FUDGE-5G

FUlly DisinteGrated private nEtworks
for 5G verticals

Deliverable 2.5

# Technology Components and Platform – Final Release

Version 1.0

Work Package 2

| | |
|---|---|
| Main authors | Sebastian Robitzsch (IDE) |
| Distribution | Public |
| Delivery date | 30 November 2022 (M27) |
| Delivered date | 12 December 2022 (M28) |



Partners

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement № 957242.

# Disclaimer



## Project details

**Project title:** FUlly DisinteGrated private nEtworks for 5G verticals
**Acronym:** FUDGE-5G
**Start date:** September 2020
**Duration:** 30 months
**Call:** ICT-42-2020 Innovation Action

## For more information

**Project Coordinator**
Prof. David Gomez-Barquero
Universitat Politecnica de Valencia
iTEAM Research Institute
Camino de Vera s/n
46022 Valencia
Spain

http://fudge-5g.eu
info@fudge-5g.eu

## Acknowledgement

FUDGE-5G has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement Nº 957242. The European Union has no responsibility for the content of this document.

## Abstract

The last FUDGE-5G WP2 deliverable mainly serves as a reference document for Enterprise Service vendors who aim to develop software that is orchestrated over a cloud-native Service-Based Architecture platform. While offering detailed platform API documentation, D2.5 also describes experiences of technical challenges with Enterprise Service applications regarding their cloud-nativeness. The deliverable also describes the testbed set-up where all platform components have been integrated with a selected set of Enterprise Services. Lastly, the integration efforts of IP Multicast on the 5G user plane are described in this deliverable.


# Versioning and Contributions

## Versioning

| # | Description | Contributors |
|---|---|---|
| 0.1 | Table of contents | IDE |
| 0.2 | SFV Orchestrator API description | IDE |
| 0.3 | Changes to SFV resource descriptor definition | IDE |
| 0.4 | SFV API refinement, VAO documentation | IDE, UBI |
| 0.5 | VAO documentation draft | UBI |
| 0.6 | VAO documentation update based on IDE feedback | UBI, IDE |
| 0.7 | VAO documentation update and feedback | UBI, ONE |
| 0.8 | Integration Challenges | IDE |
| 0.9 | 5G Multicast Broadcast section, Cloud-Native orchestration of 5G Cores, VAO section update | UPV, IDE, UBI |
| 1.0 | Final version | IDE |

## Contributors

| Partner | Authors |
|---|---|
| IDE | Sebastian Robitzsch, Mohamad Kenan Al-Hares |
| UBI | Thanos Xirofotos |
| ONE | André Gomes |
| UPV | Carlos Barjau, Borja Inesta |

## Reviewers

| Reviewer | Partner |
|---|---|
| Hergys Rexha | AAU |
| Sébastian Lafond | AAU |

# Acronyms

| | |
|---|---|
| 5GC | 5G Core |
| API | Application Program Interface |
| CRUD | Create Read Update Delete |
| NF | Network Function |
| SC | Service Chain |
| SCC | Service Chain Controller |
| SCP | Service Communication Proxy |
| SF | Service Function |
| SFE | Service Function Endpoint |
| SFEC | Service Function Endpoint Controller |
| SFPR | Service Function Package Repository |
| SFV | Service Function Virtualisation |
| SFVO | Service Function Virtualisation Orchestrator |
| VAO | Vertical Application Orchestrator |

## Executive Summary

The specification of 3GPP's Service-Based Architecture (SBA) in Release 15 and 16 were driven by the desire of the operators to deploy Core Networks as services and in a multi-vendor fashion. This demand is deeply rooted in the necessity to build distinct networks that are even more fully customisable for a broader use case applicability than generic internet access and calling. However, because of the way 4G was designed, it did not allow a rapid deployment and customisation of a (Core) Network to meet the requirement of verticals such as education, media, health, industry, or emergency first response sectors. The promises of 5G are put under trial in FUDGE-5G with WP2 covering the development and initial integration efforts for a set of technologies.

On the premise of unifying key components of a cloud-native system that must reside outside of the actual service, i.e. routing, orchestration and telemetry, FUDGE-5G has positioned a platform layer between the service and the infrastructure which covers the unified system aspects in their functionality. This deliverable partially serves as a reference document, describing the Application Program Interfaces of the platform for Enterprise Service developers. Furthermore, it also provides a detailed description of the challenges the project had to overcome regarding the cloud-native software design of Enterprise Services and operating it over the platform. Under the commonly used phrase at industry events "cloud meets telecom", it is no surprise that 5G introduced new challenges to vendors, as they depart from a 4G model where cloud-native procedures and multi-vendor environments were not a requirement.

Also, a detailed description of integrating IP multicast on the 5G user plane is provided in this document, demonstrating the flexibility 5G enables based on its system architecture.

# TABLE OF CONTENTS

# 1 Introduction

Over the course of the last 28 months, FUDGE-5G has continuously worked on the development and component integration based on the work in Work Package 1 (WP1). The resulting deliverables D1.1 [FUD11] focussing on use cases and the validation framework, combined with the architectural work published in D1.3 [FUD13], marks the foundation for the work in Work Package 2 (WP2).

Deliverable D2.5 shall be used in conjunction with D1.3 (Final FUDGE-5G Platform Architecture Components and Interface) and complements the system description with detailed interface specifications for Enterprise Service developers aiming at developing software in a cloud-native fashion for the 5G telco domain. As the FUDGE-5G project unifies routing, orchestration and telemetry as a single Platform-as-a-Service (PaaS) offering, the benefits of such unification shall be explored in more depth first. D2.2 offers a comprehensive discussion around the unification aspects with benefits in the resource scheduling domain. Like public cloud offerings of hyperscalers, the service developers must not add code inside their services to perform routing, orchestration (as in scaling) or telemetry (apart from reporting data points of interest). Externalisation and offering the externalised functionality in a programmable and tenancy-enabled fashion is crucial for a widespread acceptance of this methodology and is commonly described as the cloud-native proposition. In order to achieve such clear separation of platform and service layers, D2.5 provides the detailed list of platform components and their interfaces that enable that. Furthermore, D2.5 also provides a comprehensive description of the testbed where all platform components and Enterprise Services were integrated and tested.

Directly linked with the design patterns for cloud-native 5G NFs published in D2.4 [FUD24], D2.5 provides experiences in integrating the various 5GCs over the FUDGE-5G platform.

Due to the complexity of integrating IP multicast on the 5G User Plane, D2.5 also provides the details of this work instead of D2.2.

# 2 Platform Components and their Interfaces

This section presents the platform APIs Enterprise Service vendors may utilise in order to get their application orchestrated. The section is structured along the FUDGE-5G platform components, i.e. routing, orchestration and telemetry.

## 2.1 Service Routing

When using the SFVO or VAO to orchestrate Enterprise Services, there is no need to communicate with the Service Routing (aka Service Communication Proxy (SCP)) directly. However, the SCP of the FUDGE-5G platform exposes an open API to register and deregister service endpoints using their FQDN and IP address. When orchestrating an Enterprise Service, (depending on whether the application is a 5GC NF or vertical service) the SFVO or VAO utilise the interface to inform the SCP that a new service endpoint is available and which FQDN it serves.

As described in detail in D1.3 [FUD13], the SCP has an internal component entitled Service Proxy Manager (SPM) which offers the registration API. The SPM is available from any IP endpoint which has received an IP address over the SCP on the control plane. The FQDN to use is “spm” on Port 8080. The Create Read Update Delete (CRUD) API is described in further detail below.

### 2.1.1 Registration

The registration API offers to add a new FQDN to an existing IP address using the following URI:

```
curl -X POST spm/flips/nm/domain/<fqdn>/srv/<ip>
```

where <ip> is the IP address of the endpoint and <fqdn> the FQDN to be registered.

### 2.1.2 Deregistration

The deregistration follows the same semantic as the registration with the difference of using the DELETE HTTP method:

```
curl -X DELETE spm/flips/nm/domain/<fqdn>/srv/<ip>
```

## 2.2 Service Function Virtualisation

This section describes the RESTful interfaces of the Service Function Virtualisation Orchestrator of the eSBA FUDGE-5G platform used to orchestrate 5GCs in a cloud native fashion. The workflow for achieving that is illustrated in Figure 1 and depicts responsibilities/actions for 5GC vendors and platform admins. Also, the figure illustrates

activities that are conducted outside the SFVO and the ones that utilise APIs exposed by the SFV Orchestrator.

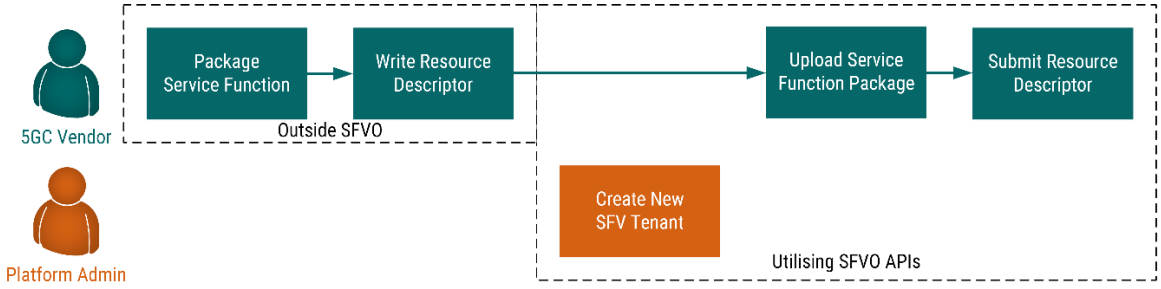


*Figure 1: Workflow for Provisioning Service Chain via Service Function Virtualisation Orchestrator*

The "onboarding" process for 5GC vendors is provided in form of an online documentation [GLT25] which comes with a set of bash scripts to package one or more software executables into a Linux Container (LXC). These procedures are executed outside of a deployed SFVO. Along with the packaging of a Service Function, the 5GC vendor is asked to write a Resource Descriptor that defines how the Service Chain is composed of (which Service Functions) and what properties each SF has, (e.g. compute, memory, storage). The Resource Descriptor also allows the 5GC to define which SF is provisioned into which state onto which Service Host. More information about this can be found in the next section.

Upon the completion of the "offline" actions, the platform administrator creates a new user within the SFVO for the 5GC isolating all 5GCs from each other. Once done, the 5GC vendor can upload their packaged Service Functions to the SFP repository (SFPR).

The implemented SFV architecture is provided in Figure 2 and illustrates the 5GC vendor on the top left interfacing with the Service Function Package Repository (SFPR) and the Service Chain Controller via the Ssfpr1 and Sscc1 interfaces, respectively. These two interfaces are described in further detail in the following two sections.

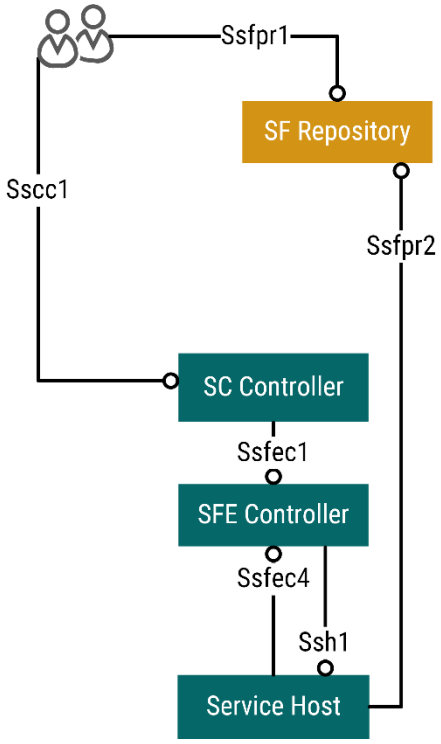


*Figure 2: Implemented Service Function Virtualisation Architecture.*

The interfaces 5GC vendors utilise are Ssfpr1 and Sscc1 which are documented in further detail below. The other interfaces are SFV internal and are not described in their entirety.

A pre-recorded SFV Tour is available on YouTube as an unlisted video [YTSFV].

### 2.2.1 User Management Interface Specification

Both SCC and SFPR offer the same user management API for JSON Web Token (JWT)-based authorisation. When the platform administrator created a new tenant account, SCC and SFPR follow the same procedures for authentication. Each request to the SCC and SFPR API requires a valid time limited JWT.

| Resource | HTTP Method | Description |
|---|---|---|
| /login/<username> | POST | Obtain JWT for using any of the APIs of the SCC or SFPR. |

#### 2.2.1.1 /login/<username>

This resource accepts HTTP request with the method POST and a JSON-encoded payload providing the password to the username in the resource. An example HTTP transaction is provided below using CURL:

```
~$ curl -i -X POST scc:8080/login/fudge -H"Content-type: application/json" \
-d"{\"password\":\"fudge-5g-2020\"}"
                                    HTTP/1.0 200 OK
```

```
Content-Type: application/json
Content-Length: 287
Server: Werkzeug/2.0.2 Python/3.8.10
Date: Mon, 24 Jan 2022 20:37:01 GMT

{
  "token":
"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJmcmVzaCI6ZmFsc2UsImlhdCI6MTY0MzA1NjYyM
SwianRpIjoiYmIyMGM4OWUtYmY1YS00MTcxLThkODgtODcwYmJjNmVkNWZmIiwidHlwZSI6ImFjY2Vz
cyIsInN1YiI6ImFkbWluIiwibmJmIjoxNjQzMDU2NjIxLCJleHAiOjE2NDMwNTc1MjF9.4EMhbtbdr4
froOoDTT8kHPVGiW0p-mPWfo8F-Xme780"
}
```

### 2.2.2 Ssfpr1 Interface Specification

The SFPR offers the ability to manage packaged SFs, i.e. Service Function Packages (SFPs), which are used at provisioning time to instantiate a Service Function as one or more Service Function Endpoints on Service Hosts. The table below provides an overview of available resources exposed by the SFPR on the Ssfpr1 interface towards the 5GC vendor.

*Table 1: Summary Table of the Ssfpr1 SFV Interface.*

| Resource | HTTP Method | Description |
|---|---|---|
| /util | GET | Obtain the storage utilisation of the Service Function Repository |
| /sfp | POST | Upload a SFP to the repository |
| /sfps | GET | Obtain the list of stored Service Function Packages |
| /sfp/<sfpid> | GET, DELETE | When using the GET method the properties for a Service Function Package are returned. The response comes as JSON-encoded with the fields size, mtime, sum and url.<br>When using the DELETE method the SFP is deleted from the SFPR. |
| /sfp/<sfpid> | DELETE | Delete a Service Function Package using its identifier |

#### 2.2.2.1 /util

This resource allows to check the total and used storage capacity of the SFPR. An example HTTP transaction is provided below using CURL:

```
~# curl sfpr:8080/util -H"Authorization: Bearer $TOKEN"
```

```
{
  "diskGB": 2.53888,
  "diskUsedGB": 0.58688
}
```

with $TOKEN being JWT obtain separately, as described in Section 2.2.1.1.

#### 2.2.2.2 /sfp

To upload a new Service Function Package, the /sfp resource can be called via an HTTP method POST. The SFPR responds with the SFP ID, e.g. db.lxc.tar.gz. An exemplary HTTP transaction using CURL is provided below:

```
~$ curl -i -X POST sfpr:8080/sfp -H"Authorization: Bearer $TOKEN" \
-F "file=@amf.lxc.tar.gz"

HTTP/1.1 100 Continue

HTTP/1.0 201 CREATED
Content-Type: text/html; charset=utf-8
Content-Length: 13
Server: Werkzeug/2.0.0 Python/3.8.5
Date: Thu, 06 Jan 2022 18:32:44 GMT

amf.lxc.tar.gz
```

with $TOKEN being the JWT obtained beforehand using the resource defined in Section 2.2.1.1.

#### 2.2.2.3 /sfps

This resource allows the 5GC vendor to obtain all previously uploaded SFPs. An example HTTP transaction using the HTTP request method GET looks as follows:

```
~# curl -i sfpr:8080/sfps -H"Authorization: Bearer $TOKEN"
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 36
Server: Werkzeug/2.0.2 Python/3.8.10
Date: Mon, 24 Jan 2022 21:08:04 GMT

["amf.lxc.tar.gz", "smf.lxc.tar.gz"]
```

with $TOKEN being the JWT obtained beforehand using the resource defined in Section 2.2.1.1.

#### 2.2.2.4 /sfp/<sfpid>

This resource allows the 5GC vendor to obtain the properties of a SFP identified through its SFPID such as the time when it was uploaded, its size in bytes, the SHA254 checksum and the URL under which it can be downloaded (to be used in the Resource Descriptor). An example HTTP transaction is provided below:

```
~# curl -i sfpr:8080/sfp/amf.lxc.tar.gz -H"Authorization: Bearer $TOKEN"
HTTP/1.0 200 OK
Content-Type: application/json
Content-Length: 174
Server: Werkzeug/2.0.2 Python/3.8.10
Date: Mon, 24 Jan 2022 21:18:53 GMT

{
  "mtime": 1643058393.6747239,
  "size": 0,
  "sum": "e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855",
  "url": "/sfp/download/admin/amf.lxc.tar.gz"
}
```

with $TOKEN being the JWT obtained beforehand using the resource defined in Section 2.2.1.1.

The deletion of a SFP uses the same resource but with the HTTP method DELETE:

```
~# curl -i sfpr:8080/sfp/amf.lxc.tar.gz -H"Authorization: Bearer $TOKEN" -X
DELETE
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 0
Server: Werkzeug/2.0.2 Python/3.8.10
Date: Mon, 24 Jan 2022 21:21:13 GMT
```

### 2.2.3 Sscc1 Interface Specification

The Sscc1 interface allows vendors to provision and manage Service Chains that form their 5GC. The table below provides an overview of available resources exposed by the SCC on the Sscc1 interface towards 5GC vendors.

| Resource | HTTP Method | Description |
|---|---|---|
| /chain | POST | Submit a Resource Descriptor to request the orchestration of a Service Chain. |
| /chains | GET | Obtain the list of provisioned Service Chains |
| /chain/<scid> | GET, DELETE | Obtain the status of a provisioned Service Chain or request to delete it. |
| /servicehosts | GET | Obtain a list of Service Hosts |

#### 2.2.3.1 /chain

This resource allows the submission of a Resource Descriptor, as further explained in Section 2.2.4. The SCC returns a Service Chain Identifier (SCID) which must be used for managing the SC. An example HTTP transaction using CURL is provided below:

```
~$ curl -i -X POST scc:8080/chain -H"Content-Type: text/x-yaml" -
H"Authorization: Bearer $TOKEN" --data-binary @rd.yml
HTTP/1.0 200 ACCEPTED
Content-Type: text/html; charset=utf-8
Content-Length: 2
Server: Werkzeug/2.0.2 Python/3.8.10
Date: Mon, 24 Jan 2022 21:40:58 GMT

my5gc
```

with $TOKEN being the JWT obtained through the resource described in Section 2.2.1.1.

The SCC implements an internal scheduler and the above request is placed into a queue which is processed in the background. Thus, the SCC confirms any new Resource Descriptor with a 202 HTTP response.

#### 2.2.3.2 /chains

This resource allows to request to list all available SCs for a specific user. The response is a JSON-encoded list of SCs. An example HTTP transaction is provided below:

```
~# curl -i -H"Authorization: Bearer $TOKEN" scc:8080/chains
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 2
Server: Werkzeug/2.0.2 Python/3.8.10
Date: Mon, 24 Jan 2022 21:44:09 GMT

["my5gc"]
```

with $TOKEN being the JWT obtained through the resource described in Section 2.2.1.1.

#### 2.2.3.3 /chain/<scid>

This resource allows to request the status of a previously orchestrated SC.

```
~# curl -i -H"Authorization: Bearer $TOKEN" scc:8080/chain/my5gc
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 2
Server: Werkzeug/2.0.2 Python/3.8.10
Date: Mon, 24 Jan 2022 21:44:09 GMT

["state": "provisioned"]
```

with $TOKEN being the JWT obtained through the resource described in Section 2.2.1.1.

Possible states this API returns are:

- requested: The orchestration request for this new Service Chain has been received but the SCC has not started its provisioning
- provisioning: The Service Chain is currently being provisioned to all Service Hosts.
- provisioned: The Service Chain has been successfully provisioned.

- eliminating: The Service Chain is currently being eliminated from all Service Hosts.
- eliminated: The Service Chain has been successfully eliminated.
- failed: The provisioning or elimination of the Service Chain has failed.

The state machine for the Service Chain states is illustrated below.

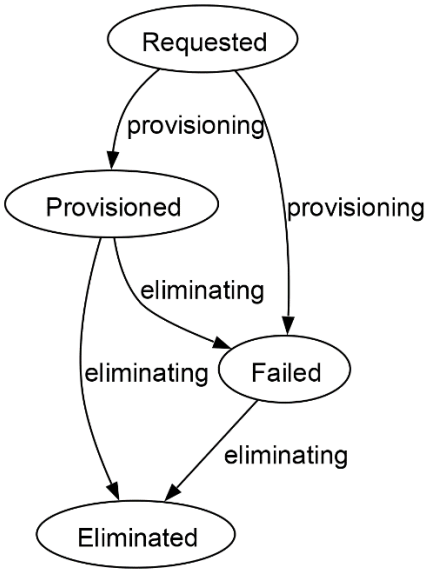


*Figure 3: Service Chain States and Transitions.*

#### 2.2.3.4 /servicehosts

This resource allows the 5GC vendor to obtain a list of all available Service Hosts using the HTTP method GET. An example HTTP transaction is provided below:

```
curl -i -H"Authorization: Bearer $TOKEN" scc:8080/servicehosts
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 2
Server: Werkzeug/2.0.2 Python/3.8.10
Date: Mon, 24 Jan 2022 21:23:58 GMT

[
  {
    "name": "dc2-sh",
    "identifier": "fa:16:3e:9c:40:ec ",
    "capabilities":
      {
        "compute": 37,
        "memory": 25.513652,
        "storage": 797.57454,
        "ims": ["kvm", "lxc", "docker"]
      },
                                                    "utilisation":
```

```
      {
        "compute": 0.0,
        "memory": 0.708744,
        "storage": 3.565464
      }
  }
]
```

with $TOKEN being the JWT obtained through the resource described in Section 2.2.3.1.

The units for compute, memory and storage are number of logical cores, GB and GB, respectively.

## 2.2.4 Swai Interface Specification

### 2.2.4.1 /whoami/<SFID>

The Swai interface of the SFEC offers SFEs the ability to query information about themselves. The response comes as an JSON-encoded payload and is mainly extracted from the resource descriptor submitted to the SCC initially.

```
curl -i -H"Authorization: Bearer $TOKEN" sfec:8080/whoami/fa:16:3e:b0:d2:91
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 2
Server: Werkzeug/2.0.2 Python/3.8.10
Date: Mon, 21 Nov 2022 20:12:04 GMT
[
  {
    “sc”: “fokus”,
    “sf”: “nrf”,
    “sfids”: “nrf.fudge”,
    “sh”: “fa:16:3e:90:37:7e”
  }
]
```

## 2.2.5 Resource Descriptor

As defined in D1.2 [F5G12], the SCC offers a RESTful API to provision enterprise services, e.g. 5G Cores, as a Service Chain (SC). Each SC must be described in regarding its required resources and provisioning states. The definition is composed of three main areas:

- **meta**: Meta information describing the Service Chain and descriptor.
- **service_functions**: A list of all Service Functions that form the Service Chain and their properties and required resources.
- **provisioning**: The declaration which Service Function is instantiated on which Service Host in which state. Also, the number of instances can be declared here that are available to be lifecycle managed through policies.

The actual resource descriptor definition is provided below and is provided in YAML.

```
meta:
    definition_version:
        required: true
        type: string
        description: The version of the descriptor definition allowing
versioning and backwards compatibility
    service_chain:
        required: true
        type: string
        description: The name of the chain following a reverse TLD convention,
e.g. com.foo.vr.premium-users

service_functions:
    service_function:
        required: true
        type: string
        description: The name of the Service Function
    identifiers:
        required: true
        type: array
        type_schema: string
        description: The list of identifiers (FQDNs) for this service function
    identifiers_configs:
        required: false
        type: array
        identifier:
            required: true
            type: string
            description: One of the FQDN listed under identifiers
        suppress_http_requests_get:
            required: false
            type: bool
            description: Suppresses the delivery of HTTP requests (GET) to
servers when performing co-incidental multicast
        suppress_http_responses_post:
            required: false
            type: bool
            description: Suppress the delivery of HTTP responses (OK) to
servers when performing multicast for POSTs
    service_function_package_url:
        required: true
        type: string
        description: The URL from where the SFP can be obtained
    instance_manager:
        required: true
        type: string
        description: The instance manager type this SF has been packaged for,
e.g. LXC, Docker or Android
    compute:
        required: true
```

```
        type: integer
    memory:
        required: true
        type: integer
        description: The demanded amount of memory for this SF in MB
    storage:
        required: true
        type: integer
        description: The demanded amount of storage for this SF in MB
    constraints:
        kernel_libraries:
            type: array
            type_schema: string

provisioning:
    service_function:
        required: true
        type: string
        description: The name of the Service Function, as declared under
service_functions > name
    service_host:
           required: true
           type: string
           description: Hostname of the Service Host on which the Service
Function should be provisioned.
    state:
        required: true
        type: string
        description: The state for the SFE(s)
        valid_values: [ "non-placed", "placed", "booted", "connected" ]
    instances:
        required: true
        description: The number of SFEs in this particular state on the Service
Host
```

## 2.3 Virtual Application Orchestration

Application Orchestration layer is empowered by the Vertical Application Orchestrator (VAO) builds atop well-established container and Infrastructure as a Service (IaaS) cloud platforms to provide automated service deployment, LCM, and scaling while dynamically exploiting the underlying provided services for optimising application performance. It also aims to help the vertical customers, by creating a development environment where services can be easily prototyped and quickly deployed into production. In that concept it allows verticals to compose applications following a conventional microservices-based approach where each component can be independently orchestrated.

Application developers who have already made the effort to disaggregate their applications into multiple independent components/microservices can readily register those components to the VAO UI by providing an application registry Uniform Resource Locator (URL) (e.g., docker registry). Apart from the components themselves,

vertical service providers/developers can also express dependencies between components, thus forming a DAG with application components as vertices and their dependencies as edges. In the case of monolithic (i.e., single-component) applications the application graph takes its simplest form, i.e., one vertex without edges. Once all application components are registered with the VAO, the UI visualises an application graph with all of its components and links.

Figure 4 visualizes the steps taken by the VAO and its internal components to perform UI-driven application composition and application policy definition. As it can be seen, first comes the registration of each application component through a UI panel. Then, the panel offers a wizard which allows to drag and drop application components and draw links between them. Once the graph is composed, the vertical application provider declares constraints and SLAs regarding the deployment of the graph. The process is thoroughly described in the following sections.

### 2.3.1 Registration and Composition of an Application

The actions for integrating and onboarding the vertical applications inside the VAO Platform is highlighted in the next diagram.

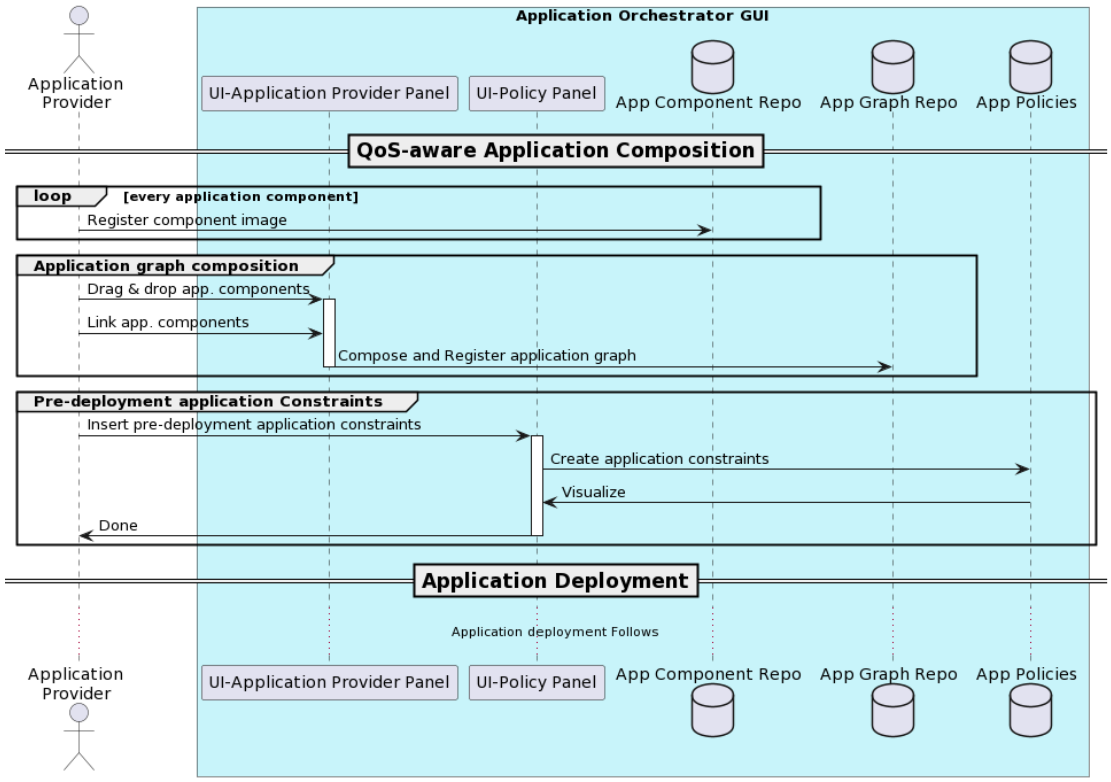


*Figure 4: Registration of an application diagram*

An application may comprise of one or more application components. Each app component is a standalone entity that executes part of the application logic. App components are organized as a direct acyclic graph. Dependencies/Connections between app components are edges. Application providers may choose which app components will be accessible by users.

**For starters** the technical integration with the VAO is to bring the different components to be deployed and communicate using dedicated interfaces or a common message broker in case this is needed.

VAO requires as input containers and operates on top of cloud technologies like Kubernetes and OpenStack. This way provides a layer of abstraction to the vertical customers to construct their service excluding technology-specific details.

A proper format for declaring all these information with regards to vertical applications are through the VAO-GUI which provides a stepwise service construction approach while it is also compatible with **Docker-Compose format.**

#### 2.3.1.1 Web Platform

On the web platform, role-based access control features are enabling specific users to use specific features and specific views of the web platform.

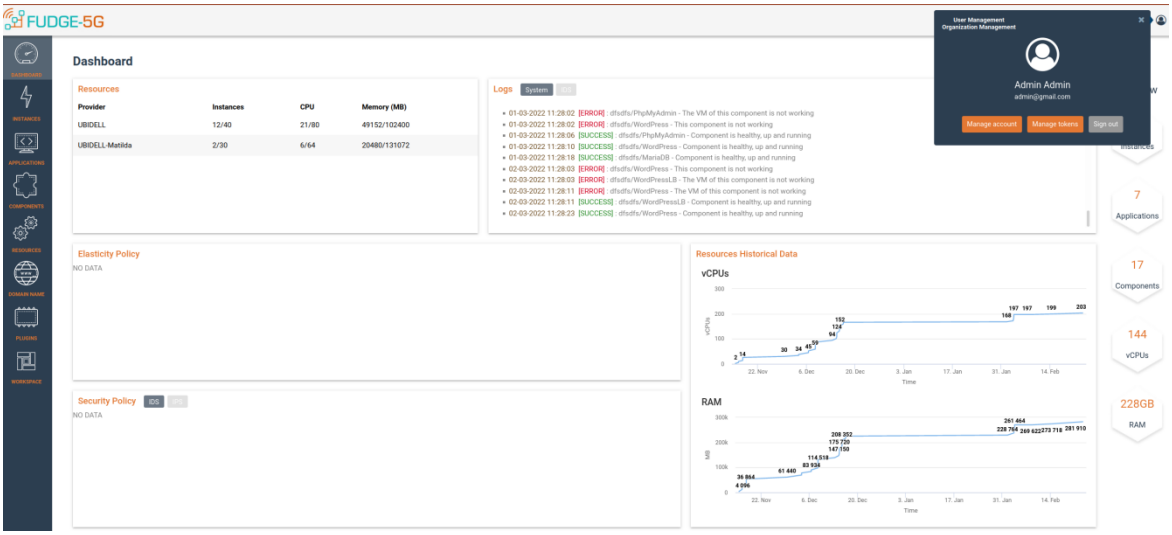


*Figure 5: Roles and Logging into the web platform*

#### 2.3.1.2 Registration of Individual Application Components

To onboard an application, the VAO offers a UI. For the creation of the application components, we need first to fill the following 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 that 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 from a MySQL and a PhpMyAdmin microservices, in our acyclic graph it 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 the order has been concluded, the onboarding of microservices as components to the platform can be started by filling the following forms:

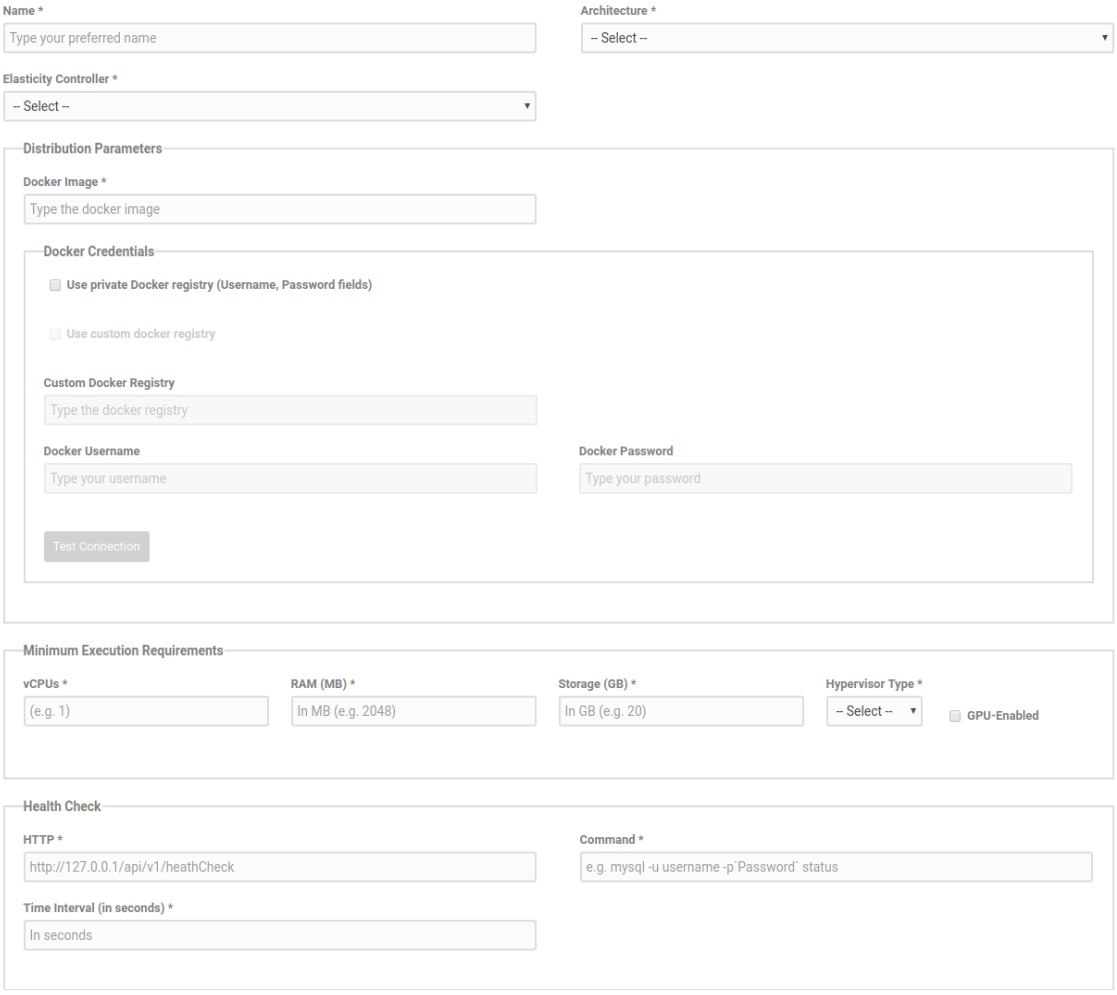


*Figure 6: Application's Component Details*

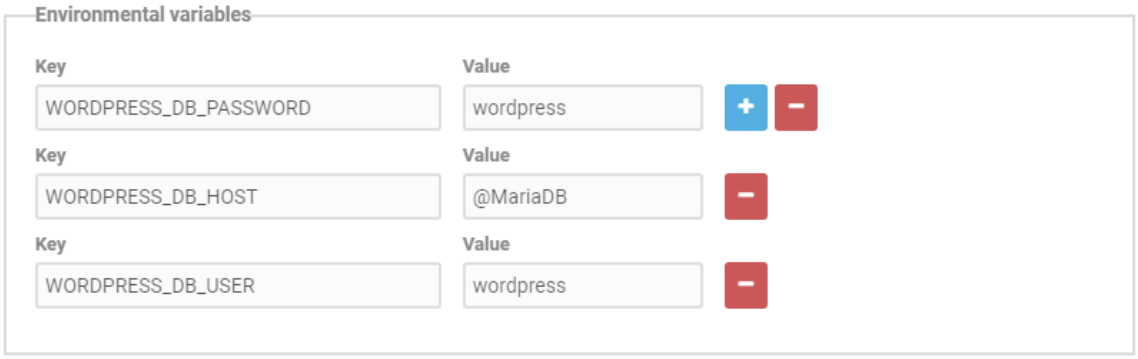


*Figure 7: Application Component Environment Variables Declaration*

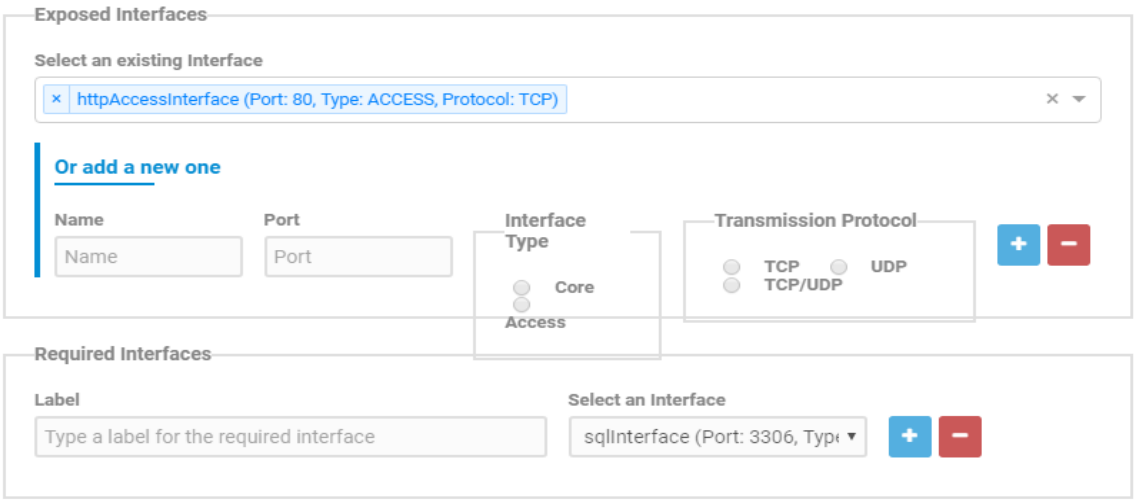


*Figure 8: Components Registration*

On the figure above we can find all the required basic configurations to create a component, like the docker image and registry of it, the minimum execution requirements that the container needs as also the health check exposed to find out if the microservice is running properly. The user can configure the environmental variables, the exposed ports as exposed interfaces, the dependencies on another microservices as required interfaces, the volumes, etc.

Specific reference should be made on the graph link constraints that Figure 8 shows. Such specifications targets to 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 8.

#### 2.3.1.3 Compose and Register Application Graph

After the application component registration ends, VAO provides a view to select and link all the necessary application components that constitutes the application graph. The image below demonstrates how the application graph can be created. The Application Provider can just search for any component needed to add to application graph and then just link the components by grabbing the spheres around the components and dropping it on the component you want to be linked.

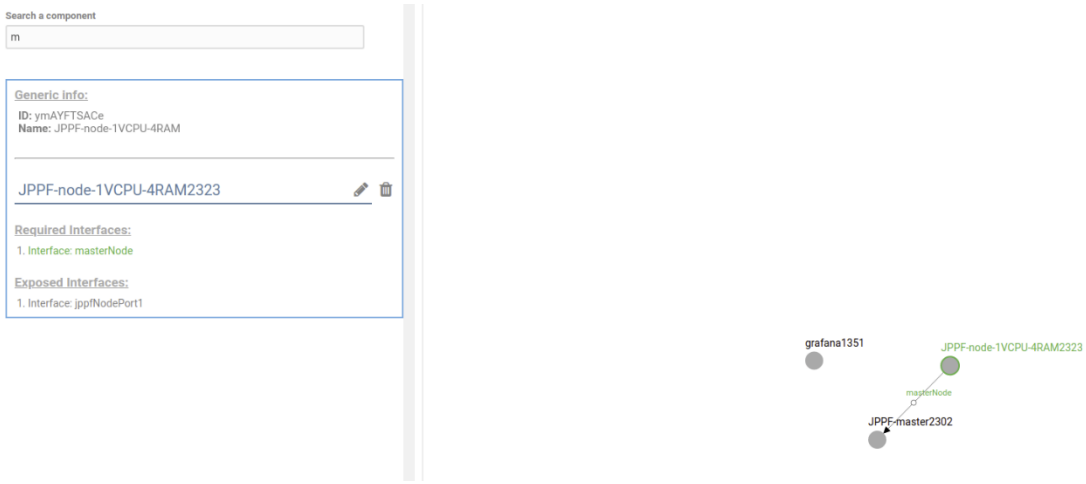


*Figure 9: Application graph creation*

#### Pre-deployment Application Constraints

After the composition of the application graph VAO 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 figures below illustrate these user interfaces for providing resource and high-level network constraints.


*Figure 10: Deployment Constraints (VIM related)*


*Figure 11: Deployment Constraints (resources)*

### 2.3.2 Application Deployment

Given that all the previous steps are successfully completed by the end user the actual deployment of the application will take place. The following picture depicts the successful deployment of an application on top of the programmable resources. Logging information is provided with details regarding the status of the application graph, along with some basic monitoring metrics per application component.

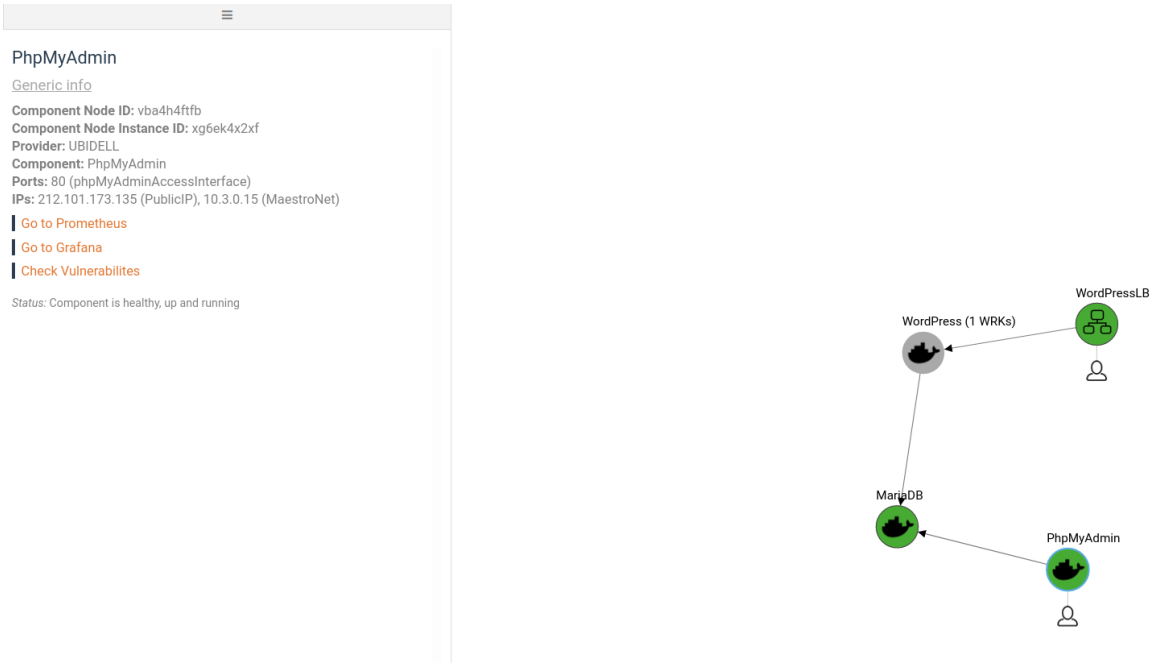


*Figure 12: Graphical Representation of a Deployed Vertical Application.*


*Figure 13: Monitoring/Debugging Information of a Deployed Vertical Application.*

### 2.3.3 Runtime Policies

The Policies Manager provides policies enforcement over the deployed application graphs following a continuous match-resolve-act approach. Specifically, the match phase regards the mapping of the set of applied rules that are satisfied based on alerts coming from the monitoring infrastructure. The resolve phase regards the process of conflict resolution for different rules that may be valid and triggered at the same time. Thus, the resolve phase aims at resolution among these rules taking into account the pre-defined priority of each rule. The act phase regards the provision of a set of suggested actions by the policy manager to the orchestration components, the Deployment Manager and the Execution Manager of the VAO, responsible for application graphs placement and management, respectively.

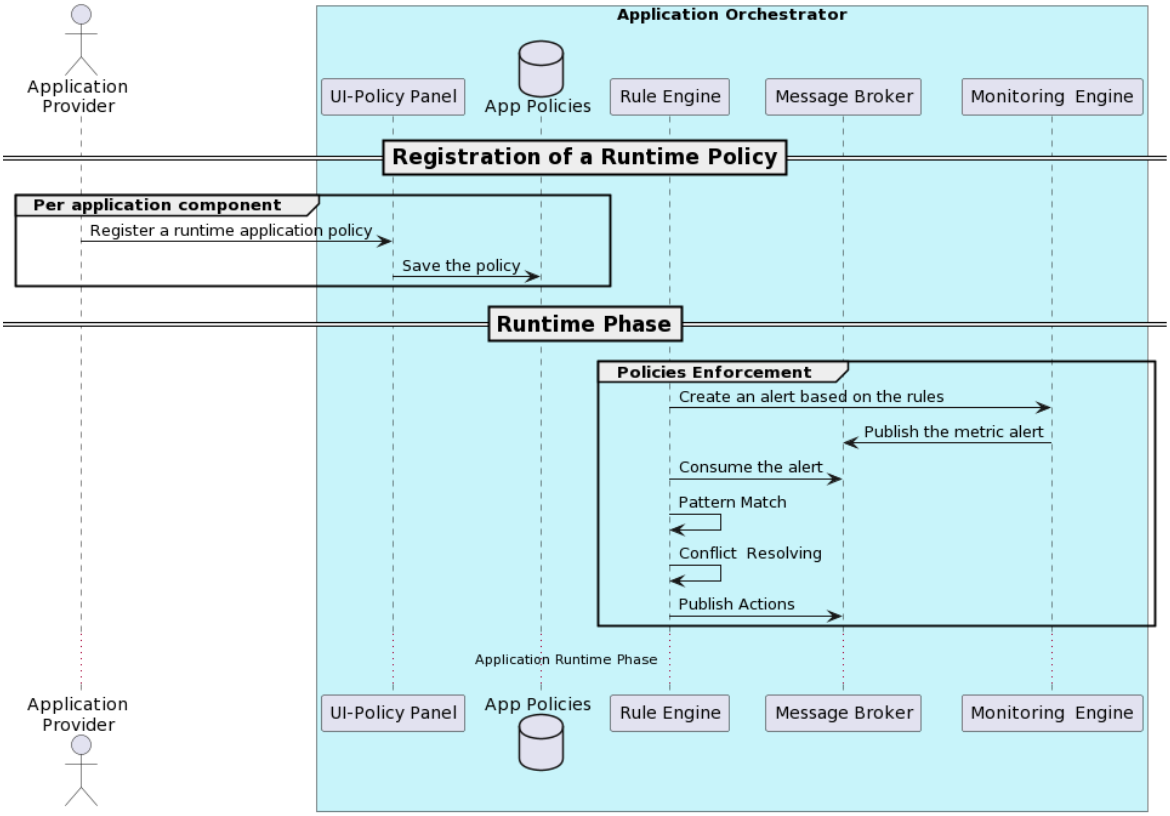


*Figure 14: Runtime Policies Activation*

Policies enforcement is realized through a rule-based framework that attempts to derive execution instructions based on the current set of data and the active rules; rules associated with the deployed application graphs at each point of time. Specifically, we have

adopted Drools rules-based management system [Drools], an open-source solution that supports the implementation of runtime policies enforcement mechanisms.

Policy Editor uses Drools under the hood. In order to overcome the cumbersome specificities of the system, a user-friendly way for the declaration of runtime policies was implemented. The policies regard elasticity and security management policies.

The Declaration of a policy is done through the UI (Figure 15) and a set of validation mechanisms on the backend, multiple rule-based expressions can be declared following a condition, event, action approach (upon specific conditions, identification of events that lead to actions). The user interface is developed with REACT.JS, while the validation mechanisms are based on the design and development of mechanisms (based on Java) for translating the declared rules to Drools.

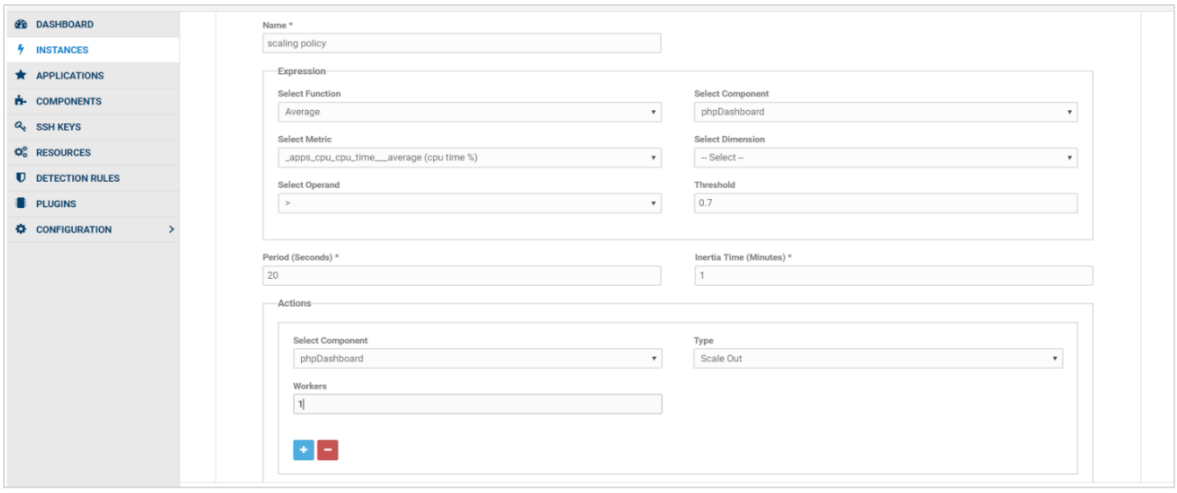


*Figure 15: User Interface for the Policies Definition.*

To Declare a Policy an application must be alive (i.e successfully deployed and running). The Monitoring metrics must be present inside Prometheus. After that through the GUI the composition of the policy targets a specific metric of a specific application component (or a combination of two or more metrics).

The actions that are for the moment supported are:

- scaling actions
    - scale in
    - scale out
- security actions
- block traffic
- Forensic

# 3 System Integration

This section describes the efforts in WP2 to integrate all platform components into the integration testbed and onboard Enterprise Services. While providing a detailed description of the infrastructure and platform layer, this section offers technical insights and challenges into the cloud-native orchestration of Enterprise Services into a platform which implements a Service-Based Architecture.

## 3.1 Testbed Infrastructure

The testbed is composed of a range of components that reside in the infrastructure layer of the FUDGE-5G system, as described in D1.3 [FUD13]. This includes the Radio Access Network (RAN), the compute hosts for the platform and the switching fabric which interconnects the RAN and compute infrastructure.

The RAN in the testbed is an Amarisoft Callbox Mini [AMA22] which offers 2x2 MIMO. It can operate in the FR1 and FR2 spectrum, and allows to run in 5G SA mode with the 5G Core fully externalised. During the course of the FUDGE-5G project, the base station received software updates to support 3GPP's Rel. 16 specification. The spectrum at the testbed location (i.e. London) was allocated by the UK authority Ofcom and is a shared licence for the C-Band n79 at 40MHz channel bandwidth.

The modems available for testing were:

- Waveshare [WSH22] operating via GPIOs over a Raspberry Pi 4 and is based on Qualcomm's Snapdragon X55. The modem comes with external antenna ports allowing better antenna positioning for MIMO. This modem implements Rel. 15
- Quectel development kit [QUE22] operating as an USB device and comes with integrated antennas. It also uses Qualcomm's Snapdragon X55 but comes with their own modem manager QMI. This modem implements Rel. 15.
- Fivecomm's modem [FIV22] also operating via GPIOs of a Raspberry Pi using Quectel's module. This modem implements Rel. 15.

The compute infrastructure is based on a range of HP rack and Dell Precise Tower desktop machines with different CPU, RAM and storage configurations. These compute hosts are managed by an OpenStack NFV framework offering the orchestration of platform components as Virtual Machines (VMs) in an automated manner. Also, towards the edge of the infrastructure, Intel NUCs were used to reduce the form factor, without significantly compromising on performance or compute capabilities.

The switching fabric is based on Pica8 [PIC22] hardware SDN switches offering 48 1G ethernet and four 10G SFP ports. Each switch supports standard L2/L3 packet forwarding

capabilities (i.e. managed switch), OpenFlow-based operations and Pica8's owned CrossFlow support (operating switch in both modes).

Figure 16 depicts the infrastructure with OpenStack-managed compute hosts starting with "os-*" and the Intel NUCs at the bottom of the figure as "nuc*". The gNB is illustrated at the bottom left as "amarisoft". As illustrated, all compute hosts are interconnected via three SDN switches "Pica8-*" forming a 10G triangle core connection and 1G branches.

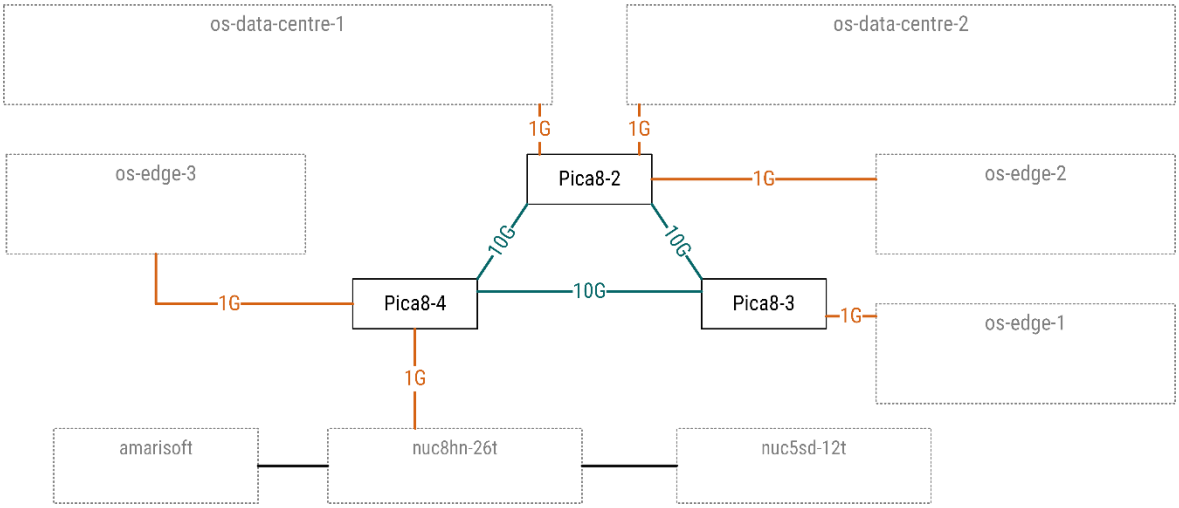


*Figure 16: Infrastructure of FUDGE-5G's Integration Testbed*

The properties of all compute hosts are listed in the table below.

*Table 2: Properties of Infrastructure Compute Hosts.*

| Compute Host | vCPUs | Memory [GB] | Storage [GB] |
|---|---|---|---|
| os-data-centre-1 | 40 | 32 | 750 |
| os-data-centre-2 | 40 | 32 | 750 |
| os-edge-1 | 16 | 64 | 250 |
| os-edge-2 | 16 | 64 | 250 |
| os-edge-3 | 16 | 64 | 250 |
| nuc8hn-26t | 8 | 24 | 220 |
| nuv5sd-12t | 8 | 24 | 220 |

## 3.2 Testbed Platform

As described in [FUD13], the platform is composed of the functionalities routing, orchestration and telemetry. The respective technologies implementing these

functionalities are deployed as VMs and Linux Containers (LXCs) via OpenStack and by hand, respectively. The deployment and logical interconnection among VM/LXC instances are illustrated in Figure 17. Colour coding is leveraged to illustrate which instance implements which of the three platform functionalities.

Note, the orchestration of Enterprise Services is split between the Service Function Virtualisation Orchestrator (SFVO) and Vertical Application Orchestrator (VAO). The VAO is colour coded in pine green and located in the os-data-centre-1 compute host. It manages the DN instance at the boom right, where vertical applications are orchestrated and reachable over the 5G user plane.

Meanwhile, the SFVO offers orchestration of 5GC NFs into SHs, colour coded as cyan. The SFVO components SCC, SFEC and SFPR are located in the os-data-centre-1 compute host with several SHs spread across the infrastructure, labelled as "*-sh".

The Service Communication Proxy (SCP) resides with an "*-sp" instance at each compute host where an SH has been instantiated. At the bottom of the figure, the gNB has dedicated links for N2 and N3 traffic into the SCP (fe1-sp) and SH (fe1-sh), respectively. Lastly, the "sia-vpn" instance on the top of Figure 17 offers VPN access to all platform APIs for Enterprise Service vendors to orchestrate and monitor their applications. The offered APIs are the ones described in Section 2 of this document.

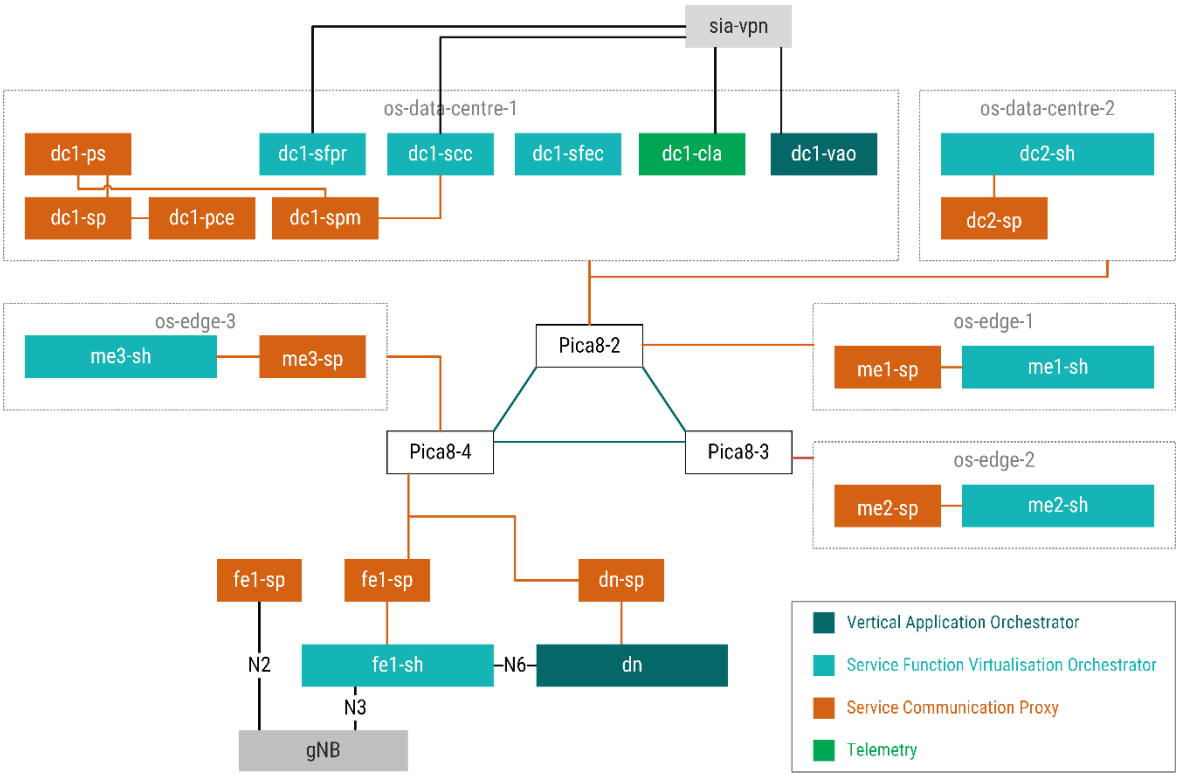


*Figure 17: FUDGE-5G Platform of Integration Testbed*

## 3.3 5G Core Integration Challenges and Experiences

FUDGE-5G aimed to put various 5G technologies under trial in a private network setting, aiming at demonstrating 5G's customisation potential. One of the key propositions of 5G is the ability of the Core Network to be much more flexible regarding cloud-native provisioning, multi-vendor deployment and scalability. This sub-section focuses on the experiences of 5G Core Networks orchestration via the SFVO in the integration testbed.

The platform's SFV Orchestrator offers a simplified descriptor definition compared to TOSCA or HOTs. The SFVO departs from traditional NFV descriptor information where networking declarations is an integral part of a cloud-centric definition changing the focus point of connectivity to the FQDN of an instance only. This has the benefit to externalise the networking (aka service routing) entirely from the service and the infrastructure. Furthermore, location-aware and cloud-native procedures to provision and life-cycle manage software packaged as Linux or Docker containers or Kernel Virtual Machines (KVM) are introduced in the SFVO descriptor. As described in further detail in Section 2.2.5, the SFVO's resource descriptor defines which NF should be deployed into which location with information attached about compute, memory and storage requirements only. The challenges of orchestrating 5GCs in a cloud-native fashion are described hereafter.

FUDGE-5G's SBA platform comes with a Service Communication Proxy which offers service routing capabilities for HTTP-based traffic. However, in order to utilise this feature Network Functions must implement the **usage** of **Fully Qualified Domain Names (FQDNs)** and therefore the **support** for **Domain Name Service (DNS)**. Furthermore, service routing can be performed only when more than one instance of the same Network Function is deployed so that the SCP can choose the most appropriate one to serve a HTTP request (e.g. based on shortest path routing). But while the latter one would require the implementation of a Network Function using modern 12-factor app methodologies, none of the implementations available to the project implemented such Network Function. Furthermore, the usage of FQDNs was also a rather scarcely implemented functionality with many NFs utilising IP addresses directly as the Host identifier. D2.4 [FUD24] addressed this by studying the design requirements on how to develop cloud-native NFs.

In FUDGE-5G, the proposition of the SFVO is that all **Network Functions** are fully **pre-packaged** and can be freely deployed **without** any **further configuration**, fully decoupling the internals of a Network Function from the cloud-native orchestration procedures. However, when using Docker, it is often considered to build and deploy the software that implements a Network Function at the time of creating the container. Thus, post-provisioning configurations are realised using cloud-init or Docker Compose, which is not implemented in the SFVO. Also, outside of FUDGE-5G, all 5GC vendors have their own choice of deployment framework in the likes of Ansible or Chef in addition to integrated Gitlab pipelines for conformance testing before releasing software. The integration of the

SFVO into the various frameworks was not pursued aiming to decouple these processes to minimise the development effort required on top of implementing features within Network Functions.

Once all **NFs bootstrap**, some **implementations** have **dependencies on other NFs** in order to instantiate correctly. Often, this refers to databases or even the NRF as the first point of registration of an NF. The SFVO however does not have the ability to understand what is running inside a Service Function, as its focus point is on instantiating the SF via the correct instance manager (Docker. LXD, KVM). Also, even though an instance is started successfully, the SFVO has no knowledge about what is running inside the SF and how long it takes for the software inside the SF to bootstrap. Thus, the SFVO offers a script as part of the packaging environment which allows to provide a set of FQDN, transport protocol and port tuples. The script is executed after networking in the SFE becomes available and tries to establish a TCP or UDP connection to the FQDN and port provided as an argument. Only until all endpoints are reachable the script exits, allowing SF software to bootstrap and have one SFE available to complete the bootstrapping successfully.

Further to the dependency on other SFs being operational already, some NFs are implemented to **only listen** on the **IP address their local interface** have for security purposes and/or to register with this IP directly with the NRF (bypassing the use of FQDNs entirely). This information is often pre-configured through an external deployment chain, e.g. ansible or puppet. As IP addresses are assigned through DHCP and each deployment in a different infrastructure might have different IP ranges, the SFVO does not offer the ability to statically assign an IP address prior to the orchestration of a NF. Again, mainly driven by the assumption to be network agnostic, as the orchestration should be fully automated and agnostic to the underlying infrastructure – aka cloud-native.

Similar to the above, **some FQDN-based NF implementations require their own FQDN to be resolvable at bootstrapping time** to bind to the correct IP address. This logic causes two rather significant challenges:

- When provisioning the NF, the SFVO first creates the instance, boots it and then connects it to the routing layer. As BOOT and CONNECT are distinct operations in the SFVO and SCP, there is a likelihood that the CONNECT state is reached after the bootstrapping started. The ability to separate BOOT and CONNECT is driven by the design ambition allowing NFs to bootstrap entirely and be ready to be connected without any delay to scale an NF service as fast as possible.
- Related to the above, if the state of an NF is set to BOOT only at provisioning time, the NF would fail to bootstrap properly. If the decision to set the NF instance to CONNECT, the SCP assumes the NF service is reachable. If this NF instance is the first one to come up when being set to BOOT, there is other NF instance already registered under this FQDN, allowing the booted NF to resolve its FQDN (note, all NF instances

would have the same FQDN, allowing the SCP to pick the most suitable instance to serve a HTTP request).

The above challenges and experiences have been mainly around the design choices of how NFs are implemented regarding how flexible they are to be deployed anywhere at any time without any external dependencies. One of the most challenging NF to be integrated into the cloud-native workflow of the SFVO are UPFs. The reason for that is the nature of UPFs to switch/route packets on the User Plane in comparison to other NFs that are implementing pure application layer logic accessible via RESTful APIs. During the project, it has become apparent that UPFs cannot be agnostic to the underlying network and might even require dedicated networks exposed directly to the software implementing the UPF logic. Thus, the SFVO has received updates in its resource descriptor allowing to specify which network types must be present for SFs that represent a UPF (see Section 2.2.5 for more details). The SFVO then is aware which network available on a Service Host is offering N3, N4, N6 or N9 functionality.

On top of this network abstraction, it has been still rather challenging to move away from NFV principles, as UPFs are implemented using severely different routing approaches and therefore it became a requirement for the SFVO to still expose networks to Service Functions (aka Network Functions).

The example given in Figure 18 illustrates a UPF that operates N3 and N6 in two different IP networks in order to have full control over the flows between UEs and DNs via the same PLMN. Each UE is configured with a different APN, i.e. hospital, i-npn and industry. Each DN is operating in a dedicated 192.168.1x.0/24 IP subnet where UEs and UPF establish PDU sessions over the GTP-U tunnel on 192.168.82.0/24 (N3/orange colour code in the figure). Each DN then is mapped to a UE subnet matching in the Class C value of the IP address, i.e. 10 for hospital, 11 for i-npn and 12 for industry.

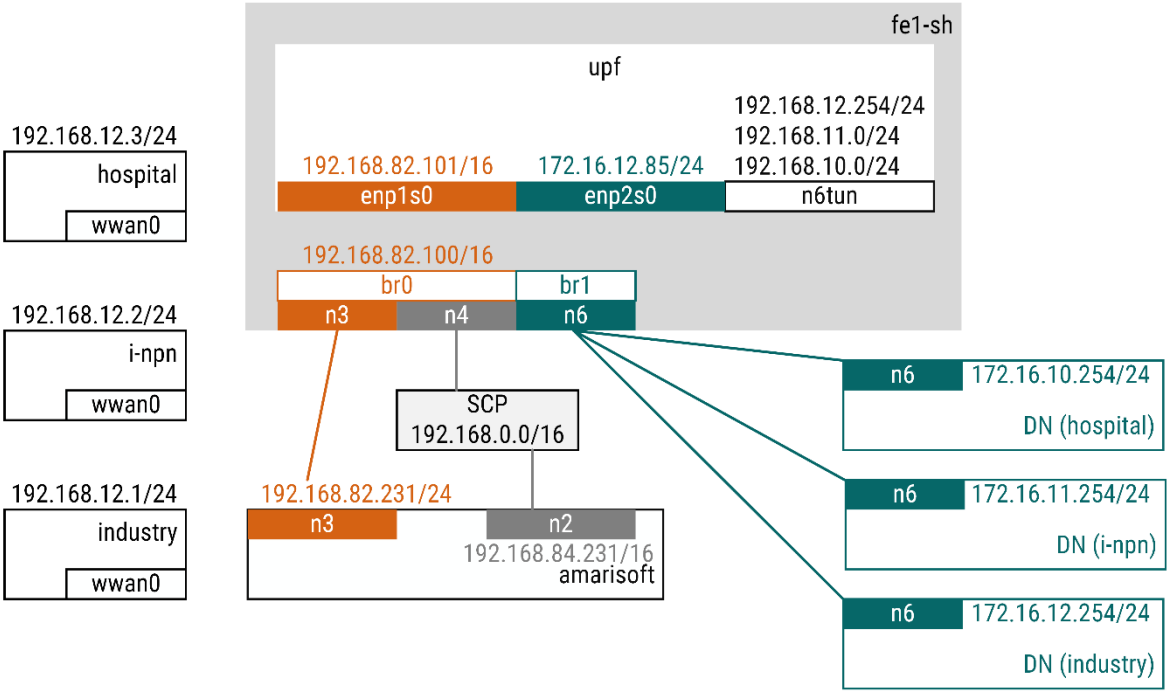


*Figure 18: IP Routing Configuration of User Plane*

The challenges with the SFVO orchestration procedures is that the platform (SCP) and infrastructure (gNB/DN) must be fully aligned with the IP subnet configuration inside the UPF. While platform and infrastructure are following NFV principles where infrastructure information around subnets and network names are communicated outside of an NFVO (such as OpenStack), the SFVO did not offer any form of network awareness to SFs such as the UPF. As a result, the SFVO received the ability to define what network types are available on a Service Host, e.g. N3, N4 and N6. This allows the SFVO to expose the networks available on a Service Host to Enterprise Service vendors to agnostically write the resource descriptor correctly (which SF goes onto which Service Host). Furthermore, it allows the SFVO to guarantee the requested networks are available.

However, one challenge could not be addressed in a cloud-native fashion, i.e. the network-agnostic provisioning of UPFs. The UPF in the example above assumes N3 and N4 on the same interface and in the same subnet. For that reason, the Linux bridge br0 on the Service Host fe1-sh combines the N3 and N4 interface and leaves it to amarisoft and scp node to drop or accept packets in the correct subnet they serve.

What could be addressed successfully in the SFVO though was the requirement to have specific kernel modules and libraries installed, loaded and exposed to Linux and Docker containers. The resource descriptor allows to specify the exact Linux package name and version that must be installed (see Section 2.2.5).

In summary, all non-UPF challenges with 5GCs available to the project were merely software design patterns related and rather straight forward addressed by the developers of the 5GCs and SFVO. However, the abstraction of platform networks on the Service Host towards UPFs could not be fully automated if UPFs require specific number of interfaces for N3, N4, N6 and N9.

## 3.4 5G Multicast Broadcast

5G Multicast Broadcast or 5MBS has finished standardization in 3GPP, marked by the closure of the associated Work item in June 2022 and validated during the SA#96 plenary. However, even if the technology is standardized, it is up for manufacturer implementation to support this solution in commercial equipment. To bridge the gap, UPV has developed a 5MBS software prototype which provides E2E multicast capabilities over FOKUS Open5GCore, as it has been reported in FUDGE-5G D2.3 [FUD23].

The design followed in the 5MBS prototype is an extension of the existing Network Functions provided by FOKUS Open5GCore. In detail, the MB-UPF has been implemented by adding new functionality to the UPF, essentially creating a collocated MB-UPF/UPF with both multicast and unicast routing capabilities. In the same vein, the MB-SMF is an extension of the SMF and handles both multicast and unicast sessions. To validate the multicast capabilities of the prototype, the Benchmarking tool included in Open5GCore is used, where the software provides virtual gNBs and UEs, represented as Linux terminals, which emulate real users and can send commands to the internet. In this regard, the 5MBS prototype has experienced a breakthrough in the implementations thanks to a close collaboration between FOKUS and UPV. The earlier versions, deployed over Open5GCore Release 6, relied on an IGMP endpoint between the MB-UPF and the gNBs, as shown in Figure 19. The role of the endpoint is to modify the upcoming downlink multicast flow from the N3mb interface into a unicast so the virtualized gNBs can forward data accordingly to the emulated UEs. This introduces a limitation where only one UE could be connected to a gNB as the gNB was not able to discern which UE to forward the multicast-converted packets.

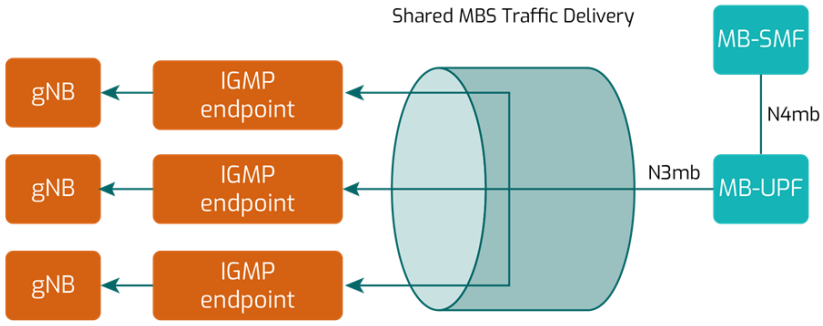


*Figure 19: Open5GCore Release 6 5MBS implementation by UPV. An IGMP endpoint is deployed for every virtual gNB instanced for the experiment, which will translate the multicast IP from a multicast flow into a unicast one.*

The prototype was migrated to Open5GCore Release 7. The most relevant breakthrough in this regard is the update to the Benchmarking Tool where the virtualized gNB now natively include the possibility of accepting and forwarding multicast flows to the emulated UEs under it.

The current deployed version in UPV premises uses this version, which is deployed following the architecture below:

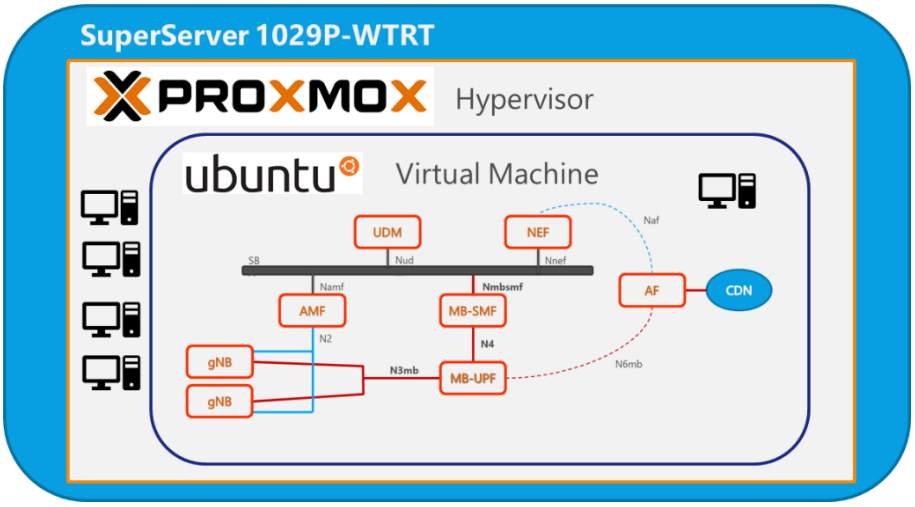


*Figure 20: Deployment architecture of the UPV node, showcasing the 5MBS prototype.*

An initial validation test has been carried out to evaluate a possible performance degradation of the multicast flows inside the modified Open5GCore against a unicast transmission to the UEs. To do so, a batch of experiments using the iperf bandwidth test tool between a Network Host and a virtualized UEs has been done. The setup process is like the one presented in D2.3, which the main difference being that there is no need to launch the IGMP endpoint. The Network Host launches the iperf server and delivers data over N6mb to the MB-UPF, which will deliver the data to the gNB. The gNB will forward this data to the UE. This

test was done for a range of bandwidths, both using unicast and multicast delivery. The parameters of the test and the VM running the prototype are described in Table 3.

*Table 3: Parameters used for the 5MBS prototype validation.*

| Parameter | Value | Description |
|---|---|---|
| Nº of Processors | 4 | Number of processors allocated to the VM |
| Processor Speed | 2.1 GHz | Speed of the processors allocated to the VM |
| RAM | 4 Gb | Amount of memory allocated to the VM |
| Disk | SSD | Type of storage used in the VM |
| Tested Bandwidth | 1,2,5,10,15,20,35,50 Mbit/s | Bandwidth parameter specified in iperf for the test |
| Type of Traffic | UDP | UDP is mandatory in iperf for multicast traffic |

The executed commands are detailed next. The parameter in red is what is variable from test to test. The unicast commands are as follows:

```
On the UE: iperf -s -u -i 1
On the Network Host: iperf -c 192.168.6.2 -u -T 32 -t 300 -i 1 -l 100 -b 20M
```

In the case of multicast, the commands are:

```
On the UE: iperf -s -u -B 239.0.0.6 -i 1
On the AF/IGW: iperf -c 239.0.0.6 -u -T 32 -t 300 -i 1 -l 100 -b 20M
```

The parameters evaluated were % of lost packets and datagram reordering. Note that datagram reordering is not inherently a negative KPI. The UDP datagrams could be reordered depending on the amount of threading and core affinity of the different parts of the Open5GCore. The results of the test are shown below:

*Table 4:Results obtained from the iperf testing over unicast and multicast scenarios.*

| Type of Traffic | Bandwidth | Datagram Disordering | % of lost datagrams |
|---|---|---|---|
| Unicast | 1 Mbit/s | No | 0% |
| Unicast | 2 Mbit/s | No | 0% |
| Unicast | 5 Mbit/s | Yes | 0% |
| Unicast | 10 Mbit/s | Yes | 0% |
| Unicast | 15 Mbit/s | Yes | 0% |
| Unicast | 20 Mbit/s | Yes | 0% |
| Unicast | 35 Mbit/s | Yes | 0.88% |
| Unicast | 50 Mbit/s | Yes | 14% |
| Multicast | 1 Mbit/s | No | 0% |
| Multicast | 2 Mbit/s | No | 0% |
| Multicast | 5 Mbit/s | No | 0% |
| Multicast | 10 Mbit/s | Yes | 0% |
| Multicast | 15 Mbit/s | Yes | 0% |
| Multicast | 20 Mbit/s | No | 0% |
| Multicast | 35 Mbit/s | No | 0.75% |
| Multicast | 50 Mbit/s | Yes | 22% |

From the results it can be obtained that the 5MBS prototype and the Open5GCore in UPV premises can send without errors up to 35 Mbit/s, where some datagrams will start to get lost. The number of lost packets in this bandwidth range is similar both for unicast and multicast. However, as the data rate raises, this number increases for both transmission modes, reaching 22% in multicast and 14% in unicast. It can be derived that the implementation of the 5MBS technology introduces a performance degradation of the overall capabilities of Open5GCore when a bandwidth of 50 Mbit/s or more is used.

# 4 Conclusions

This last section marks the end of D2.5 and WP2 in FUDGE-5G. D2.5 provided the list of platform APIs allowing Enterprise Service providers to orchestrate their software in a cloud-native fashion. D2.5 also provided a comprehensive description of the integration testbed used to incorporate all platform components. Along with the compute capabilities of the testbed, D2.5 discussed the challenges and experiences of orchestrating various 5G Cores available to the project.

The technologies developed and integrated in WP2 were handed over to WP3 for integration across all five use cases.

# References


[AMA22] Amarisoft SAS, "Callbox Series", Online: https://www.amarisoft.com/products/test-measurements/amari-lte-callbox/

[FUD11] FUDGE-5G Consortium, "D1.1: Technical Blueprint for Vertical Use Cases and Validation Framework", 2021. Online: https://www.fudge-5g.eu/download-file/493/zIrbJ2b9meNEGKKXwkEV

[FUD13] FUDGE-5G Consortium, "D1.3: FUDGE-5G Platform Architecture Final Release", 2022. Online: https://www.fudge-5g.eu/download-file/543/iYIVtTS3SKWmXm6QfdQp

[FIV22] Fivecomm, "5G BROAD", Online: https://fivecomm.eu/fivecomm-5g-devices/

[FUD23] FUDGE-5G Consortium, "D2.3: Converged 5GLAN with TSN and Multicast Capabilities", Online: https://www.fudge-5g.eu/download-file/546/1bAeN0CztfGCoRpgNGtS

[FUD24] FUDGE-5G Consortium, "D2.4: Disintegrated Network Functions for Cloud-native Service Orchestration", Online: https://www.fudge-5g.eu/download-file/547/bujMxEwFz4V55VwvJG2H

[GLT25] FUDGE-5G Gitlab, "T2.5 – Testbed Integration", Online: https://gitlab.fudge-5g.eu/wp2/t2.5-testbed-integration

[OFC22] Ofcom, "", Online: https://www.ofcom.org.uk

[PIC22] Pica8, "P3297", Online: https://www.pica8.com/white-box-switch/

[QUE22] Quectel, "RMU500EK 5G Development-Kit", Online: https://www.tekmodul.de/produkt/5g-modul-quectel-rm500q-gl/

[WSH22] Waveshare, "SIM8200EA-M2 5G HAT", Online: https://www.waveshare.com/wiki/SIM8200EA-M2_5G_HAT

[YTSFV] Sebastian Robitzsch, "Service Function Virtualisation – Demonstration of InterDigital's SBA Orchestration Capabilities for 5G Core Vendors", 2022. Online: https://youtu.be/2ybJp8Ff3uM