The software baseline is derived by the nApp catalogue of VITAL-5G [5], but in this version we enhance the information model to describe more in details an atomic component, and the links between them. In addition, the 5G-IANA nApp catalogue can upload atomic components without the needs to be encapsulated in a nApp template. Finally, the 5G-IANA nApp catalogue includes customized management operations like the retrieve and deletion of atomic components, queried specifically, like for example.

4.3. Workflows and APIs

The following section explains the operations made available to the user, i.e.: a developer, a composer GUI or a Gitlab platform.

4.3.1. On-boarding

The on-board operation, in Table 16, is used to store a network application to be reused in the future, and to be deployed on the 5G-IANA environment.

The network application is uploaded as a .zip file with the following information:

- Blueprint.json, the template to be onboarded on the catalogue, with all the information regarding the components and the links between them referred in the Section 4.1.
- Licenses folder, which contains all the licenses for the components used in the network application defined by the nApp package.
- Documentation folder, which contains all the documentation for using the network application like all the component's documentation.
- Test Cases folder, a folder containing the test cases to be executed to be completely sure that the network application is stable and works perfectly.

Table 16 On-board Endpoint

API	Type	Brief Description

/portal/catalogue/netapppackages	POST	Onboard a new nApp template with licenses folder, documentations folder and test cases folder. All this information is packaged in a .zip file and passed as an argument to the REST Endpoint

Depending on the development cycle, the on-boarding REST API is used by the nApp developer, the Vertical Service Composition and Customization or from the Gitlab platform. In the first development cycle, a nApp is onboarded directly from the Vertical Service Composition and Customization, using the REST API provided by the catalogue.

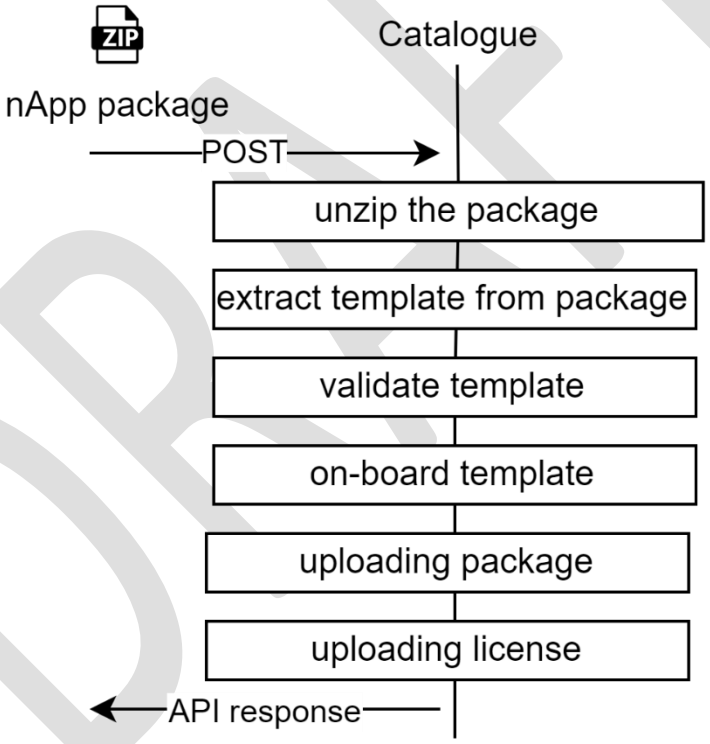

Figure 10 On-boarding workflow

After calling the POST REST API, the catalogue verifies that the nApp information is correct. If anything in the template or in the folder structure, is not correct, the catalogue terminates the operation and notifies the caller with an error message. If the template is correct, the catalogue starts storing the blueprint information

in its relational database and the licenses, the documents and the test cases in its object storage, to be retrieved by the other components of the platform (Figure 10).

The API response, could be one of the following:

- 201 – Created - onboarded of new nApp package with ID=<UUID>
- 400 – Malformed request - The request contains elements impossible to process
- 409 – nApp package already existing - There is a conflict with the request
- 500 – Internal exception - Status 500.

4.3.2. Query

The query REST API, in Table 17, is used by the Vertical Service Composition and Customization and by the nApp developer for searching the catalogue for applicable network applications or atomic components.

Table 17 Query Endpoint

API	Type	Brief Description
portal/catalogue/netapppackages/blueprint?parameter={PARAMETER}	GET	Return a list of blueprints which match the query parameters reported below

A list of parameters is provided to the query method to be used as a filter on the catalogue. The method should return either an atomic component template or a network application template, depending on the nature of the argument supplied as a filter.

The list of parameters included in the method are:

- Testbed: For retrieving nApps and components that reside in a particular testbed.
- AccessLevel: For retrieving specific nApps and components based on the Access Level.
- SpecLevel: For retrieving specific nApps and components based on the Spec Level.
- UseCase: For retrieving specific nApps and components based on the Use case where they are used.
- ServiceCategory: For retrieving nApps and components that are used for a particular service category.
- DockerRegistry: For retrieving nApps and components that are stored in a specific centralized registry.
- DockerCustomRegistry: For retrieving nApps and components from the 5G-IANA centralized registry, or, in case of private nApps and components, for retrieving them from a private registry.

4.3.3. getBlueprint

This REST API Endpoint depicted in Table 18, returns a JSON which identifies and describes the nApp Template stored in the catalogue for the nApp Package Id passed as parameter.

Table 18 getBlueprint Endpoint

API	Type	Brief Description
portal/catalogue/netapppackages/{nAppPackageId}/blueprint	GET	Return a specific blueprint

4.3.4. GetAllnAppPackages

The getAllnAppPackages action, in Table 19, is another way to query the catalogue. There are no parameters required for this operation, which simply searches the catalogue for every template that has been uploaded.

Table 19 GetAll nApp packages Endpoint

API	Type	Brief Description
portal/catalogue/netapppackages	GET	Return all the blueprints stored in the catalogue

A list of templates with some unique information, such as the Identifier, the name, and the version of each template contained in the catalogue, are returned by this REST API.

4.3.5. Delete

This operation in Table 20, deletes a network application package from the catalogue and must be used with a nApp package id as parameter for deleting the right package from the catalogue. The nApp package id could be retrieved by query methods. The operation deletes the onboarded nApp template from the catalogue which can no longer be used. Vertical services that used the deleted template need to be updated with new templates or components.

Table 20 Delete Endpoint

API	Type	Brief Description
portal/catalogue/netapppackages/{nAppPackageId}	DELETE	Delete the blueprint from the catalogue

5. APPLICATION AND NETWORK FUNCTIONS DEVELOPMENT

As defined in Section 2.2, a nApp is structured by a number of AF and NF components that can communicate with each other and can be instantiated separately with different requirements. The ensuing nApp can enable scalable vertical service deployment while taking into account predefined requirements and resource availability. Furthermore, in the context of the project we distinguish the NF and AF in three different categories: The baseline AF and NF, shown in Table 21 and Table 22 respectively, and the communication NF shown in Table 23. Baseline components are the AF and NF that compose the 5G-IANA nApp Starter Kits i.e., the different nApps offered by the project that third parties can use as a baseline to develop their own nApps. Communication NF are used by multiple nApps to handle either 5G or Cooperative Intelligent Transport Systems (C-ITS) communications.

The following section presents the AF and NF that will be available in the 5G-IANA registry, as an initial point of reference that could be used by third party experimenters. Additional details such as the progress of the development, interfaces offered by each VNF, details on their inputs and outputs will be available in the deliverable D4.2 'First report on intelligent nApps and 5G-IANA UCs development' due in M25.

Table 21: Baseline AFs offered by 5G-IANA

VNF id	VNF name	Description
B-VNF01	Manoeuvre Planning	Gets configuration data from the Subscription Service (B-VNF02) and receives poses and trajectories from the Vehicle Interface (C-VNF04). Then, it issues manoeuvre coordination replies after having calculated the best trajectories every vehicle should follow for a safe and efficient travel.
B-VNF02	Vehicle Subscription Service	The service enables the enrolling of vehicles to the Manoeuvres Coordination for Autonomous Driving nApp to let them participate to the manoeuvre coordination
B-VNF03	AR content repository	A storage for AR objects that can be retrieved by B-VNF04
B-VNF04	AR Media Access Function	The AR media access function is an AR streaming application that relies on buffering and multi-threading techniques to give access to multi users to different AR content such as 3D objects. This AF provides the access to the AR content stored in B-VNF03.
B-VNF05	Video Encoding/Decoding	This AF encodes video so it can be transmitted through the 5G network. It is also responsible for decoding and playing the received video on a web application.
B-VNF06	Sensors' data analysis	Processes the information from vehicle sensors and takes decisions regarding its' movement

B-VNF07	Remote Driving Module	This AF receives the control orders (direction, angle, and speed) from the actuator and moves the vehicle accordingly
B-VNF08	Remote Driving Central Control	This AF is the responsible of collecting the information from the driver. For this purpose, a steering wheel is used. The movement data from this peripheral is sent to the server to be processed by the actuator
B-VNF09	Object Detection with Deep Learning	Video captured is processed on the edge to detect pedestrians, cars, and/or road elements such as traffic signals
B-VNF10	Vehicle Condition Warnings Service	Representation of warning signals and alerts in the GUI of a WebApp
B-VNF11	360° Video Stream Endpoint	This NF facilitates sending the 360° Video Stream from the Far Edge to Edge Cloud
B-VNF12	Foveated Rendering Sink	This VNF receives foveatic data (i.e., "fixation points") from B-VNF21 and provides it to B-VNF18
B-VNF13	360° Video Stream Cache	This VNF handles 360° Video Stream and acts as a buffering mechanism that can be employed to maintain video fidelity to the end users, even in no network service availability scenarios
B-VNF14	Foveated Rendering Data Broker	This VNF is a data broker that receives foveatic data (i.e., point of view) from the VR users and acts as a broker for modules that consume this data, located in the Far Edge
B-VNF15	VR Server Module	An authoritative Unity server that is the backbone the VR application facilitating the Virtual Bus Tour presented in UC3
B-VNF16	Log Reporting Service Data Broker	A network function that exposes Use Case specific data e.g., location-based data stored in the B-VNF22. This data can be used either for UC specific functionalities and for debugging/monitoring purposes
B-VNF17	Active Network Monitoring Module	This NF provides a mechanism that will estimate the available network bandwidth utilizing active probing. This estimation will be used as input to B-VNF20
B-VNF18	360o video slicer	This AF masks a 360o video stream so that the parts where the users focus have high resolution while the remaining parts have low resolution
B-VNF19	Privacy Masking Module	This AF applies privacy masking to a 360º video stream, meaning that footage of pedestrians passing by, car plates etc. is blurred for anonymization
B-VNF20	Live Stream Encoder	This AF the video encoding i.e., compressions and re-encoding tasks. The anonymized stream is split into tiles and encoded into MPEG-DASH and HLS stream i.e., use of chunks at different qualities. It receives information from the B-VNF17 to decide if compression is needed
B-VNF21	Field of View Predictor	This AF utilizes Deep Learning AI techniques to predict the future Points-of-View of VR users
B-VNF22	UC-Specific Log Reporting Service	A network function that exposes UC related data e.g., location-based data stored in B-VNF16. This data is used either for UC specific functionalities and for debugging/monitoring purposes

B-VNF23	Monitoring VNF	Collecting data from the field (e.g., sensors, cameras), or services such as the cooperative awareness service (B-VNF28) and storing them. Also distributing data (including video streams) to 3rd parties, e.g., other VNFs
B-VNF24	Streaming VNF	Streaming VNF is a video proxy component receiving video stream from cameras and forwarding it to the end users
B-VNF25	Analytics VNF	Analytics VNF serves for data visualization and reports creation. Data visualization and report structure is based on customer requirements
B-VNF26	Multi-object tracking	This VNF has the functionality of detecting and tracking objects from a video stream
B-VNF39	vDNS	The vDNS is based on the open-source BIND9 software. The DNS is packaged with a RESTful server to enable its run-time configuration (Day1/Day2)
B-VNF40	Virtual FW	The vFW is based on the open-source VyOS software. The vFW can be also used to perform an ad-hoc traffic steering using the routing functionalities
B-VNF41	virtual IDS/vIPS	Virtualized IPS/IDS that detects and prevents the attacks based on Network Monitoring VNFs data collected
B-VNF42	Virtualized Cache - vCache	This AF is the cache on the Edge Server. The Virtualized Cache is based on the Apache Traffic Server (ATS) open-source software, which can be configured to act as a reverse proxy. The function currently offers also an embedded RESTful server that enables its run-time configuration (Day1/Day2). In addition, it embeds also a Telegraf agent to export metrics
B-VNF43	UHD Origin Streaming Server	The UHD Origin Streaming Server is based on the PLEX community version. It's an UHD-capable Origin Server that performs adaptive streaming using MPEG-DASH
B-VNF44	Load Balancer	The Load Balancer is based on the open-source HAPROXY software. It provides load balancing functionalities for applications based on HTTP/TCP, basically performing HTTP redirects towards application servers. The Load Balancer can be configured to apply different policies and offers a RESTful server for its run-time configuration (Day1/Day2), e.g., to add a new application server to its farm. Load balancing between cloud and edge
B-VNF45	Elasticsearch (incl. Kibana and Logtash)	Implements the Elasticsearch stack (Logstash, Elasticsearch and Kibana) for monitored data management, analysis and storage and for processing applications' data and logs' events. The Elasticsearch Stack is a monitoring framework composed by different tools with different functionalities. Beats are data/log/event collector, Logstash is a data aggregator and Elasticsearch is time-series DB
B-VNF46	Telegraf	Telegraf is a monitoring agent to export data/metrics/statistics through the usage of plugins
B-VNF47	Hazardous event receiver and display	This VNF is responsible for receiving information on road risk level change and display a relevant warning to the driver

B-VNF48	Hazardous driving behaviour detection	This VNF will be responsible for detecting and evaluating hazardous driving events (harsh braking, harsh acceleration, speeding, mobile use)
B-VNF49	DML Aggregation Node (AggN)	This VNF: (A) interacts with the DMLO so as to perform client selection (see D3.1); (B) dispatches (at each round) the global ML model to the currently selected clients I.e., instances of ML node –Training Agent (MLN) (B-VNF50) at participating OBUs; (C) receives locally trained models and aggregates them into a global model (at each round); (D) delivers statistics to the DMLO
B-VNF50	ML node -Training Agent (MLN)	This VNF: at each training round (A) receives the current version of the global ML model from the DML Aggregation Node (AggN) (B-VNF49); (B) receives training data from the ML pre-processing node (B-VNF51); (C) trains a ML model using training data obtained; (D) forwards the locally trained model to the Aggregation node (B-VNF49)
B-VNF51	ML Preprocessing	The VNF transforms the acquired raw data into an ML-specific dataset e.g., for predictive QoS into a spatio-temporal network latency data-set

Table 22: Baseline NFs offered by 5G-IANA

VNF id	VNF name	Description
B-VNF27	ETSI Decentralized Environmental Notification Service	This NF is in charge to manage in input and in output the Decentralized Environmental Notification Messages (DENMs) that provide alerts about possible hazards and events
B-VNF28	ETSI Cooperative Awareness Basic Service	This NF is in charge to manage in input and in output the Cooperative Awareness Messages (CAMs) that provide information about the vehicle status, its position and dynamics
B-VNF29	ETSI Collective Perception Service	This NF is in charge to manage in input and in output the Collective Perception Messages (CPMs) that provide information retrieved from the processing of raw sensors data (e.g., object type, object dynamics, free spaces)
B-VNF30	ETSI Manoeuvre Coordination Service	This NF is in charge to manage in input and in output the Manoeuvre Coordination Messages (MCMs) that are used by vehicles to share their intentions about manoeuvres and to coordinate manoeuvres
B-VNF31	ETSI Traffic Light Manoeuvre Service	This NF is in charge to manage in input and in output the Signal Phase And Timing Extended Messages (SPATEMs) that provide information related to the traffic light controller of a specific intersection
B-VNF32	ETSI Road and Lane Topology Service	This NF is in charge to manage in input and in output the MAP Extended Messages (MAPEMs) that provide information related to the geometry of a given intersection and the topology of related lanes
B-VNF33	ETSI Infrastructure to Vehicle Information Service	This NF is in charge to manage in input and in output the Infrastructure to Vehicle Information Messages (IVIMs) that provide information about road signage

B-VNF34	Position and Time Service	This NF implements a Position and Time Service to offer similar functionalities to the ETSI Position and Time (PoTi) service specified in ETSI EN 302890-2. On the OBU the VNF provides time and position retrieved from a GNSS RTK receiver
B-VNF35	Enhanced Local Dynamic Map Service	This NF implements a Local Dynamic Map (LDM) Service that provides information to applications about local events, real time dynamic object information and other nearby connected vehicles
B-VNF36	Events Relevance Service	This NF selects the events that are relevant to a vehicle according to its trajectory
B-VNF37	OBU Localization Service	This NF is to be deployed at the edge server and it provides the OBUs position information to AF/NF running at the edge server using an approach compliant to the MEC Location API as defined by the ETSI GS MEC 013
B-VNF38	Simulator of ETSI Cooperative Awareness Basic Service	This NF simulates the same functionalities of B-VNF28 "ETSI Cooperative Awareness Basic Service"
B-VNF52	Network Monitoring	The Network Monitoring analyses and records all the network traffic on the network and provide a single point of access to the data stored. The VNF monitors the network behaviour passively and actively from the edge. It sniffs the application packets received by the edge and calculates network-based metrics (such as data rate and latency)
B-VNF53	QoS prediction	This VNF is based on the trained Distributed ML model present at the Edge node. An LSTM prediction model is trained on each edge node and aggregated at the DML server. This aggregated global model is then transmitted to all the Edge nodes for training and inference
B-VNF54	Network (NW) monitoring	The VNF monitors the network behaviour passively and actively from the edge. It sniffs the application packets received by the edge and calculates network-based metrics (such as data rate and latency)
B-VNF55	Vehicle Abstraction Service	This VNF guarantees protocol compatibility between vehicle and the Vehicle Interface (C-VNF04)

Table 23: 5G or ITS communications related NFs offered by 5G-IANA

VNF id	VNF name	Description
C-VNF01	Uu data communication	This NF is in charge to exchange data among AFs on the far-edge and AFs on edge and cloud segments. The NF transmits and receives data on behalf of an AF over the Uu interface using the demanded application layer protocol
C-VNF02	Free5Gcore	Free5GC is an open-source 5G Core Network that can be orchestrated and allows to perform network slicing by deploying multiple AMFs and UPFs

C-VNF05	Sensors' data capturing	Collects data related to the distance and angle of a vehicle to near obstacles from sensors
C-VNF06	Actuators Interface	Receives the movement commands from the user and generates the control order (direction, angle, and speed) in a language understood by the vehicle
C-VNF07	Uu C-ITS messages communication	This NF implements the communication of C-ITS messages over the Uu interface. This NF follows the same approach used in the 5G-CARMEN project for the communications among vehicles via network (i.e., V2N2V) and between vehicles and the network (i.e., V2N)
C-VNF09	Vehicle interface	This NF is in charge of interacting with the vehicle network to exchange information between the vehicle and the OBU
C-VNF10	Sensor Awareness Service	This NF is in charge of receiving data from available connected sensors and to provide the retrieved information to other NFs
C-VNF11	Long-distance data communication	This NF is in charge to transmit and to receive data for other VNFs for long-distance 5G communication channel to specific edge/cloud service. It can be used by third party developers that do not want to implement any specific protocol at the application layer of the communication.

6. CONCLUSION

In this document we present the initial release of the 5G-IANA toolkit which includes the Vertical Service Composition and Customization and the nApp catalogue components. It defines an information model in Section 4.1, for modelling a nApp, which is provided by the Vertical Application Composition and Customization and onboarded on the nApp catalogue.

In Section 2.2.2 the template to use to create an atomic component in a standardised way is specified. With this structure, the developer is guided in the implementation of AFs/NFs by using a CI/CD approach.

The nApp toolkit provides a list of nApp starter kits which are available to familiarize with the platform and which do not require a high level of proficiency to be used. A nApp developer can use a nApp and can exploit the 5G services made available by the 5G infrastructure where this application is deployed.

Next steps, that will be reported on the next WP4's deliverable, are the composition of Vertical service, intended as concatenation of multiple nApps and the enhancement of the GUI for a more user-friendly use.

This deliverable will be used from WP3-4-5 as a starting point to compose, deploy and test a network application or a vertical service exploiting the testbed's 5G services.

REFERENCES

1. 5G-IANA - D2.1 - Specifications of the 5G-IANA architecture
2. 5G-IANA - D7.7 - Exploitation plan
3. https://docs.docker.com/registry/spec/api/
4. https://docs.gitlab.com/ee/ci/docker/using_kaniko.html
5. VITAL5G - D2.1 Initial NetApps blueprints and Open Repository design
6. 5G ERA, “D4.1: 5G-ERA Middleware initial version”, June 2022
7. https://12factor.net/
8. https://quarkus.io/
9. https://spec.openapis.org/oas/v3.0.1

ANNEX – NAPP TEMPLATE


```
{
        "name": "string",
        "description": "string",
        "version": "string",
        "publicApplication": "boolean",
        "hexID": "string",
        "organization": "string",
        "type": ["SERVICE","COMPONENT"],
        "serviceCategory": ["HAZARD_NOTIFICATION","VEHICLE_MOVEMENT",
"SMART_TRAFFIC_PLANNING", "INFOTAINMENT",],
        "packageType": ["HELM"],
        "specLevel": ["VERTICAL_AGNOSTIC","VERTICAL_SPECIFIC"],
        "accessLevel": ["PRIVATE","RESTRICTED","PUBLIC"],
        "useCase": "string",
        "testbed": ["NOKIA","TS"],
        "softwareLicenses": [{

        }],

        "componentNode":[{
                "componentNodeID": "long",
                "hexID": "string",
                "name": "string",
                "component": [{
                        "id": "long",
                        "name": "string",
                        "hexID": "string",
                        "publicComponent": "boolean",
                        "architecture": "string",
                        "iconBase64": "string",
                        "dockerImage": "string",
                        "dockerRegistry": "string",
                        "dockerCredentialUsing": "boolean",
                        "dockerCustomRegistry": "boolean",
                        "dockerUsername": "string",
                        "dockerPassword": "string",
                        "exposedInterfaces": [{
                                "interfaceID": "long",
                                "name": "string",
                                "port": "string",
                                "interfaceType": "string",
                                "transmissionProtocol": "string"
                        }],
                        "requiredInterfaces": [{
```

```
                        "graphLinkID": "long",
                        "friendlyName": "string",
                        "interfaceId": "long",
                    }],
                    "requirement": [{
                        "requirementId": "long",
                        "CPU": "Integer",
                        "Ram": "Float",

                    }],
                    "healthCheck": [{
                    }],
                }]
        }],
        "LinkNodes":[{
                "linkNodeID": "long",
                "componentNodeFrom": "ComponentNode",
                "componentNodeTo": "ComponentNode",
                "graphLink": {
                    "linkID": "long",
                    "friendlyName": "string",
                    "interfaceObj": {
                        "interfaceId": "long",
                        "name": "string",
                        "port": "string",
                        "interfaceType": "string",
                        "transmissionProtocol": "string"
                    }
                },

        }],
        "required5GCoreService":[{
                "fiveGServiceSpecid": "string",
                "version": "string",
                "function": ["NWDAF","LCS"],
                "mandatory": "boolean",
                "name": "string"
        }]
}
```