



OPEN CALL #2
TECHNICAL MANUAL
PLATFORM USER GUIDE

Launch date: 20th January 2024

TABLE OF CONTENTS

1. **INTRODUCTION**
 - 1.1. *Definitions*

2. **AF/NF DEVELOPMENT AND PACKAGING**

3. **AF/NF UPLOADING**
 - 3.1. *Component's privacy*

4. **NETWORK APPLICATION COMPOSITION**
 - 4.1. *Enter the web platform*
 - 4.2. *Registration of individual application and network functions (components)*
 - 4.3. *Network app Graph composition and registration*
 - 4.4. *Onboarding*
 - 4.5. *Network Application deployment*

5. **NETWORK APP STARTER KIT**

6. **STEP-BY-STEP ONBOARDING GUIDE**
 - 6.1. *Step#1 NF/AF Development*
 - 6.2. *Step#2 NF/AF Containerization and orchestration*
 - 6.3. *Step 3PRE-I Prerequisite: Communication with 5G-IANA - GitLab access & GitLab Repository Creation*
 - 6.4. *Step 3PRE-II Prerequisite: Communication with the targeted 5G-IANA testbed owner - VPN Access*
 - 6.5. *Step 3PRE-III Prerequisite: Request Composer-GUI credentials*
 - 6.6. *Step 3PRE-IV Prerequisite: Add docker-registry self-signed certificate*
 - 6.7. *Step#3 5G-IANA Onboarding preparation*
 - 6.7.1. *Step#3.1: Network app image creation & upload to Central Repository*
 - 6.7.1.1. *Step#3.1.1 Clone the network app repository*
 - 6.7.1.2. *Step#3.1.2: Update for target NF/AF*
 - 6.7.1.3. *Step#3.1.3: Create gitlab-ci.yml & Configuration of GitLab Repositor*
 - 6.7.1.4. *Step#3.1.4: Push changes to remote repository*
 - 6.7.1.5. *Step#3.1.5: Merge Request to master branch*
 - 6.7.2. *Step#3.2: Descriptor Adaptation for 5G-IANA platform*
 - 6.7.2.1. *Step#3.2.1 Create a maestro-compose.yml to your local repository.*
 - 6.8. *Step#4 Blueprint Creation & Version verification of uploaded artefacts*
 - 6.8.1. *Step#4.1 Version verification*
 - 6.8.2. *Step#4.2 Blueprint creation*
 - 6.9. *Step#5 5G-IANA Platform Onboarding*
 - 6.10. *Step#6 Complete Service Chain Validation in regards to proper communication and functionality of all images on the testbed*

REFERENCES

ANNEX – NETWORK APP TEMPLATE



Executive Summary

The following manual outlines a methodology for the development and uploading of Application and Network functions in the context of the 5G-IANA automotive ecosystem. The document provides an overview of the principles involved in the process and highlights the steps that should be taken to ensure a successful outcome.

The manual emphasizes the importance of following a structured approach to development, including clear objectives and a detailed plan of action.

The manual provides practical guidance on the process of uploading and managing functions within a 5G context. This includes advice on managing the deployment process, monitoring performance metrics, and addressing any issues that may arise during the development process. The document also emphasizes the importance of continuous improvement and iteration, with a focus on refining the functionality of the system to meet the specific needs of end-users.

In Section 1, some definitions are given in order to comprehend the 5G-IANA project. In Section 2, the development and packaging process is explained, while in Section 3, it is explained how the developed components can enter into the 5G-IANA ecosystem. In Section 4, described is the concatenation of the various developed functions to create a network application deployable on a 5G-enabled environment.

Section 5, reports the list of network apps already available in the 5G-IANA repository to be used by third parties and Section 6 describes the onboarding operation that a developer should do to deploy a 5G enabled network application.

Section 6, is a step by step guide to assist with the onboarding process of software artefacts to the 5G-IANA platform and with network app composition, and it refers to the expert user, namely to the user who has already been trained for using the 5G-IANA platform.

Finally in the Annex there is the network app Template used by the network app toolkit. Please note that the list of AFs/NFs/network apps available in the network app toolkit can be found in 5G-IANA website: .

1. INTRODUCTION

The purpose of this manual is to provide a standardized way to develop, package and use application/network functions and network applications in the 5G-IANA automotive environment.

1.1. Definitions

In the following section of this document, we describe how the developers should use the features provided by the platform and we refer to some particular definitions:

An atomic component is a virtualizable function that is deployable in a container. In simple words, an atomic component is a docker image that runs on an executor and offers some functionalities.

There are 2 categories of atomic components:

- Application Functions (AF) which implement the logic of the applications. Examples of application functions can be:
 - A remote driving module application function, which receives control orders: direction, angle, and speed from the actuator and moves the vehicle accordingly.
 - A hazardous driving behaviour detection function, which is responsible for detecting and evaluating hazardous driving events (harsh braking, harsh acceleration, speeding, mobile use).
- Network Functions (NF) that implement the communication between application functions and ensure connectivity with the 5G network. For example:
 - A sensor's data capturing function which collects data related to the distance and angle of a vehicle to near obstacles from sensors.

A component or a network app follows several steps from the development to the deployment. In the document we will use the following terminology:

- Upload/Registering: The process to insert a docker image in a docker registry.
- Onboarding: The process to save a network application or an atomic component inside the catalogue to be used several times.
- Deployment: The process to orchestrate a network application and deploy it on a cluster that will run the application and will expose the results.

2. AF/NF DEVELOPMENT AND PACKAGING

The 5G-IANA platform offers great potential for third-party developers to use its network applications, tools, and testbed environment to create new applications for the automotive sector. One of the key aspects that makes this platform stand out is the ability to use the network app starter kit as a software baseline for building applications (see Section 5). These starter kits are publicly available on a Git repository and offer several open-source software components that developers can adjust and extend as necessary to create new applications. Additionally, there are also starter kits that include proprietary software components that can be combined with external third-party software to develop new applications.

A developer can even create a brand-new network application starting from scratch, creating several atomic components that are uploaded on the centralized registry of the 5G-IANA environment to be used as basic components to create the desired network application.

Developers use various types of artifacts to create network applications, prioritizing both functionality and component privacy. Each component of the network application should be designed as a self-executing function that can expose relevant information and receive the required input through several communication frameworks like REST API, message bus, pub-sub and so on. For that, in 5G-IANA the Docker approach is used. With this approach a developer is free to build his own application/network function with the most suitable language or the language he/she knows better. After a functionality is developed, it needs to be built and packaged in a 5G-IANA compliant way. Therefore, it is mandatory to provide a Dockerfile to compile, build, and package as a docker image each application/network function to be used. In this way it is possible, at a later time, to compose a network application linking together several application/network functions, that can be viewed as components of a network application.

The Docker platform's efficient separation of the application from the infrastructure provides virtualization for application logic, ensuring consistency and ease of deployment across different environments, including 5G-enabled testbeds. Developers can make use of containers to package their applications with the necessary dependencies and libraries, further simplifying the deployment process. DevOps pipelines, which automate the entire software development process from code changes to deployment to production, can assist developers in complying with other virtualized components. Figure 1 provides an overview of a high-level DevOps pipeline for component development.

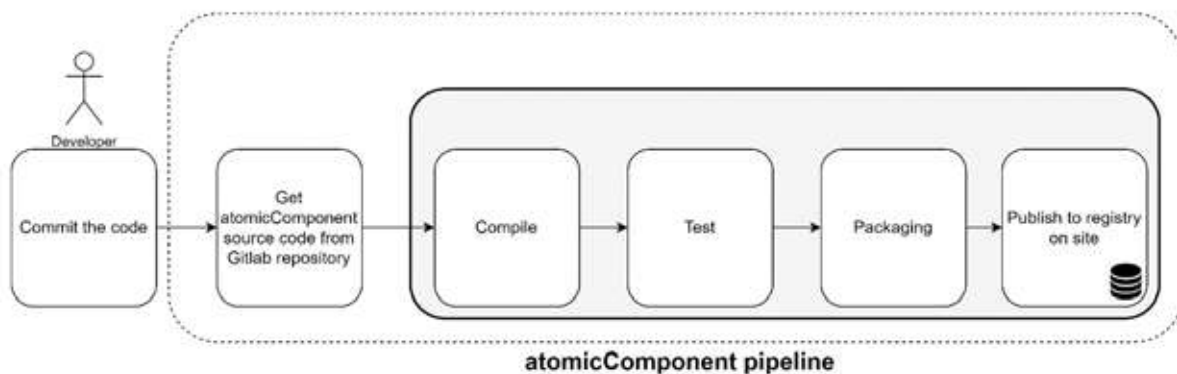


Figure 1 DevOps pipeline for components

In order to let the DevOps pipeline build and package the components, it is required that each application and network function developed has the structure depicted in Figure 2.

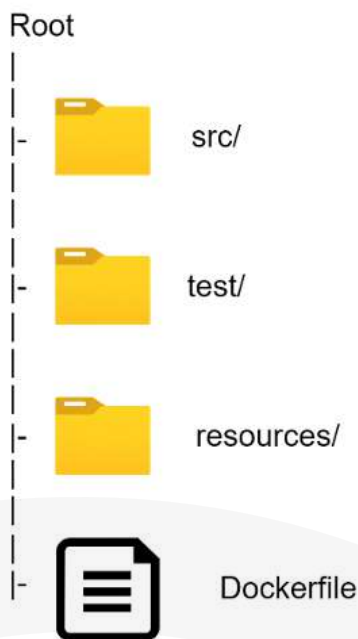


Figure 2 application/network component structure

- The src/ folder will contain all the source files used by the component to provide its functionalities,
- The test/ folder will contain the test cases used by the developer to test the functionalities,
- The resource/ folder will contain the resources needed by the component, and all the documentation that is necessary for using the component.
- The Dockerfile that will be taken by the DevOps pipeline to build and package the component and upload it on the 5G-IANA centralized registry

3. AF/NF UPLOADING

After a component is developed, it can be automatically uploaded to the 5G-IANA centralized registry and made available for use to the network app developers.

The Gitlab platform available at <https://gitlab.ubitech.eu> has a direct connection to the NOKIA testbed and it is used to provide all the automatic functionalities for building, packaging, and uploading the atomic components on the 5G-IANA environment.

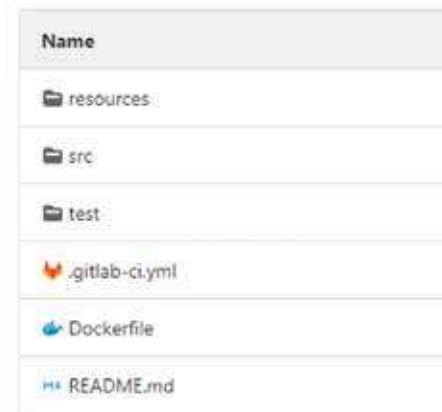


Figure 3 Gitlab project structure

As an example, in Figure 4, we show the content of the Dockerfile for an application function (the network monitoring application function, available on the network app starter kit).



Figure 4 Network Monitoring Dockerfile

We can see that the Dockerfile is divided into 2 stages, the build stage, and the package stage. The former specifies how the component can be built. In the example it is possible to see that the project is a maven project and it is built with the command `mvn -f /home/app/pom.xml clean package`.

The latter specifies how the component can be packaged as a docker image. As it is showed in Figure 4, the docker image is based on the openjdk-11-jre-slim and the entry point is `java -jar /usr/local/lib/ntopng-kafka-producer`.

In the same way, a developer, should provide in the root folder of the projects, a Dockerfile with these two stages in order to be compliant with 5G-IANA.

After the docker image is built by the 5G-IANA CI/CD pipeline, if not specified, the images are directly uploaded on the 5G-IANA centralized registry located in the NOKIA testbed and available to all the network app developers to be used as components for the network app development.

The pipeline uploads the docker image of the component directly on the 5G-IANA centralized registry. If not specified as an environment variable, the version of the image is the GitLab-commit-sha, as depicted in Figure 5.

```
if: '$VERSION' == "" then
  /kaniko/executor --context "${CI_PROJECT_DIR}" --dockerfile "./Dockerfile" --destination "192.168.100.1:5000/oxu/${CI_PROJECT_NAME}:${CI_COMMIT_SHORT_SHA}"
else
  if: '$VERSION' != "" then
    /kaniko/executor --context "${CI_PROJECT_DIR}" --dockerfile "./Dockerfile" --destination "192.168.100.1:5000/oxu/${CI_PROJECT_NAME}:${VERSION}"
```

Figure 5 5G-IANA CI/CD pipeline snippet

3.1. Component's privacy

The operation described above is used for all the open-source components. The source code is available on the shared Gitlab platform and the docker images are available in the shared centralized registry of the 5G-IANA environment. There may be some cases where, for several reasons, the components cannot be open-source or the docker image is private and cannot be used by other developers.

In the former case, the source code can be stored in the private partner repository, but the docker image must be uploaded on the 5G-IANA centralized registry located at 192.168.100.2:5000, to be used by the network app developers. This case implies that the third-party developer needs to have an open VPN connection with the NOKIA testbed to contact the 5G-IANA centralized registry to upload his/her docker images directly. After the docker images are uploaded on the 5G-IANA centralized registry, the images are available to use by the network app developers.

In the latter case, otherwise, also the third-party docker images are private and need to be stored privately. The third-party can have two options: Upload the docker images in his private registry and connect the private registry to the 5G-IANA composer, or upload the docker images in the 5G-IANA centralized registry behind an authentication mechanism, to let the docker images available only to whom has the credentials.

4. NETWORK APPLICATION COMPOSITION

After all the components are uploaded on their specific registry, the network app developer enters the 5G-IANA composer GUI to create a network application.

4.1. Enter the web platform

The first step requires the end-user registration and authentication. Role-based access control features are enabling specific users to use specific features and specific views of the web platform.

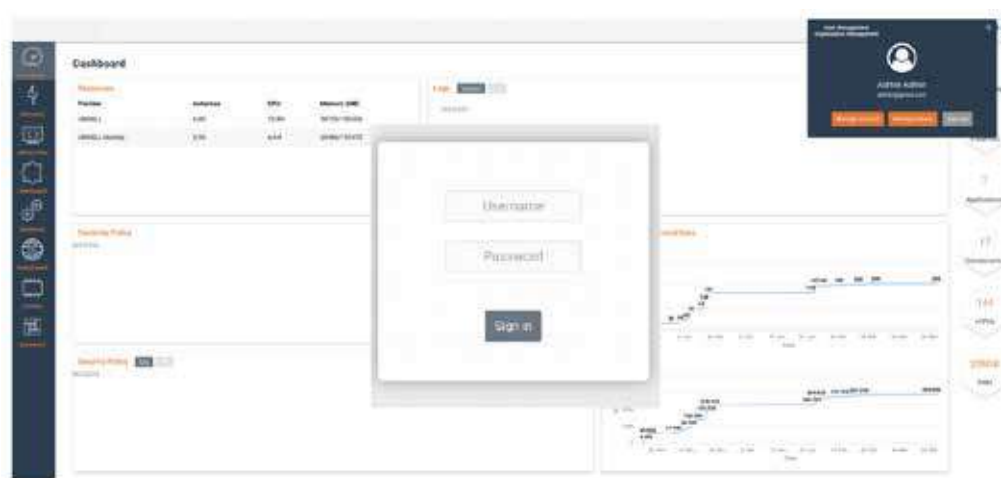


Figure 6 - Roles and Logging into the web platform and web interface sign in

4.2. Registration of individual application and network functions (components)

After registration, the end-user (developer) creates the components to use for composing the network application. This is done by selecting the “components” view from the main interface.

For the creation of the components, one needs first to fill a set of forms with the corresponding values. This type of information can be found in the application’s docker-compose or helm charts descriptor (if such descriptors exist) that the user may have. What must be thought very thoroughly before starting the onboarding of the microservices, is the order in which the components will be created. That can be achieved by creating the acyclic graph of the application, where each link between two microservices represents the dependency that they have. So, for instance if our application consists of a MySQL and a PhpMyAdmin microservice, in our acyclic graph there will be one link that will go from the PhpMyAdmin to the MySQL and it will represent that the first needs the second in order to start and work properly; so, in that case we should create first the MySQL component and then the PhpMyAdmin, in order to be capable to refer to the dependency of the MySQL component during the creation of the PhpMyAdmin component. After one has concluded with the order, one can start onboarding the microservices as components to the platform by filling the following forms (Figure 7):

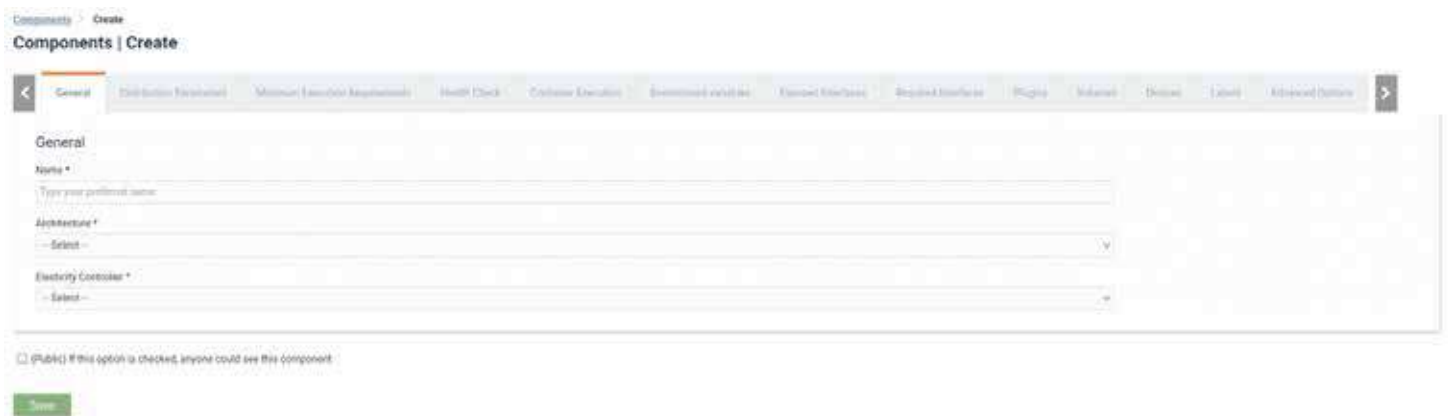
- General Information
 - Name, the name of the component that will be displayed in the GUI
 - Architecture, the architecture in which the component runs
 - ElasticityController,

- Distribution parameter
 - DockerImage, the name of the docker image to be used
 - Docker Username, the username to use for getting the image from the centralized registry or from another custom registry
 - Docker Password, the password to use for getting the image from the centralized registry or from another custom registry
 - Docker Custom Registry, the URL of a docker custom registry, if not specified, the centralized registry will be used.

- Minimum execution requirements
 - vCPUs, the minimum number of CPUs required by the component
 - RAM (MB), the minimum amount of RAM, in Megabyte, required by the component
 - Storage (GB), the minimum amount of storage, in Gigabyte, required by the component
 - GPU enabled, if the component requires or not a GPU

- Exposed Interface
 - A list of interfaces exposed by the component, that contain:
 - Name, the name of the exposed interface
 - Port, the port where the interface is exposed
 - InterfaceType, the interface's type (core or access)
 - Transmission Protocol, the protocol's type (TCP, UDP, TCP/UDP)

- Required Interface
 - A list of already defined interfaces of other components that are added to the network apps



Components **Create**

Components | Create

General Distribution Parameters **Minimum Execution Requirements** Health Check Container Execution Environment variables Exposed Interfaces **Required Interfaces** Plugins Volumes Devices Labels Advanced Options

Distribution Parameters

Docker Image *

From the Docker Registry

Docker Credentials

Use private Docker registry (Username, Password fields)

Docker Username

Docker Password

Use custom Docker registry

Custom Docker Registry

(Public) If this option is checked, anyone could see this component

Components **Create**

Components | Create

General Distribution Parameters **Minimum Execution Requirements** Health Check Container Execution Environment variables Exposed Interfaces Required Interfaces Plugins Volumes Devices Labels Advanced Options

Minimum Execution Requirements

CPUs *

RAM (MiB) *

Storage (MiB) *

Hypervisor Type *

GPU enabled

(Public) If this option is checked, anyone must see this component

Minimum Execution Requirements Health Check Container Execution Environment variables **Exposed Interfaces** Required Interfaces Plugins Volumes Devices Labels Advanced Options

Exposed Interfaces

[Add a new one](#)

Name	Port	Interface Type	Transmission Protocol
<input type="text" value="Type interface name"/>	<input type="text" value="Port"/>	<input type="radio"/> Core <input type="radio"/> Access	<input type="radio"/> TCP <input type="radio"/> UDP <input type="radio"/> TCP/UDP

Minimum Execution Requirements Health Check Container Execution Environment variables Exposed Interfaces **Required Interfaces** Plugins Volumes Devices Labels Advanced Options

Required Interfaces

Label	Select an interface
<input type="text" value="Type a label for the required interface"/>	<input type="text" value="- Select -"/>

Figure 7 – network app component (application and network function) configuration parameters

4.3. Network app Graph composition and registration

After all the required components are made available to the components' registry, the next step is to proceed with the composition of the graph. In this case, the platform GUI provides a view for selecting and linking all the necessary components that constitutes the application graph. This is done by selecting the “Applications” view from the main page and then clicking on “create new”. It is noted that one can also select one of the existing applications listed and then edit its parameters or even modify it by adding new components. Any change will result in the creation of a new network app.

The process below demonstrates how the application graph can be created. All the onboarded application components are listed in the main canvas of the composer GUI as depicted in Figure 8 (left). From that view it is possible to drag and drop the components from the list on the left of the screen to the canvas. After a component is dropped on the canvas, it is included in the network app and some configurations are asked to the network app developer. Dropping a component with required interfaces, popups a window for inserting the component linked with the required interface, as depicted in Figure 8 (right).

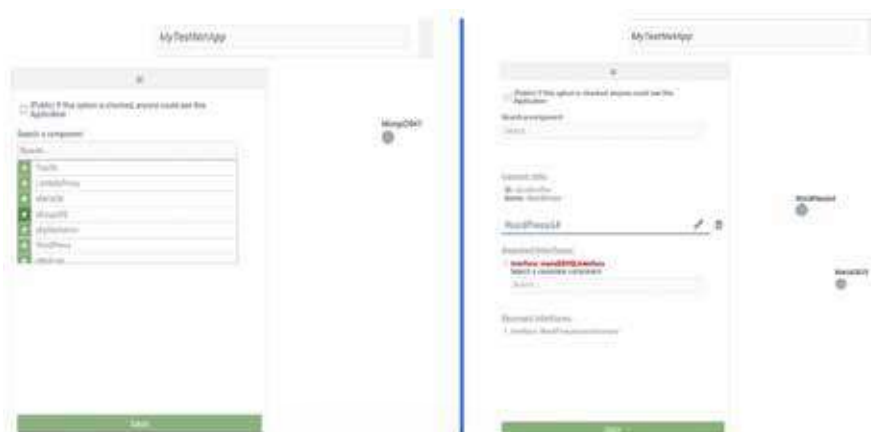


Figure 8 - Components are added to the canvas (left) and Linking components together (right)

When a network app is fully built in the composer canvas, it can be saved clicking on the green button. This action also triggers the onboarding mechanism of the catalogue and the network app is automatically onboarded to the catalogue ready to be deployed.

After the composition of the application graph the GUI provides a set of user interfaces supporting the declaration of a set of resource and high-level network constraints that have to be fulfilled during the placement of the application in order to provide the desired network functionalities. The figure below demonstrates these user interfaces for providing resource and high-level network constraints.

General

Select SSH Key
adminKey

Select Provider
KubernetesSp

Select Region *
gr-athens

Save

Flavor

vCPUs *
1

RAM *
2048

Storage *
20

Save

Figure 9 – Deployment constraints for VIM (upper tab) and resources (lower tab)

Specific reference should be made on the graph link constraints that Figure 10 shows. Such specifications target the requirements that must be satisfied regarding the quality of the virtual link which is established between two components. During the deployment of an application component all interfaces that it exposes are bound to network identifiers that are indicated by the infrastructure provider. Through this configuration, the materialized link between components must satisfy some network requirements in terms of delay, jitter, packet loss and throughput. At this point, it should be clarified that a component that participates in a service mesh may expose two types of interfaces. The first type is the CORE and the other is the ACCESS type.

CORE interfaces are the ones that are used among the components while ACCESS interfaces are the ones that interact with the UEs. It should be clear that Graph Link constraints refer to the interconnection of CORE interfaces only. On the other hand, ACCESS

interfaces entail a completely different metamodel. ACCESS interfaces are like a label for an application component to require a deployment with a network attached to radio equipment. Specifically, the type of constraints that can be provided relate to latency, bandwidth, the QoS Class Identifier (a.k.a. QCI) like in Figure 10.

Set the Constraints of "mariaDBSQLInterface" graph link

Maximum Delay (ms)

Value:

Constraint Type
 Soft Hard

Maximum Jitter (ms)

Value:

Constraint Type
 Soft Hard

Maximum Packet Loss (%)

Value:

Constraint Type
 Soft Hard

Minimum Throughput (Kbps)

Throughput

Constraint Type
 Soft Hard

Save

Figure 10 – Graph link application related constraints

4.4. Onboarding

Once a network application is developed using either the network app starter kit or self-deployed components, it can be onboarded onto the 5G-IANA catalogue, which is available on the platform. This allows developers to provision, monitor, and test their applications using the tools made available in the platform and the 5G-IANA testbed environment. This has the potential to help developers create highly functional and reliable network applications for the automotive sector, delivering great value to end-users and businesses alike.

Third-party developers could use the 5G-IANA platform and some of the network applications developed in the project in order to design, develop and test new network applications for the automotive sector. In particular, the platform can be leveraged to develop and deploy network applications that can be built using the network app starter kit as software baseline.

In detail, the 5G-IANA platform provides a public Git repository to distribute all the network app starter kits and related software documentation, which could be used as a starting point for developing third party network applications. Some starter kits provide open-source software components which can be adjusted and extended as needed to develop new applications, while others provide proprietary software components (libraries or stand-alone applications exposing open interfaces) to be composed with external third-party software for building new applications.

The following table provides the list of available network app starter kits, categorized on the basis of the services and functionalities they implement.

A network app developer could checkout those network apps from the public Git repository and use them as a starting point to make a completely new network app. Documentation on how to use each network application and each of its component, is stored into the Git repository in a dedicated folder as described in the following of this section. A network app starter kit repository is structured as shown in Figure 12.

```
-  
- *blueprint.json  
- license/  
- documentation/  
- test/
```

Figure 12 Network app Starter kit repository structure

The structure contains the network app template called blueprint.json and three optional folders respectively for the licenses of the network application and its components, the documentation on how to use it and the tests that are potentially provided within the starter kit.

The template blueprint is required in order to use the network app and is structured in the following way:

```
{
  "netAppPackageId": "UUID autogenerated",
  "name": "string",
  "description": "string",
  "version": "string",
  "type": [SERVICE, COMPONENT],
  "serviceCategory": [HAZARD_NOTIFICATION, VEHICLE_MOVEMENT, SMART_TRAFFIC_PLANNING, INFOTAINMENT],
  "servicePackageURL": "string",
  "packageType": [HELM, ...],
  "specLevel": [VERTICAL_AGNOSTIC, VERTICAL_SPECIFIC],
  "accessLevel": [PRIVATE, RESTRICTED, PUBLIC],
  "useCase": "string",
  "testbed": [NOKIA, TS],
  "atomicComponents": [],
  "requiredEquipment": [],
  "providedInterfaceSpec": [],
  "softwareLicenses" : []
}
```

Figure 13 Network app template

A network app developer could create his/her own network application using the components that are contained in the network app starter kit templates or could implement his own component, that needs to be containerized and referred into the template in the atomicComponents section. In order to be compatible with the 5G-IANA platform, the new application components need to be delivered as container images manageable under Kubernetes framework. The network app template is automatically created by the 5G-IANA Graphical composer, but a network app developer can create it manually following the structure depicted in Figure 13:

- The atomicComponents section stores all the information of each component in the network application like how to get the image to use, which are the exposed interfaced and which are the connections with the other components.
- The providedInterfaceSpec and softwareLicenses sections contains the references to the documentation and to the software licenses contained in the network app package.
- The requiredEquipment section is used for listing all the equipment that are required for running the network app.

When the network app developer has created his network application filling a network app template, this template could be onboarded on the network app toolkit using a REST API.

The onboarding mechanism will validate the network app template and will retrieve all the information of the network application, in order to be used by the orchestrator to request a deployment. Furthermore, a third party could use an already available network application stored in the catalogue, and modify them directly in the Vertical App Composition & Customization tool in the same way as described in Section 4.

4.5. Network Application deployment

Given that all the previous steps are successfully completed by the end-user, the actual deployment of the application takes place by clicking on the instantiate button. The deployed network app appears in the “instances” view of the GUI, alongside with certain metrics and runtime logging information. The following picture depicts a successful deployment of an application on top of the programmable resources. Logging informa-

tion is provided with details regarding the status of the application graph, along with some basic monitoring metrics per application component.

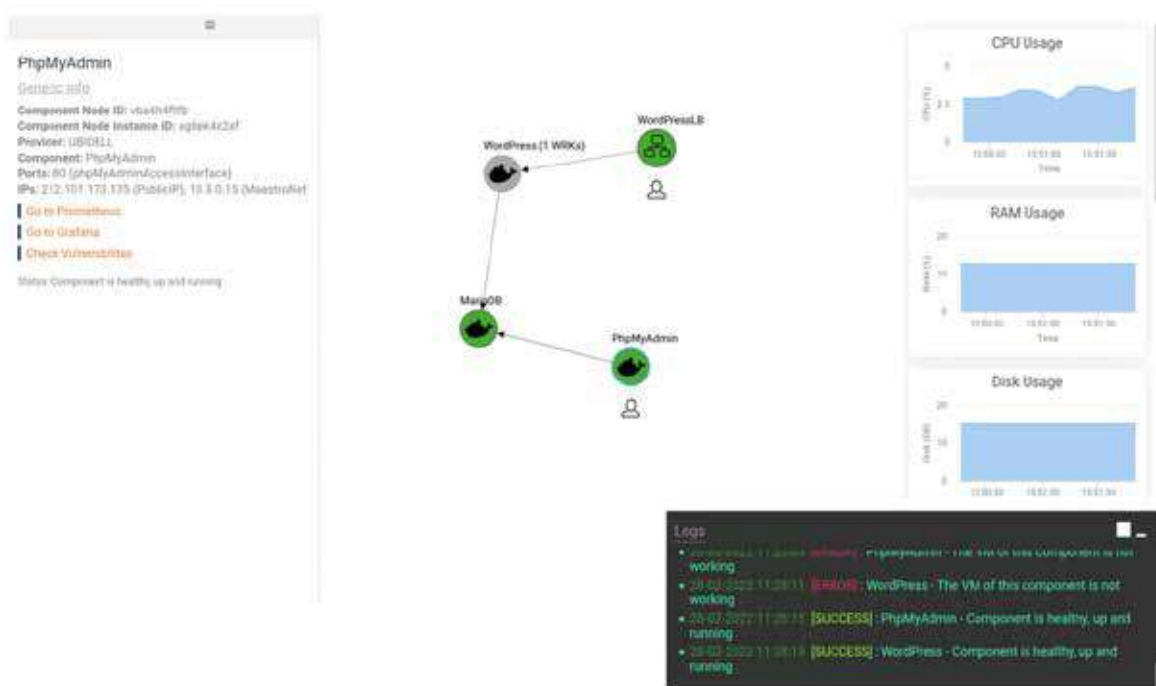


Figure 11. Application deployment just realized and runtime logs for the application's deployment

5. NETWORK APP STARTER KIT

5G-IANA has created network app "Starter-kits" specifically designed to aid in the development of advanced Automotive Vertical Services. These kits are intended to support the creation of Vertical Services within identified service categories by providing a baseline set of AFs/NFs (atomic components) for deployment. By utilizing these kits, Service Creators and Providers can better leverage the resources available through the 5G infrastructure, including the ability to orchestrate and run applications on Far-edge resources like OBUs and RSUs.

In addition to facilitating the development of advanced Automotive Vertical Services, the network app "Starter-kits" also aim to provide Verticals with the necessary knowledge to understand the specific purpose and usage of low-level functionalities. This is particularly important because the deployment of certain AFs/NFs may be required to consume and forward information on top of an OBU, such as Intelligent Transport Systems (ITS) communication functions.

As each Vertical has unique needs and requirements, 5G-IANA offers a variety of open-source network app "Starter-kits," each designed to support the roll-out of 5G-IANA and third-party UCs. These kits are available as ready-to-use network app Packages that contain all the relevant information necessary for their usage in specific contexts/scenarios.

For example, Figure 13 provides an illustration of network app "Starter-kits" for a manoeuvres' coordination service, highlighting two different kits, each designed to aid in the implementation of specific functionalities. The AFs highlighted in purple are customizable and can be integrated by experimenters and third parties looking to provide a specific logic/algorithm for the Manoeuvres Planning functionality.

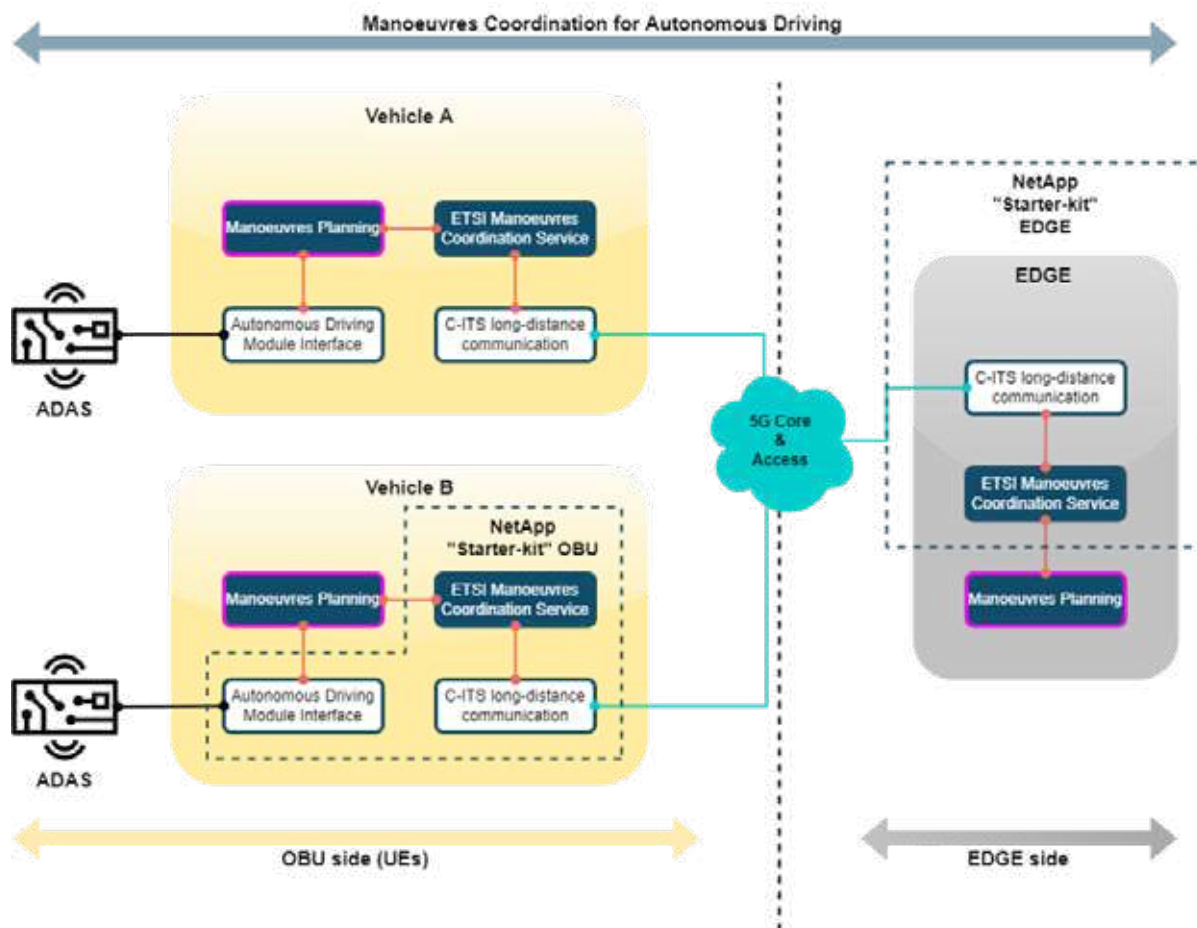


Figure 11 Manoeuvres Coordination for Autonomous Driving network app "Starter-Kits" Example

Overall, the integration of network app "Starter-kits" aims to streamline the development of advanced Automotive Vertical Services and enhance the utilization of resources available through the 5G infrastructure.

The list of the network app starter-kits is available in Table 1.

Table 1 Network app Starter kit

Categories	Network App name
Telematics Services, Infotainment Systems, Autonomous Vehicle Networks	Real Time Stream Delivery
Telematic Services, Autonomous Vehicle Networks	Object Detection Stream and Data Delivery

Categories	Network App name
Telematic Services, Vehicle-to-everything Communication, Autonomous Vehicle Networks	AGV Data Processing, Communication and Control
Telematic Services, Vehicle-to-everything Communication, Connected Car Platforms, Autonomous Vehicle Networks	Remote Driving
Telematic Services, Vehicle-to-everything Communication, Connected Car Platforms, Autonomous Vehicle Networks	MCAD Edge Node
Telematic Services, Vehicle-to-everything Communication, Connected Car Platforms, Autonomous Vehicle Networks	MCAD Rover Node
Infotainment Systems	AI Enhanced Video Stream Delivery
Infotainment Systems	Video enabled VR Client
<u>Autonomos</u> Vehicle Networks	Active Network Monitoring Module
Infotainment Systems, Autonomous Vehicle Networks	Virtual Bus Tour – UC3
Infotainment Systems	AR App
Infotainment Systems	AR streaming application
Telematic Services, Vehicle-to-everything Communication, Connected Car Platforms	Real-time Hazardous driving event detection
Telematic Services, Vehicle-to-everything Communication, Connected Car Platforms	Aggregated Hazardous driving event detection
Telematic Services, Vehicle-to-everything Communication, Connected Car Platforms	Aggregated Hazardous driving event detection
Telematic Services, Vehicle-to-everything Communication, Connected Car Platforms	Real-time vehicle trajectory prediction
Telematic Services, Vehicle-to-everything Communication, Connected Car Platforms	Hazardous driving event notification
Telematic Services, Cybersecurity Solutions, <u>Autonomos</u> Vehicle Networks	Predictive QoS
Cybersecurity Solutions, <u>Autonomos</u> Vehicle Networks	Obtain Training Data

Categories	Network App name
Vehicle-to-everything Communication, Connected car platforms, <u>Autonomos Vehicle Networks</u>	DML Training
Telematic Services	Video stream delivery
Telematic Services, Vehicle-to-everything Communication, Connected Car Platforms, <u>Autonomos Vehicle Networks</u>	Environmental/IoT monitoring
Cybersecurity, Vehicle-to-everything Communication, Connected Car Platforms	Simulator of ETSI Cooperative Awareness Service
Telematic Services, Vehicle-to-everything Communication, Connected Car Platforms <u>Autonomos Vehicle Networks</u>	Vehicle monitoring
Telematic Services	ITS communication nApp
Cybersecurity	Network Monitoring

6. STEP-BY-STEP ONBOARDING GUIDE

This is a step by step guide to assist with the onboarding process of software artefacts to the 5G-IANA platform and with network app composition. These steps are:

6.1. Step#1 NF/AF Development

A working software service is expected by the experimenter.

Step Completion Requirements: Service Chain can be replicated in a lab environment.

6.2. Step#2 NF/AF Containerization and orchestration

The service should be provided in the form of multiple Dockerfiles & orchestrated through docker-compose.yml files.

Step Completion Requirements: Service Chain can be replicated in a lab environment.

6.3. Step 3PRE-I Prerequisite: Communication with 5G-IANA - GitLab access & GitLab Repository Creation

1. Access to GitLab is required in order to be able to push your images to the 5G-IANA platform.
2. A GitLab Repository per NF/AF should be created.
3. Projects should follow the following demo network app template structure. The demo network app template provides a snapshot adaptation of an open source dockerized service; The snapshots can be found in their relevant branches (initial, demo, master). Moreover, some helper scripts are provided to i) retrieve the latest uploaded tag of a service, ii) manually push all services contained in a docker-compose to the specified repo, respecting the version tagging and incrementing it (Note: The docker-compose should have been built beforehand and the images should be present in the local repository).

Step Completion Requirements: You have access to your projects GitLab. You can push to both dev & master branches of your project.

6.4. Step 3PRE-II Prerequisite: Communication with the targeted 5G-IANA testbed owner - VPN Access

Access to the testbed is required to access the 5G-IANA Composer GUI, and to be able to validate the network app deployment prior to field validation.

Relevant Partner - NOKIA/Testbed Slovenia

NOKIA Testbed Useful Internal IPs

- Docker Registry: 192.168.100.2
- Composer GUI: 192.168.100.3/dashboard

Step Completion Requirements: You have access to the testbed via VPN or any other network configuration that is decided between the two parties (Experiment owner - Testbed Owner). You can access the network location provided by the Testbed Owner.

6.5. Step 3PRE-III Prerequisite: Request Composer-GUI credentials

Composer-GUI utilizes a role-based login mechanism. Credentials to enter the composer GUI are required and have to be obtained by 5G-IANA by contacting open-call@5g-iana.eu for the targeted testbed's composer GUI installation.

Step Completion Requirements: You can login to the composer GUI.

6.6. Step 3PRE-IV Prerequisite: Add docker-registry self-signed certificate

In order to be able to pull from the docker-registry, the installation of the self-signed certificate is required as you need to ensure Docker trusts the certificate. You can do this by copying the certificate file to the Docker certificates directory and restarting Docker. Otherwise, you can edit your docker engine configuration by adding the following parameters to your configuration file (windows) / daemon.json (nix system) file:

```
"insecure-registries": [  
  "192.168.100.2:5000"  
]
```

Windows:

For Windows-based machines please follow this guide: [windows - guide](#). Note, Windows docker desktop does not have a specific certificates' folder for docker, it uses the trusted root authorities (mmc).

Ubuntu:

For Debian-based machines, please follow this guide: [ubuntu - guide](#). Sometimes it is required to add the certificate under the docker tracked registry as well. Copy the certificate to the Docker certs directory:

```
sudo mkdir -p /etc/docker/certs.d/192.168.100.2:5000
```

```
sudo cp /path/to/your/certificate.crt /etc/docker/certs.d/192.168.100.2:5000/
```

```
sudo service docker restart
```

Now you should be able to login successfully

```
docker login https://192.168.100.2:5000
```

IT managed environment

In case your IT department is managing the certificates for your machine please contact the IT administration or create a VM.

Step Completion Requirements: You are able to successfully login to the docker-registry (for NOKIA : <https://192.168.100.2:5000/v2>) with the following command:

```
docker login https://192.168.100.2:5000/
```

6.7. Step#3 5G-IANA Onboarding preparation

Current limitations:

- Total NF/AF/pod name should not exceed 64 characters:
 - The name is calculated as: network app name + NF/AF name + 4char NF instance+ 10 character hexcode.
- Exposed interface: should be defined to all exposed interfaces and currently the same radio slice type has to be used.
- GPU is not exposed to the container.
- Volume Binding is not supported
- ConfigMaps are not supported.

6.7.1. Step#3.1: Network app image creation & upload to Central Repository

As described in Chapter 3, the GitLab CI/CD pipeline automatically uploads (pushes) the generated images to the 5G-IANA docker registry.

In order to assist with this process, a demo application structure can be found there. This GitLab project is provided as an initial stepping point to adapt your descriptors and create the appropriate gitlab-ci.yml.

6.7.1.1. Step#3.1.1 Clone the network app repository

Git clone – replace \$PATH_TO_NAPP_REPOSITORY with your created GitLab network app repository that you would like to work on:

- HTTPS: [https://gitlab.ubitech.eu/5g-iana/\\$PATH_TO_NAPP_REPOSITORY](https://gitlab.ubitech.eu/5g-iana/$PATH_TO_NAPP_REPOSITORY)
- SSH-KEY: [git@gitlab.ubitech.eu:5g-iana/\\$PATH_TO_NAPP_REPOSITORY](ssh://git@gitlab.ubitech.eu:5g-iana/$PATH_TO_NAPP_REPOSITORY)

Step Completion Requirements: You have to clone the repository to your local machine.

6.7.1.2. Step#3.1.2: Update for target NF/AF

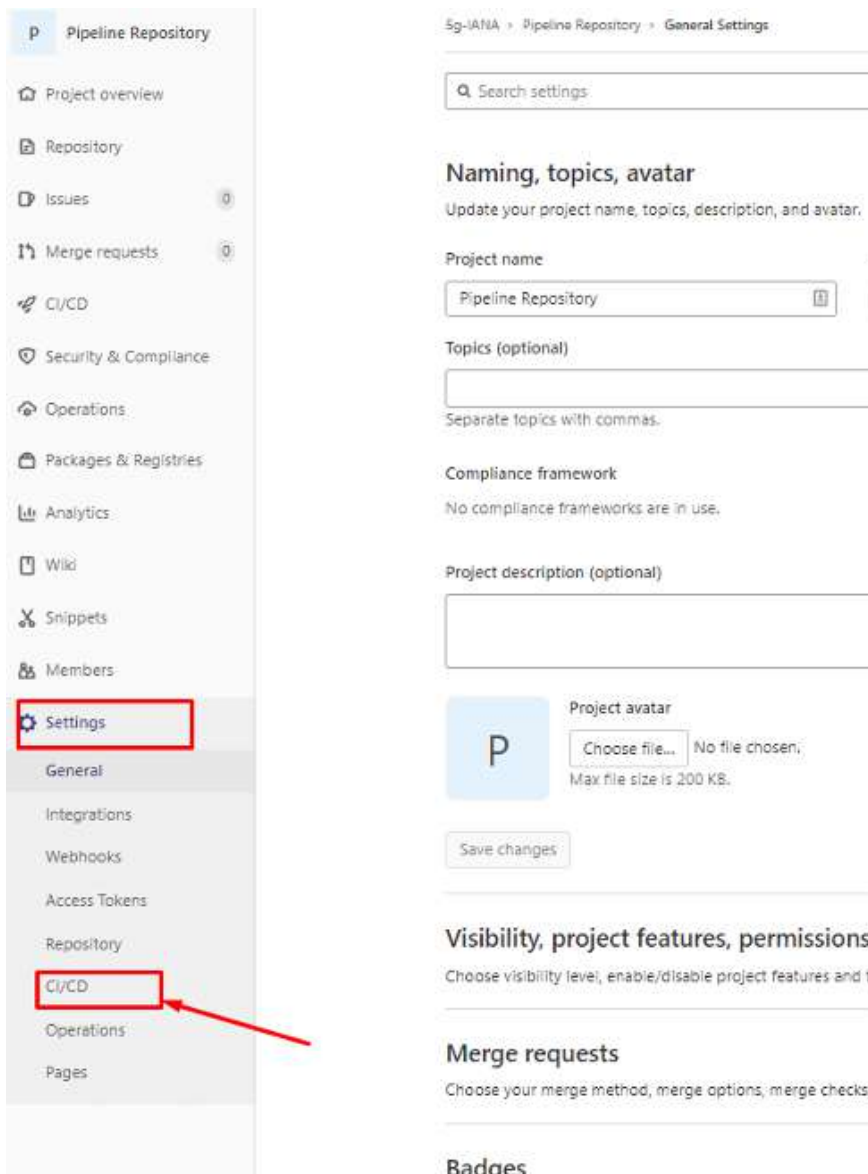
Add your network app files under the local repository. Note: The Dockerfile or docker-compose of the network app should be at the root of the Project.

6.7.1.3. Step#3.1.3: Create gitlab-ci.yml & Configuration of GitLab Repository

There are 2 ways to create a gitlab-ci.yml:

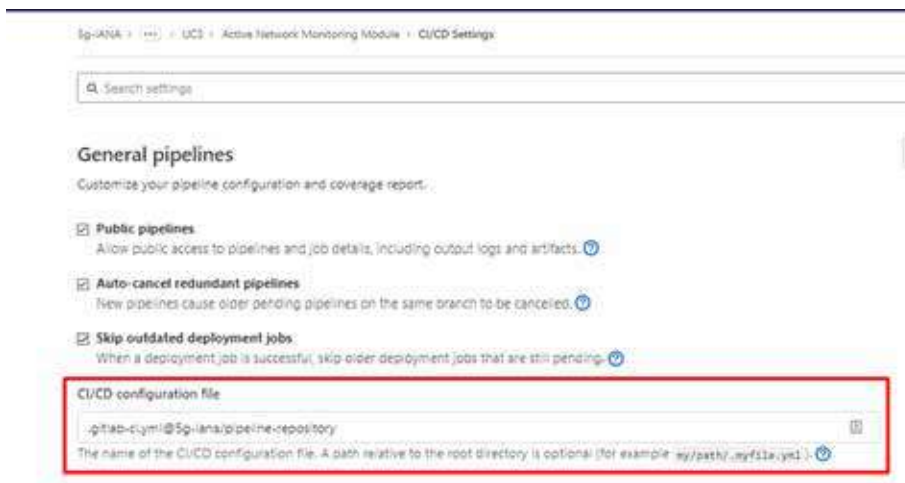
- i) Outline the whole process in a single file .
- ii) Following the provided GitLab pipeline template.
 - 1) Set the following GitLab variables:
 - a) PARTNER_NAME
 - b) COMPONENT_NAME
 - c) COMPONENT_VERSION

In both cases the variables can be set to the following location: Repository/Settings/-CI/CD.





In case i) where the provided pipeline repository GitLab CI is selected you have to link to it here:



Otherwise follow this template for your projects:

E.g.: variables:

variables:

```

COMPONENT_NAME: "module_name_lowercase"
DOCKER_TAG_PREFIX: "partner_name_lowercase"
CI_REGISTRY_IMAGE: 192.168.100.2:5000
KANIKO_IMAGE: "gcr.io/kaniko-project/executor:v1.9.0-debug" #USE THIS ONE
COMPONENT_VERSION: "develop"
DOCKERFILE_PATH: "Dockerfile"
DOCKER_CONFIG: "/kaniko/.docker/"
    
```

stages:

- build
- deploy

build:

stage: build

image:

name: \$KANIKO_IMAGE

entrypoint: [""]

only:

- devCycleA

script:

- echo "Building \$COMPONENT_NAME"

- /kaniko/executor --context \$CI_PROJECT_DIR \

- dockerfile \$DOCKERFILE_PATH \

- destination \$CI_REGISTRY_IMAGE/\$DOCKER_TAG_PREFIX/\$COMPONENT_NAME:\$CI_COM-

MIT_TAG --no-push

deploy:

before_script:

- |

echo "-----BEGIN CERTIFICATE-----"

MIICIDCCAf2gAwIBAgIUac+ko3JCbLkKofsw4zZ7jmK2hWUwDQYJKoZIhvcNAQEF

BQAwfDELMAGGA1UEBhMCWFgxDAAKBgNVBAGMA04vQTEMMAoGA1UEBwwDTi9BMSAw

HgYDVQKDBdTZWxmLXNpZ25lZCBjZXJ0aWZpY2F0ZTEvMC0GA1UEAwwmMTkyLjE2

OC4xMDAuMjogU2VsZi1zaWduZWQgY2VydGlmYWVhdGUwHhcNMjMwNzI4MDcyOTAz

WhcNMjQwNzI3MDcyOTAzWjB8MQswCQYDVQQGEWJYWDEEMMAoGA1UECAwDTi9BMQww

CgYDVQQHDANOL0ExIDAeBgNVBAoMF1NlbgYtc2lnbmVklGNlcnRpZmljYXRIMS8w

LQYDVQQDDCYxOTluMTY4LjEwMC4yOiBTZWxmLXNpZ25lZCBjZXJ0aWZpY2F0ZTCB

nzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEAAniESM4TXypLuqkkkXe6wdAlVo/ln

iaPVIV6WH64dab8s5idpk16ThvkpuON6czF8oQtEC5OCWvHUmPf8wu29kC7s2Gop

8yeWlu8BG0fD28qDxhURbDoxqlrbEVQN3r+ekYKIE83yxM4Zay+r1+s1fzYkf5q

/O0n8WV74Sf4/tkCAwEAAAMTMBEwDwYDVR0RBAgwBocEwKhkAjANBgkqhkiG9w0B

AQUFAAOBGCQJ5618apVWYG2+mizc3HgDgOrY88wUdXOnpejj5r6YrhaQp/vUHGMY

Tv5E3G+IytNJDzqfjMNzXGzK6A7D66tU+MuO7yHX7a370JyBF/5rc0YQM+ygllr

2WQ58cXzY9INB2l+JTbzDXA+gL7EvGzu/8CW0Ud9RabSTRRz6hd2OQ==

-----END CERTIFICATE-----" >> /kaniko/ssl/certs/additional-ca-cert-bundle.crt

stage: deploy

image:

name: \$KANIKO_IMAGE

entrypoint: [""]

only:

- devCycleA

- master

- merge_requests

script:

- echo "Building \$COMPONENT_NAME"

- echo {"auths":{"192.168.100.2:5000/v2/":{"username":"5g-iana","password":"5g-iana"}}}

/kaniko/docker/config.json

- /kaniko/executor --skip-tls-verify --context \$CI_PROJECT_DIR \

- dockerfile \$DOCKERFILE_PATH \

- destination \$REGISTRY_PREFIX/\$DOCKER_TAG_PREFIX/\$COMPONENT_NAME:\$CI_COMMI-

T_TAG

Step Completion Requirements: You have created the gitlab-ci.yml

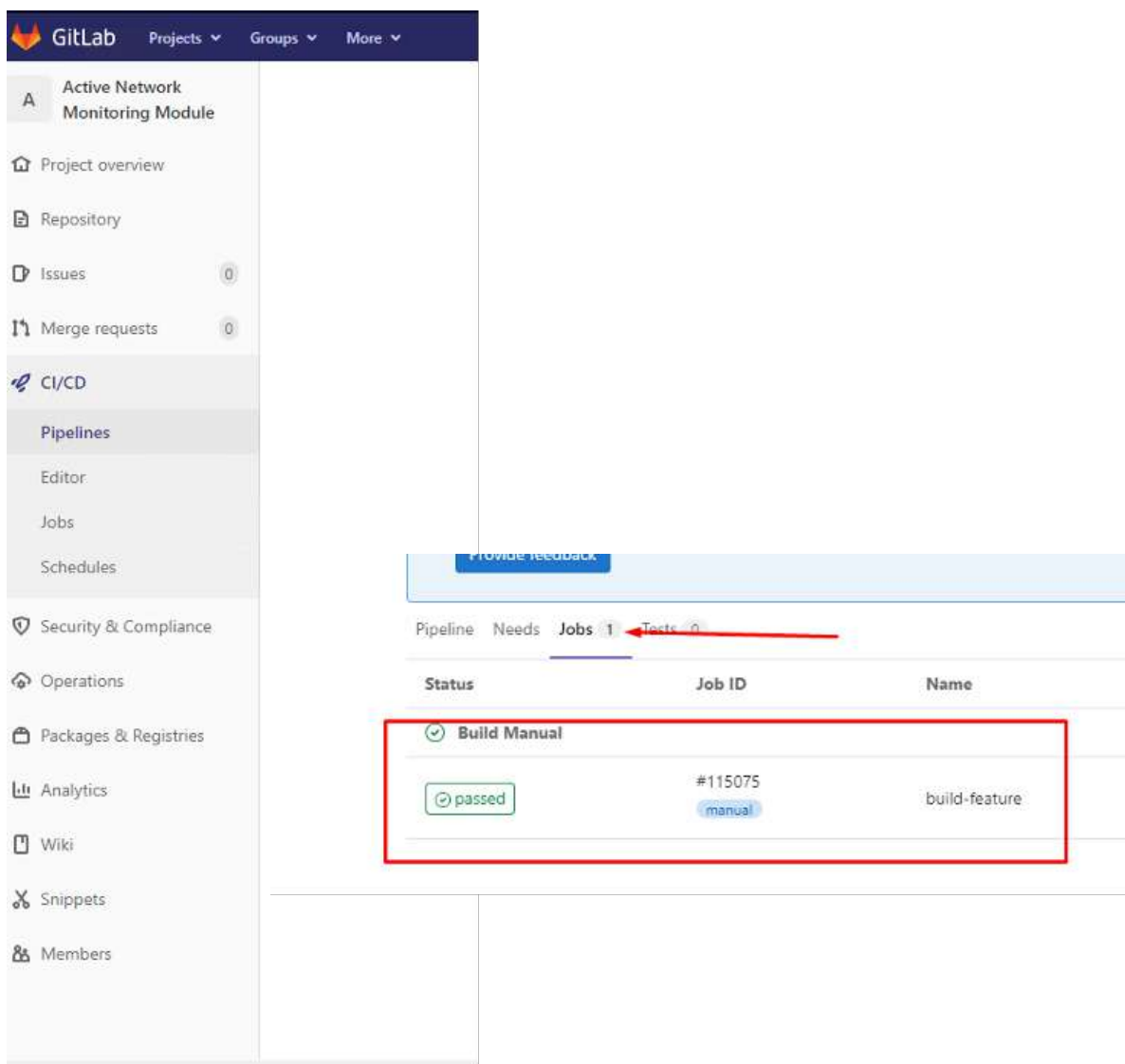
6.7.1.4. Step#3.1.4: Push changes to remote repository

Push the changes to the remote repository. There are two branches named master & dev. The pipeline requires to push and validate initially at the dev branch and push the final version to the master branch through a merge request (pull request if you are from a GitHub background).

```
git add *
git commit -m "Commit msg"
git push origin --set-upstream origin dev
```

Validate the pipeline stage by checking the following Job status sections (more can be found: <https://docs.gitlab.com/ee/ci/pipelines/>) – You can inspect the ci-cd logs here.

Step Completion Requirements: The GitLab CI/CD pipeline has been completed successfully.



6.7.1.5. Step#3.1.5: Merge Request to master branch

You will have to move your changes to the master branch. This can be achieved through a merge request (from GitLab GUI) or by pushing directly the master branch (similar to Step 3.1.3: Push changes to remote repository)

Step Completion Requirements:

- i) The pipeline has succeeded for master branch.
- ii) You can see your image in the list provided by running the following command from a terminal that has access to the network of the targeted testbed (e.g., through VPN):

Show all available Images:

```
curl -u 5g-iana:5g-iana -X GET https://192.168.100.2:5000/v2/_catalog -k
```

Show all available Tags for repository:

```
curl -k -u 5g-iana:5g-iana -X GET https://192.168.100.2:5000/v2/<repository name taken from the previous command>/tags/list
```

- iii) You can see your image as listed at the 5G-IANA composer GUI (for NOKIA testbed: Composer GUI: 192.168.100.3/dashboard)

E.g.:

```
git checkout origin master
```

```
git merge origin dev
```

```
git commit -m "merge with master"
```

```
git push --set-upstream origin master
```

```
curl -u 5g-iana:5g-iana -X GET https://192.168.100.2:5000/v2/_catalog -k
```

6.7.2. Step#3.2: Descriptor Adaptation for 5G-IANA platform

This step can be completed automatically via provision of the maestro-compose.yml to GitLab CI/CD pipeline, otherwise the option to create the network apps through the composer GUI (192.168.100.3/dashboard), is provided. Prepare the descriptors for 5G-IANA platform onboarding, according to the onboarding guide where you will gain access: https://gitlab.ubitech.eu/5g-iana/use-cases/onboarding_guide.

Step Completion Requirements: The created/edited descriptors can be ingested by the 5G-IANA platform. (Can be verified through the 5G-IANA GUI, hosted at the testbed).

6.7.2.1. Step#3.2.1 Create a maestro-compose.yml to your local repository.

Create a Maestro-compose.yml in your local repository file by following the onboarding_guide. The following restrictions apply (for a full reference follow the onboarding_guide):

- i) All configurations should be passed through environment variables (if it is not possible you will probably have to alter the interface, contact 5G-IANA for following assistance).
- ii) Static IPs are not supported, the IP configuration should be provided through environment variables (e.g `WORDPRESS_DB_HOST : "MariaDB" # *`).

Step Completion Requirements: The network app is present in the 5G-IANA Composer GUI.

6.8. Step#4 Blueprint Creation & Version verification of uploaded artefacts

This is a validation step to evaluate if the onboarding prerequisites were completed successfully.

6.8.1. Step#4.1 Version verification

Verify that the images present at the docker registry are the correct ones. This can be achieved by running:

```
` curl -u 5g-iana:5g-iana -X GET https://192.168.100.2:5000/v2/_catalog -k `
```

and validating the docker registry metadata.

Step Completion Requirements: The network app developer validates if the image is correct.

6.8.2. Step#4.2 Blueprint creation

Blueprint is the internal format of the composed network app, containing all its metadata as described in Section 4.4, this is encompassed in a blueprint.json file.

Step Completion Requirements: The Blueprint creation success is reported by 5G-IANA. A report on the components that failed to be connected will be provided.

6.9. Step#5 5G-IANA Platform Onboarding

Add the validated network application to the 5G-IANA Platform either through the maestro-compose.yaml process provided by 5G-IANA as described previously, or through the compose GUI.

Step Completion Requirements: The service chain can be completely replicated by using the GUI or the maestro-compose.yaml. The service chain creation is completed without any problems and all dependencies are met.

6.10. Step#6 Complete Service Chain Validation in regards to proper communication and functionality of all images on the testbed

The whole service chain can be replicated using the 5G-IANA platform. After this you are ready for experimentation, as everything is ready to be tested.

Step Completion Requirements: Connection and communication of each NF/AF has been verified and possibly a demo version of the service chain (if possible) has been evaluated.

REFERENCES

5G-IANA - D2.1 - Specifications of the 5G-IANA architecture

5G-IANA - D7.7 - Exploitation plan

<https://docs.docker.com/registry/spec/api/>

https://docs.gitlab.com/ee/ci/docker/using_kaniko.html

VITAL5G - D2.1 Initial NetApps blueprints and Open Repository design

5G ERA, “D4.1: 5G-ERA Middleware initial version”, June 2022

<https://12factor.net/>

<https://quarkus.io/>

<https://spec.openapis.org/oas/v3.0.1>

ANNEX – NETWORK APP TEMPLATE

```

{
  "id": "Long",
  "netAppPackageId": "UUID",
  "name": "String", - Required
  "publicApplication": "boolean", - Required
  "description": "String",
  "version": "String", - Required
  "componentNodes": [{
    "componentNodeId": "Long",
    "name": "String",
    "component": {
      "id": "Long",
      "name": "String", - Required if component is
present
      "architecture": "String",
      "iconBase64": "String",
      "iconContentType": "String",
      "iconFilename": "String",
      "iconContent": "String[]",
      "dockerImage": "String", - Required if com-
ponent is present
      "dockerRegistry": "String", - Required if
component is present
      "dockerCustomerRegistry": "boolean", -
Required if component is present
      "dockerUsername": "String", - Required if

```

component is present

"dockerPassword" : "String", - Required if

component is present

"organization" : "String",
"dockerCredentialUsing" : "String", - Required if

component is present

"publicComponent" : "boolean", - Required if

component is present

"allowEdit" : "boolean",
"allowDelete" : "boolean",
"dateCreated" : "String",
"lastModified" : "String",
"exposedInterfaces" : [{
 "interfaceID" : "Long",
 "name" : "String", - Required

if exposedInterfaces is present

"port" : "String", - Required if

exposedInterfaces is present

"vna" : "String",
"interfaceType" : "String", -

Required if exposedInterfaces is present

"transmissionProtocol" :

"String", - Required if exposedInterfaces is present

"dateCreated" : "String",
"lastModified" : "String"

}},
"requiredInterfaces" : [{
 "graphLinkID" : "Long",
 "friendlyName" : "String",
 "interfaceID" : "Long",
 "dateCreated" : "String",
 "lastModified" : "String",
 "interfaceObj" : {

"Long",

"interfaceID" :

"String",

"name" :

"String",

"port" :

pe" : "String",

"vna" : "String",
"interfaceType"

sionProtocol" : "String",

"transmissionProtocol"

: "String",

"dateCreated"

"String"

"lastModified" :

```

    }

  ]],
  "requirement": {
    "requirementID": "Long",
    "cpus": "Integer",
    "ram": "Float",
    "storage": "Float",
    "hypervisorType": "String",
    "gpuRequired": "Boolean",
    "dateCreated": "String",
    "lastModified": "String",
  },
  "healthCheck": {
    "healthCheckID": "Long",
    "name": "String",
    "httpURL": "String",
    "args": "String",
    "interval": "Float",
    "dateCreated": "String",
    "lastModified": "String",
  },
  "volumes": [{
    "volumeID": "Long",
    "component": "String",
    "dockerPath": "String",
    "dateCreated": "String",
    "lastModified": "String",
    "file": "Boolean",
  }],
  "labels": [{
    "labelID": "Long",
    "name": "String",
    "dateCreated": "String",
    "lastModified": "String",
  }],
  "softwareLicenses": {
    "id": "Long",
    "softwareLicenseID": "String",
    "openLicense": "Boolean",
    "licenseFile": "String",
  }
}

```

- Required if inserting each single component via frontend

Required if inserting each single component via frontend

Required if inserting each single component via frontend


```

is present
    "openLicense" : "Boolean", - Required if softwareLicense is
present
    "licenseFile" : "String", - Required if softwareLicense is
present
    "type" : "Enum_String" "[APPLICATION_PROPRIETARY,
ISTANCE_BASED, TIME_BASED, FLAT]"
    },
    "required5GCoreServices" : {
        "fiveGServiceSpecid" : "String",
        "name" : "String",
        "version" : "String",
        "function" : "Enum_String", "[NWDAF, LCS]"
        "mandatory" : "boolean"
    }
}

```