FUDGE-5G

FUlly DisinteGrated private nEtworks
for 5G verticals

Deliverable D2.4

FUDGE-5G Disintegrated Network Functions for Cloud-native Orchestration

Version 1.0

Work Package 2

Main authors	Marco Centenaro and Nicola di Pietro (ATH)
Distribution	PU
Delivery date	July 2022
Delivered date	

Partners

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

Cloud-native orchestration of network services, together with a microservice-based approach to developing network functions, are central features of the service deployment framework and architectural vision proposed by FUDGE-5G. This deliverable reports on the activities on these subjects carried out within tasks T2.3 (Cloud-native Service Orchestration) and T2.4 (Disintegration of Network Functions as Microservices) of the project's second work package. This work contributes to bridging the worlds of information technology and mobile communication networks, adapting to the latter some frameworks for advanced service and functionality development that come from the former (e.g., cloud-native functions and microservices). In this document, we delineate FUDGE-5G's design and development approach to a cloud-native orchestration of Enterprise Services, describing the features and requirements that characterize cloud-native "orchestrable" software components. Then, within the framework of cloud-native services and functions, we move to analysing how 5G core network functions can be designed and developed as microservices. This requires several steps beyond the state of the art because a mapping between the sub-functionalities of 5G core network functions and their possible restructuring into a microservices-based architecture is not given by the standard. This work re-elaborates certain 5G architectural elements and introduces paradigms that fit within the enhanced service-based architecture proposed by FUDGE-5G.

Versioning and Contribution History

#	Description	Contributors
0.1	Initial table of contents	ATH
0.2	Main contributions to Section 1 and 3	ATH, IDE, ONE, O2M
0.3	Main contributions to Section 2 and further contributions to Section 3	O2M, UBI
0.4	Further contributions to Section 2 and 3	O2M, CMC, IDE, UBI, ONE
0.5	Final contributions to all sections, editing, and completion of the draft sent to external reviews	ALL
0.6	This version includes one external reviewer's comments	
0.7	Improvements after review of Section 2 and 3	O2M, UBI, ATH
0.8	Further improvements of Section 2 and inclusion of the second reviewer's comments	ATH, UBI
0.9	Final contributions and modifications	ATH, ONE, IDE
1.0	Submitted version	ATH

Contributors

Partner	Authors
CMC	Mika Skarp, Jose Costa-Requena
IDE	Sebastian Robitzsch, Chathura Sarathchandra, Morteza Kheirkhah
O2M	Peter Sanders
ATH	Marco Centenaro, Nicola di Pietro, Daniele Munaretto, Arif Ishaq
ONE	Luís Cordeiro, André Gomes, João Fernandes
UPV	Borja Iñesta, Josep Ribes, Carlos Barjau, David Gómez-Barquero
FHG	Pousali Chakraborty, Marius-Iulian Corici
UBI	Thanos Xirofotos, Dimitrios Klonidis
HWDU	Zoran Despotovic

Reviewers

Reviewers	Affiliation
Prof. Fabrizio Granelli	University of Trento, Trento, Italy
Prof. Rui Aguiar	Instituto de Telecomunicações, Aveiro, Portugal

Abbreviations

2G	2nd Generation of mobile networks
3G	3rd Generation of mobile networks
3GPP	3rd Generation Partnership Project
4G	4th Generation of mobile networks
5G-AKA	5G Authentication and Key Agreement
5G	5th Generation of mobile networks
5GC	5G Core network
5QI	5G QoS Identifier
6G	6th Generation of mobile networks
AF	Application Function
AKA	Authentication and Key Agreement
AMBR	Aggregate Maximum Bit Rate
AMF	Access and mobility Management Function
ATH	Athonet (partner of FUDGE-5G)
API	Application Programming Interface
ARP	Allocation and Retention Priority
BDTP	Background Data Transfer Policy
BSF	Binding Support Function
BSS	Business Support System
CAP	Common Alerting Protocol
CB	Cell Broadcast
CBCF	Cell Broadcast Control Function
CBE	Cell Broadcast Entity
CHF	Charging Function
CMC	Cumucore (partner of FUDGE-5G)
CNCF	Cloud Native Computing Foundation
CRUD	Create, Read, Update, Delete
DMZ	De-Militarized Zone
DN	Data Network
DNS	Domain Name System
EAP-AKA'	Extensible Authentication Protocol AKA'
ETSI	European Telecommunications Standards Institute
FHG	Fraunhofer FOKUS (partner of FUDGE-5G)

FQDN	Fully Qualified Domain Name
GBR	Guaranteed Bit Rate
gNB	gNodeB (5G base station)
GPU	Graphics Processing Unit
HTTP	Hypertext Transfer Protocol
HWDU	Huawei
IDE	InterDigital Europe (partner of FUDGE-5G)
IP	Internet Protocol
IP-CAN	IP-Connectivity Access Network
IT	Information Technology
JSON	JavaScript Object Notation
MANO	Management And Orchestration
MBR	Maximum Bit Rate
MOI	Managed Object Instance
NBIFOM	Network-Based IP Flow Mobility
NBR	Name-Based Routing
NEF	Network Exposure Function
NF	Network Function
NFMF	Network Function Management Function
NFVI	Network Function Virtualization Infrastructure
NGMN	Next Generation Mobile Network
NIDD	Non-IP Data Delivery
NMS	Network Management System
NPN	Non-Public Network
NRF	Network Repository Function
NS	Network Service
NSD	Network Service Descriptor
NSS	Network Slice Subnet
NSSMF	Network Slice Subnet Management Function
O2M	One2many (partner of FUDGE-5G)
OAI	OpenAirInterface
OAM	Operation, Administration, and Maintenance
ONE	OneSource (partner of FUDGE-5G)
OS	Operative System
OSM	Open-Source MANO
OSS	Operations Support System
PCC	Policy and Charging Control
PCE	Path Computation Element
PCF	Policy Control Function
PCRF	Policy and Charging Rules Function
PDN	Packet Data Network
PDU	Protocol Data Unit
PFCP	Packet Forwarding Control Protocol
PGW	PDN Gateway

PLMN	Public Land Mobile Network
PPDR	Public Protection and Disaster Relief
PRA	Presence Reporting Area
QCI	QoS Class Identifier
QoS	Quality of Service
RAN	Radio Access Network
R&I	Research and Innovation
RPC	Remote Procedure Call
RTT	Round-Trip Time
SA	Stand-alone
SBA	Service-Based Architecture
SBI	Service-Based Interfaces
SBMA	Service-Based Management Architecture
SCP	Service Communication Proxy
SDF	Service Data Flow
SMF	Session Management Function
SoR	Steering of Roaming
SP	Service Proxy
SPC	Service Proxy Controller
SPF	Service Proxy Forwarder
SPR	Subscriber Profile Repository
SSL	Secure Sockets Layer
TEID	Tunnel Endpoint Identifier
TLS	Transport Layer Security
UBI	Ubitech (partner of FUDGE-5G)
UDM	Unified Data Management
UDR	Unified Data Repository
UE	User Equipment
UPF	User Plane Function
UPV	Universitat Politècnica de València (partner of FUDGE-5G)
VCPU	Virtual Central Processing Unit
VF	Virtualized Function
VM	Virtual Machine
VNF	Virtual Network Function
VNFD	VNF Descriptor
VPLMN	Visited PLMN
WP	Work Package

Executive Summary

FUDGE-5G's Work Package (WP) 2 is composed of five tasks:

- T2.1: Unified Service Based Architecture Platform.
- T2.2: LAN in 5G Environments.
- T2.3: Cloud Native Service Orchestration.
- T2.4: Disintegration of Network Functions as Micro-Services.
- T2.5: Platform Continuous Integration in a Sandbox Environment.

This deliverable reports on activities carried out in T2.3 and T2.4. Specifically, in Section 1, we give an introductive overview of the concepts of *cloud-native* and *microservices.* We will highlight their differences and recall how they are tightly related and constitute a major paradigm for the implementation of modern Information Technology (IT) services. These concepts are enablers for FUDGE-5G's Unified Service Based Architecture [D2.1], [D2.2].

Section 2 is dedicated to cloud-native service orchestration, which in FUDGE-5G is common for vertical applications and 5G Core (5GC) Network Functions (NFs) [D2.1]. Section 2.1 and Section 2.2 elaborate on the characteristics and requirements that cloud-native software components must have in FUDGE-5G's view to be suitable for orchestration (or "*orchestrable*"), in the sense defined in [D1.2], [D2.1]. Further, Section 2.2 addresses the role of "sidecars" and proxies in cloud-native service orchestration.

Section 3 focuses on the adoption of microservices and of microservice-based architecture principles in designing 5GC NFs, motivated by the possibility of developing and deploying core network functionalities that take full advantage of orchestrability and *cloud-nativeness* (i.e., the condition of a function or software component of being cloud-native). Section 3.1 recalls the service-based architectural approach of 5G, whereas Section 3.2 analyses the criteria, advantages, and limitations of a microservice-based architectural approach in 5G. Further, in Section 3.2.4, we propose a possible degree of decomposition into microservices of some functionally heterogeneous 5GC NFs, chosen for their specific role within the 5GC and their importance as enablers of advanced 5G functionalities. Such work is in line with the architectural specifications of the 5GC [TS23.501] at an inter-NF level but goes beyond the state of the art in the internal design of the NFs, fostering the adoption of microservices in their development. Some integration testing results concerning these NFs are presented in Section 3.2.4, and further results will be reported in [D2.5].

Finally, some concluding remarks appear in Section 4.

As opportunely pointed out in the text wherever appropriate, part of the work described in this document was reported in [Cen+22], [Ish+22], and [NGMN22] with explicit acknowledgments to the FUDGE-5G project.

Table of Contents

List of Figures

List of Tables

1 Introduction: Key Concepts, Cloud-Native, and Microservices

During the last few years, the paradigm of *microservices* has gained momentum in various IT fields, embracing a multitude of business cases and targeting plenty of heterogeneous application scenarios. The concept of microservices yields from the general observation that end-to-end digital business services and the underlying computerized functionalities are becoming more and more complex to develop, deploy, interconnect, manage, heal, and update [CNCF], [Mic22]. Often, the classical approach for the design of digital services (usually referred to as *monolithic*) does not fully meet the requirements of the most recent use cases in terms of flexibility, adaptability, continuous development, scalability, or resource management of the underlying applications [Mic22].

To overcome this, the microservice approach is based on identifying independent functionalities (and the corresponding data modules) within the main service, removing unnecessary dependencies, and isolating them into modular, self-standing logical and operational blocks that relate with each other in a Service-Based Architecture (SBA) via dedicated interfaces and through an event distribution bus. Microservices conceived in such a manner can be developed, run, and orchestrated independently, because each of them is a self-contained coherent entity. They can be programmed in different languages, and the computer-scientific development and maintenance of each of them can be adapted to evolving needs without having to reshape the whole service architecture or without impacting how other microservices operate.

The composition of microservices into bigger end-to-end services is therefore flexible and scalable thanks to modularity, and technology-agnostic thanks to the SBA and the well-defined interfaces. An architecture made of microservices facilitates a software development process based on continuous delivery, enabling the implementation of small changes of the application via rebuilding and redeploying a single or few microservices. Further, it adheres to principles like fine-grained interfaces, allowing independent deployment of services, business-driven development, and the DevOps approach [MB20].

Thanks to their features, microservices can play very naturally the role of components of *cloud-native* (network) functions. Cloud-native refers to a function or an application that is specifically conceived for running in the cloud, taking advantage of the cloud's infrastructural and management capabilities. Cloud-native applications are suitable for automated orchestration and monitoring, and profitably rely on the cloud's built-in resilience, scaling, and self-healing mechanisms. The Cloud Native Computing Foundation (CNCF) states that "[…] containers, service meshes, microservices, immutable infrastructure, and declarative Application Programming Interfaces (APIs) exemplify" the

cloud-native approach [CNCF]. In particular, *containers* are technological enablers for cloud-native software. They package code, libraries, dependencies, and run-time into a single binary image, so that they can be moved easily and can run in any environment. Instances of containers are executed by a common operating system. They fit well in the microservice framework since each container occupies a well-defined "slice" of the hosting infrastructure and is isolated from the other containers.

Notice that the natural suitability of microservices for cloud environments does not strictly imply that all cloud-native functions are microservice-based, nor that microservices can only be deployed in the cloud. In some contexts, when the abuse of terminology does not create confusion, microservices and cloud-native functions may be identified, but here we prefer to stress their difference, graphically represented in the Venn diagram of Figure 1-1.

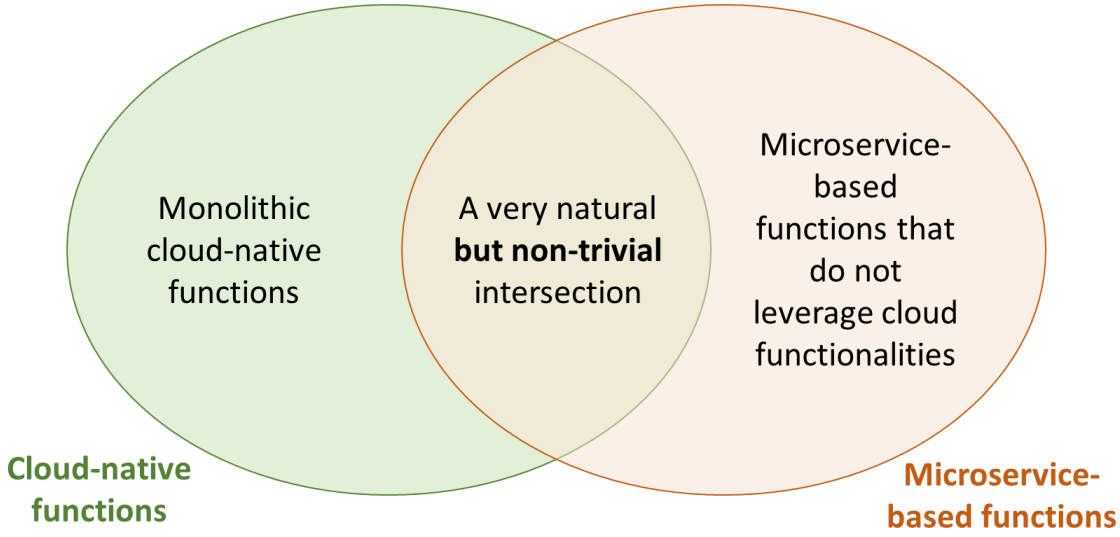

Figure 1-1 – Difference between cloud-native functions and microservices at a glance.

In the next sections, we will discuss how cloud-nativeness and microservices suitably support an efficient orchestration and deployment of network services and functionalities.

2 Cloud-Native Service Orchestration

Orchestration embraces automation and orchestration techniques have evolved to cope with vertical industries' requirements, specially to accommodate heterogeneous services on top of the generic 5G environment while facilitating the provisioning procedure by means of vertical-oriented information/data models and APIs. This is critical for the commercial survival of today's service providers, who face rising technology complexity, increased commercial pressures, and accelerated technology refresh cycles. Another missing part is the capability of the Enterprise Services (cf. [D1.2], [D2.1]) to fully exploit the flexibility of Network Function Virtualization Infrastructure (NFVI) and dynamically interact with these orchestration primitives and the programmable network. In that sense, a service orchestration, considering the needs of the core network and of vertical applications is a key factor for success in the enterprise market. The role of the orchestration process at the service level is to handle the deployment and real-time management of the aforementioned services, while inherently providing elasticity and compliance with certain high-level service policies. This process essentially decouples the service layer management procedures from the network layer management, providing service awareness to the underlay management (resource allocation, slice creation and management) of the programmable infrastructure, and compatibility with any network orchestration solution.

The overall concept is aligned with modern complex services, which are designed over distributed architectures with edge processing capabilities. Such services consist of a chain of cloud-native components that can be managed individually, i.e., microservice-based software development design patterns a.k.a. *12-factor app* [Wig]. Each microservice (component) is bundled in an orchestration-friendly way, i.e., as a Virtual Machine (VM) image, a container, or even a unikernel maintaining backward compatibility with all three industry-leading approaches, while offering important telco-interplay capabilities such as bi-directional interaction with an Operations Support System (OSS) and slice management system.

From the system design point of view, service orchestration can be made possible via an orchestrator that provides an open development environment, utilizable by application developers to create and onboard their application microservices (components) adopting a unified programmability model and a set of abstractions. In its northbound end, such an orchestrator interfaces directly with the frontend user for the onboarding of the requested application microservices, and, in its southbound interface, it is connected either directly to the programmable infrastructure or the OSS that manages the programmable infrastructure for providing the application level requests that should satisfy the service's functional and operational requirements.

2.1 Design and Development Approach

In FUDGE-5G's vision, a typical orchestrable service (such as Enterprise Services [D1.2], [D2.1]) consists of several cloud-native microservices or components – i.e., components that must collaborate in order to fulfil a broader operational scope. Collaboration implies that these components form a logical graph based on their dependencies. As described in Section 1, the term cloud-native refers to specific properties that these components should have to be ported to the cloud. However, despite the baseline abstract definition of "cloud-nativeness", let us point out that the term has never (and probably cannot fully) been strictly formalized and may lead to highly different realizations of cloud-native functions from vendor to vendor. On top of that, the flexibility of programmable underlays introduces additional parameters that should be taken under consideration for a "strict" definition of a cloud-native component. Programmable systems allow the dynamic reconfiguration of provisioned resources (e.g., compute, memory, storage, QoS, security), which are capabilities that may be overlooked during the development of cloud-native service.

In the frame of FUDGE-5G, we consider that an Enterprise Service is composed of a set of software components that are orchestrable in a cloud-native fashion. Such components have the following characteristics:

(A) Its configuration parameters are exposed separately from its own code (config file), along with their metadata (e.g., which are the acceptable values? Can these parameters change during the execution?), and outside of the component itself.
(B) It exposes chainable interfaces that will be used by other cloud-native components to create a service graph.
(C) It exposes quantitative metrics regarding the QoS level related to the cloud-native component.
(D) It is typically wrapped in a lifecycle-management programmability layer to be used during the placement of a service graph in the platform resources.
(E) It is stateless to be horizontally scalable by design.
(F) It is reactive to runtime modification of offered resources to be vertically scalable by design.
(G) It is not strongly dependent on physical storage, network, and general-purpose resources.

Regarding (A), it could be argued that, if a cloud-native component entails a specific configuration layer, it is extremely crucial to be reconfigurable by design (i.e., to adapt to the new configuration without interrupting its main thread of execution).

Regarding (B), it is clear that dynamic coupling of services is highly valuable only when an actual binding can be fully automated during runtime. This level of automation raises severe prerequisites for the developed cloud-native components. The "profile" of the chaining should be clearly abstracted. Such profile includes the offered/required datatype, the ability

to accept more than one chaining, etc. These metadata are often stored in highly efficient key-value stores (such as [Consul]) to be queried by requesting cloud-native components.

Regarding (C), it should be noted that, while cloud-native component-agnostic metrics are easily measured, the quantification of business-logic-specific metrics cannot be performed if a developer does not implement specific application-level probes.

Regarding (D), the recent developments in the virtualization compendium provided novel management capabilities. For instance, the live migration from one hypervisor to another one has been now integrated as a core built-in feature of KVM [KVM] for more than a year. Hence, a cloud-native component that is running on a specific platform may be literally "transported" to another one with minimum down-time. In other words, both cloud-native components should expose a basic programmability layer, which handles the high-level cloud-native component lifecycle (e.g., remove chained dependency).

Regarding (E), any service that is stateless can scale easily with the usage of some "facilitating" services such as network balancers or web balancers. The emergence of the infrastructure programmability model will progressively "offload" this task to Virtualized Functions (VFs) that are controlled by a cloud orchestrator. Ensuring the stateless behavior of a service graph is a challenging task since the entire business logic should entail stateless behavior in order to be horizontally scalable by design.

Regarding (F), taking under consideration the developments in hypervisor technologies and Operative System (OS) kernels in the last two years, it could be argued that the barriers of dynamic provision and de-provision of resources on an operating system have been raised. However, the dynamic provisioning of resources to a virtualized system does not imply that these resources are automatically bound to the hosted cloud-native component. On the contrary, in most of the times the cloud-native component has to be (gracefully) restarted in order to make use of the new resources.

Finally, regarding (G), it should be noted that not every valid cloud-native component is capable to be ported to a modern infrastructure since the cloud-native component cannot be hosted in all PaaS or IaaS providers. A different approach to tackle this is to provide the capability to declare some dependencies or specific platform features that are required to support the optimal functionality of the component before the instantiation of it, e.g., the need for specific hardware like a Graphics Processing Unit (GPU).

2.2 Requirements of Service Orchestration

The cloud computing/programmable infrastructure-as-a-code paradigm, along with the support of microservice-driven architecture, generated additional requirements. These include:

- rapid provisioning of compute resources,
- basic monitoring,
- rapid deployment,
- easy to provision storage,
- authentication/authorization,
- standardized interfaces (e.g., Remote Procedure Call – RPC, HTTP).

The requirements of 5G-ready services coincide with the aforementioned requirements. Yet, they are much more intensive, since provisioning of infrastructure should be "instantaneous", topology is continuously changing, delay tolerance is minimum, etc.

In the scope of FUDGE-5G, the service is distributed and consists of cloud-native components that rely on telco infrastructure as a means of network abstraction. The telco infrastructure has to operate on top of a programmable 5G environment. Towards these lines, the FUDGE-5G architecture relies on a solid interplay between various logical layers such as the actual data plane, the control plane and the configured virtualized resources that are offered by the infrastructure provider as a proper slice. The concept of the service, the modelling and supported functionalities can be considered applicable to various vertical contexts.

2.2.1 Service Orchestration Primitives

As stated above, FUDGE-5G considers Enterprise Services as consisting of multiple components that can be deployed on top of a programmable infrastructure. These components, when combined with each other, can be represented by a graph that captures the entirety of the service. In other words, a service is represented by graph where components are vertexes and edges are the component-links. For the sake of clarity, the formal metamodels of this graph and the component are analyzed in [D1.3].

Without delving into the details of this metamodel, it should be clarified that the components that comprise the graph should satisfy the requirements of cloud-native components that are written in Section 2.1 and are aligned with the 12-factor methodology [Wig]. Being cloud-native is essential for the proper placement of these components to virtualized environments. Moreover, to be in line with the state-of-the-art principles of cloud services, components should be also programmable. Programmability is provided by a transparent proxy which is running on top of the component. This proxy is also addressed

as "*sidecar*". A service graph consists of programmable components, where each of them is proxied by an individual sidecar. As written in [D1.2], in FUDGE-5G the service provider can interact with a development environment that is responsible for packaging a cloud-native component in a proper format, making it usable by the control plane (service orchestrator). In general, it is developed to support all pre-deployment steps of a vertical service. Such steps include the proper packaging and the proper combination of cloud-native components in the form of complex graphs. Cloud-native components and service graphs will be persisted in a repository to be searchable (in general Create, Read, Update, Delete – CRUD – operations) by service developers. On the other hand, the logically centralized control plane is responsible for the orchestration, monitoring and policy enforcement over the deployed service.

2.2.2 Proxies in the Service Orchestration

The deployment of an Enterprise Service (cf. [D1.2],[D2.1]) relies on an *abstracted network layer***.** The approach of network abstraction is considered as a state-of-the-art approach from industrial giants of the cloud industry. Indicatively, Google, HP, Red Hat, Twitter have converged to a high-level architecture regarding the network abstraction of cloud-native services. The core element of this architecture is the *component-proxying*, i.e. the fact that each cloud-native component is interacting with other components through a proxy. The interaction between the proxies constitutes the data plane, while the configuration actions of the proxy are based on information gathering that is performed by the proxies per se. The information gathering, along with the actions' enforcement, is addressed as control plane.

Since proxies are core part for the coordination and the deployment of vertical services, they undertake several tasks, such as

- dynamic service discovery,
- load balancing,
- Transport Layer Security (TLS) termination,
- circuit breaking,
- health checking,
- traffic shaping (layer 7),
- publication of metrics.

To perform dynamic service discovery, the proxy assumes that the entire 5G-enabled service is supported by a service registry (such as Consul [Consul]) to keep track of the existing instances. It also assumes that new instances are automatically registered with the service registry and unhealthy instances are automatically removed. Additional functionalities are that they enable to dynamically load and apply layer-7 filters. Such filters

will act as enablers of several actions. Thus, during policy formulation, the available filters that can be potentially applied determine the type of layer-7 actions that can be performed. All the described functionalities are leveraged by the orchestrator, which interacts with the intelligent proxies, in order to properly configure them and to enforce any policy that has been defined by the customer.

3 Decomposition of 5G Core Network Functions

The increasing softwarization of mobile core NFs is fostering the evolution of the mobile network architecture itself, which in 5G has moved towards a service provider/consumer framework and service-based interfaces. Moreover, the 5G architecture was conceived to be adapted to the exploitation of mobile technologies also for dedicated non-public uses as an alternative to nation-wide deployments. The 5GC is a crucial part of this architectural paradigm shift, which aims at closing the gap between the telecommunications domain and the IT world at large. Keeping in mind the topic of Section 2, the objective of this section is to shed some light on the degree of possible pervasiveness of software design concepts like microservices and cloud-nativeness in the context of 5G mobile networks.

Specifically, we will:

- discuss the general criteria for the decomposition of 5GC NFs into microservices, also analysing the differences between the public and non-public use case,
- apply such an approach to an exemplary set of 5GC NFs,
- propose ways forward towards the adoption of these concepts in beyond-5G mobile systems.

3.1 Microservices and 5G Networks

3.1.1 Service-Based Architecture in 5G Networks

Recently, thanks to the advent of an SBA also for 5G networks, the approaches of cloud-nativeness and microservices have become of interest for mobile telecommunications. When transitioning from generic distributed systems to mobile network systems, we have to account for NFs rather than generic business functions. As part of a critical infrastructure, 5G NFs may have to satisfy stringent requirements in terms of latency, dependability, and security (for security aspects, see Section 3.1.3). In the past, this was the motivation of having dedicated hardware implementing all network segments and functionalities. However, since the 4th generation of mobile networks (4G), industry-driven standardization initiatives have raised [NFV12], based on the idea to replace physical NFs with Virtual NFs (VNFs).

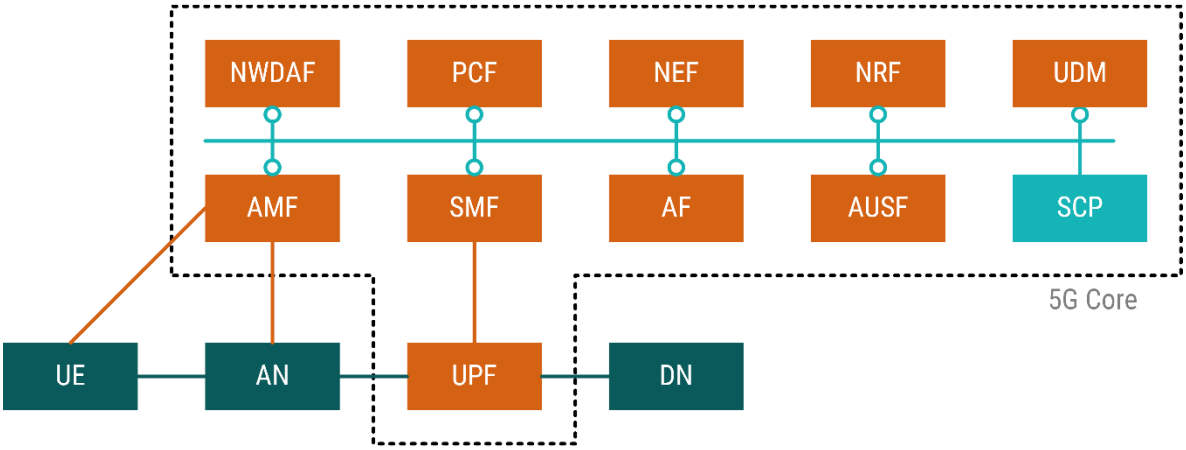

Figure 3-1 - 5G system architecture and network functions as in [TS23.501].

The trend was completed with 5G networks, starting with 3GPP's Release 15 [TS23.501]: a paradigm shift in the system architecture was introduced on how control-plane NFs communicate among each other. In pre-Release 15 systems, all NF instances had a strict one-to-one relationship with each other and used application layer protocols such as Diameter. With the advances of cloud solutions that can scale on demand, Release 15 adopted an SBA for their 5GC (cf. Figure 3-1), with the following changes:

- Decomposition of the mobile network's core functionality into smaller independent and self-contained NFs.
- Introduction of the concept of consumer (endpoint/clients) and producers (service endpoint/servers) without strict requirements on which consumer is allowed to communicate with which producer.
- Introduction of Service-Based Interfaces (SBIs) for the majority of 5GC NFs, moving to HTTP/2 as the application layer protocol and JSON-encoded payload.

All 5GC control-plane NFs interact within an SBA, except for the interfaces towards the radio access network (RAN), the User Equipment (UE), or User Plane Functions (UPFs), called N1, N2, N3, N4, N6, N9. Specifically, a key NF called Network Repository Function (NRF) manages NF service registration and discovery, enabling NFs to identify appropriate services in one another. After the discovery phase, a NF service producer can deliver notifications to a NF service consumer provided that the latter subscribes to those notifications.

In Release 16, another key component was added to the 5G system architecture, i.e. the Service Communication Proxy (SCP), taking over the responsibility of proxying traffic between a consumer and producer instances. The deployment of an SCP is optional, and the SCP does not expose a 5GC service itself. Instead, it is used for "indirect communication between NFs and NF services" [TS23.501] and is an addressable endpoint via an IP address or a Fully Qualified Domain Name (FQDN). If no SCP is deployed, consumers and producers communicate with each other without an SCP, and Release 16 refers to that as a "direct communication."

Hence, the 5G SBA is already compatible and well aligned with a (micro)service-based approach. In the work of T2.4 of FUDGE-5G, we aimed at walking further onto this path, and we have focused on the possibility of decomposing control-plane 5GC NFs into actual microservices, keeping in mind the peculiarities of mobile telecommunications with respect to the rest of the IT world.

3.1.2 Service-Based Architecture and Microservices for Public and Non-Public 5G Networks

The 5G standard has been transforming networks from pure communication systems into powerful and flexible connect-compute infrastructures, apt to serve the purposes of various *vertical* sectors, like Industry 4.0, automotive, healthcare, or energy. These verticals leverage the high adaptability of 5G networks to impose their own specific (and often restraining) requirements on the demanded telecommunication performance or on the use of virtual and physical resources. As opposed to traditional national monolithic networks, which were not designed to properly satisfy all the potentially very diverse sets of conditions imposed by the new use cases and applications, 5G network slicing [TS23.501] provides operational isolation of different parts – the *slices* – of the network, thus supporting multiple vertical customers at once.

In this context, as a complementary solution to network slicing, the deployment of 5G Non-Public Networks (NPNs) [TS23.501] is becoming a more and more common choice for private companies and public institutions that want to maximize the control capability on their own private telecommunication systems, acquiring independence from traditional mobile network operators. Such NPNs fill the performance and service gaps left unaddressed by nation-wide Public Land Mobile Network (PLMN). Consequently, interoperability among local NPNs and between each NPN and PLMN has become a major focus point for all the parties involved in the 5G conception and deployment. In this perspective, SBA and the decomposition of NFs into microservices are envisioned to play a key role to enable the efficient coexistence of NPNs and PLMNs in a truly dynamic and automated manner.

This holds true a fortiori whenever some virtual or physical resources are deployed at the edge of the network or when they are shared, for example in cases where a PLMN operator provides localized and private services to a customer (e.g., by means of a dedicated network slice) while running its usual operations in the same area. In fact, inter-network communication and end-to-end network service orchestration can all be managed faster and more efficiently when functions and applications are decomposed into independent, self-contained, and easily accessible pieces of software, like microservices.

FUDGE-5G's work on microservices and 5GC NFs has this framework as a reference. We remark, however, that there exist prominent NPN application scenarios which require to

minimize network interoperation and external points of contact. This is the case, for instance, of high-privacy networks with very stringent security requirements or NPNs deployed for specific use cases that benefit from being isolated from other systems or simply are so by nature, as those dedicated to Public Protection and Disaster Relief (PPDR) or deployed in remote locations (e.g., mid-sea oil stations, underground mines, even airplanes). For these kinds of NPNs, interworking and dynamic orchestration might not be as important as the apparatus' resilience and security, thus the choice of monolithic architectural solutions may still be taken into consideration. Nonetheless, microservices remain beneficial in this case for technological forward-compatibility (cf. Section 3.2.3).

3.1.3 Security Considerations for 5G Core Networks

With the transition to an SBA for the 5GC, 3GPP follows the semantics of the Internet for endpoints to communicate with each other including aspects around authentication and authorisation. As the Internet operates as a decentralised system, both endpoints (client and server) must implement methods to verify the identity of the other one including the authoritative verification if a certain communication or resource can be exchanged. Additionally, all the communication between the two endpoints must be secured prohibiting anyone to access the content of any information that is exchanged between the two endpoints.

For services publicly available over the Internet, different security procedures have been put in place for enabling several levels of securities mainly related to authorisation, but partially authentication too. Generally, TLS is a widely adopted protocol for securing the communication between a client and a server [Res18]. TLS is a certificate-based protocol that enables the verification of server identifies so that clients can be assured they are communicating with the entity they intend to start a "conversation". All root certificates are signed and issued by trusted authorities and can be verified online. As a result, each client application can independently verify a server and its certificate. Once the TLS session has been established, plain text information can be exchanged with the server, which is encrypted along the transport link. This is illustrated in Steps 1 in Figure 3-2.

Upon the completion of a secure channel between a client and a server and the validation of the server's identity, the client must authenticate itself allowing the server to verify the identity of the client. In order to do so, a TOKEN-based approach is widely used, where the client provides its credentials (certificate or username + password), as illustrated in Steps 2.

In Step 3, the client uses the token for every communication with the server, allowing the server to verify the client. This verification is often based on a shared master key across client and servers as part of the application.

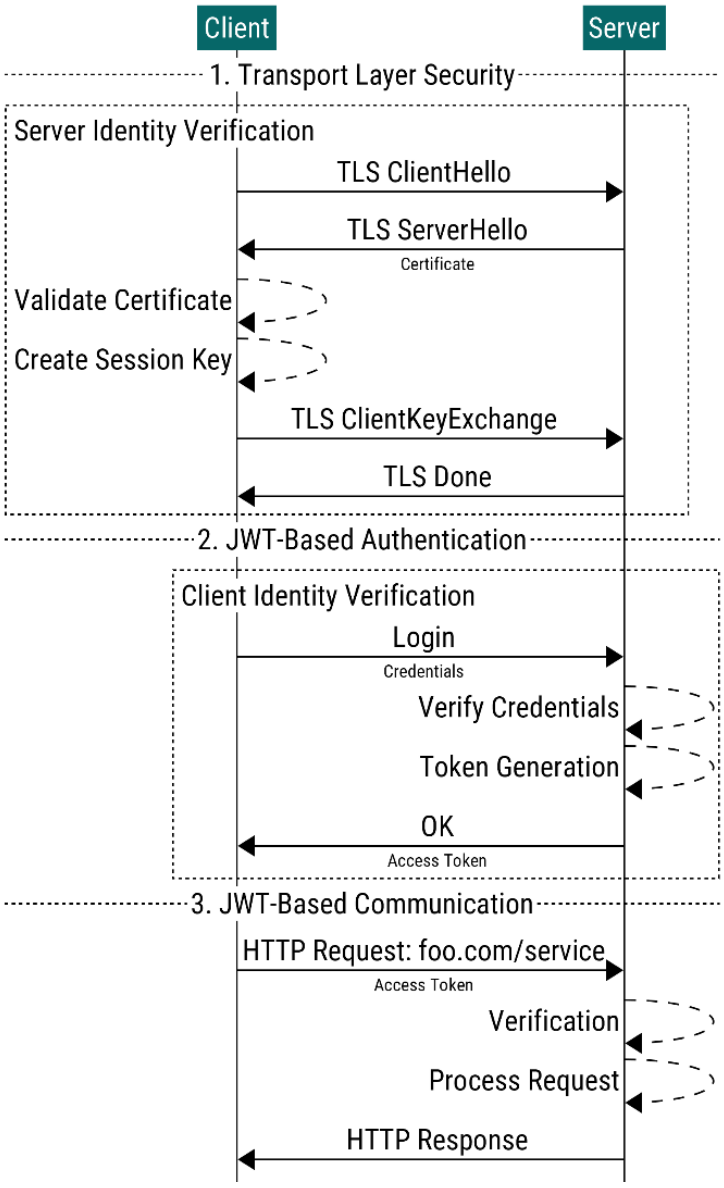

Figure 3-2 - Procedures for encrypted modern web-based communication.

All SBI-enabled 5GC NFs can utilise the TLS/SSL (Secure Sockets Layer) and Access Token procedures to verify, authenticate and authorise consumer and producer instances. Given the wide acceptance of these procedure for secure communication in the areas of banking, contractual agreements, and online commerce, FUDGE-5G assumes that once all NFs follow the procedures described herein, security does not need to be addressed separately and does not need require further attention in this deliverable.

3.2 5G Core Network Function Design Patterns

The goal of this subsection is to study the decomposability of 5GC NFs into microservices, consistently with the definition of the 5G SBA recalled in Section 3.1.1. This analysis is not trivial, especially because such a decomposition cannot be automatically obtained by creating one microservice for each of the services produced by a 5GC NF. The specific services produced and consumed by the 5GC NFs as defined by 3GPP [TS23.501], [TS23.502], are not fully compatible with the definition of microservices that we gave in Section 1. More precisely, as further explained in Section 3.2.2 and 3.2.4, such services are not always fully independent of one another and cannot always be associated with a single *bounded context* [Eva04]. Hence, it is not always possible to straightforwardly deploy as an actual microservice each 3GPP-defined service of a NF, in what would be a simple one-to-one mapping between a NF's logical sub-functionalities and the microservices into which it is split. The partition of a NF's functionalities into actual separate bounded contexts is necessary to reach an effective microservice decomposition.

After briefly overviewing some previous work on this matter and discussing the main underlying criteria, advantages, and limitations of a microservice-based approach, we are presenting in Section 3.2.4 the design work of FUDGE-5G's T2.4, which yielded some examples of 5GC NF decompositions into microservices. Our work is still aligned with 3GPP but considers CNCF's recommendations for microservice development and deployment.

3.2.1 Previous Work on Microservice Designs in 5G Core Network Functions

Some proposals to incorporate the concept of microservices in the design of 5GC NFs have recently appeared in literature. [DWWN20] reports on the cloud-native modular design and the implementation of the functional procedures of an Access and mobility Management Function (AMF) conceived for microservice-based architectures within the OpenAirInterface (OAI) project. The authors give a description of the different data and functional modules and the architectural implementation layers that constitute their AMF. However, we observe that the focus of [DWWN20] is more on the cloud-nativeness of such an approach and the compatibility with cloud environments, rather than an actual microservice-based design of the AMF itself.

Some vendors have proposed cloud-native 5GC designs that claim to embrace microservices principles [Ora20], [Sam20], however the lack of a shared approach to NF decomposition weakens the effectiveness of the design. Finally, related work exists on how to provision, manage, and automatically orchestrate microservice-based NFV service platforms and VNFs in 5G [dDWZ20], [Soe+18].

3.2.2 General Criteria for Network Function Decomposition into Microservices

We have identified two main approaches on how to make NFs interact via SBIs:

1. Based on interfacing the entire monolithic NF.
2. Based on identifying its sub-functions providing NF services to be interfaced.

The first approach is quite conservative but may be safer or more pragmatic while transitioning from monolithic NFs towards a full-fledged microservice architecture. It consists in implementing, whenever possible, SBIs between NFs belonging to the legacy monolithic software architecture. The advantage of this approach is that the vendor does not need to decompose the monolithic NF into NF services, rather designing a unique SBI to interface different monolithic NFs.

On the other hand, the second approach is more aligned with the spirit of 3GPP specifications, which provide both

1. The functional description of each NF [TS23.501].
2. The services provided by each NF [TS23.501].

As further elaborated in Section 3.2.4, our NF decomposition work follows the second approach and leverages an *ingest API* or similar functionality for the integration into 5G's main SBA and SBI of the cluster of microservices into which each NF is decomposed, implementing the specific sub-functionalities.

As a matter of fact, in principle, each of the NF services offered by a NF shall be self-contained, reusable and use management schemes independently of other NF services offered by the same NF (e.g., for scaling, healing). This allows for agile dynamic scaling (horizonal/vertical), independent lifecycle management, and data isolation, in a framework that is highly compatible with microservices. Off-the-shelf methodologies like, e.g., the 12-factor app [Wig], exist on how to convert a monolithic software into a set of microservices.

In the specific case of a 5GC, the exercise consists in decomposing a monolithic NF into a set of sub-functions, each one implemented as a microservice, that form the NF as a whole. As we mentioned, 3GPP itself defines the functionalities of each NF, and helps making such an exercise easier especially for control-plane NFs since the services each of them provides are specified too.

In other words, we may state that the 3GPP specifications provide initial (although not complete) guidelines for a logical decomposition in microservices of service producers. On the other hand, a service consumer does not offer any services and this criterion cannot be directly applied to them.

Nonetheless, the decomposition of service producers and consumers can be done as per the criteria below, that FUDGE-5G has also recently brought to the attention of the Next Generation Mobile Network (NGMN) Alliance [NGMN22]:

- *Bottleneck Mitigation and Parallel Execution* – An NF functionality that poses a bottleneck in terms of, e.g., performance within the same NF or direct interactions with other 5GC NFs may indicate that an ad-hoc microservice should be created for that. The intention would be to allow the utilisation of more compute capabilities for this microservice to mitigate the bottleneck.
- *Resilience* – Key functionalities of an NF with high-availability requirements shall be isolated in dedicated microservices, so to increase resilience against failures and increase the dependability of the NF.
- *State Dependency* – There can be dependencies between NF services within the same NF due to sharing some common resources/state, e.g., context data. In general, the state of an application is its condition or quality of being at a given moment in time – their state of being. In general, whether an application is stateful or stateless depends on how long the state of interaction with it is being recorded and how that information needs to be stored. In this context, there must be a criterion that depends on the data that a NF service needs to read/upload/retrieve/pre-process for running. It may be detrimental (in terms of latency or computational effort) to read/upload/retrieve/pre-process the same amount of data for several microservices, especially if of consistent size, and to keep it in sync across different instances of the same NF. This does not preclude that NF services offered by a single NF are managed independently of each other.

Finally, we remark that decomposing a monolithic NF in microservices may increase the risk of breaches due to, e.g., unauthorised access. Thus, it is of paramount importance to adhere to state-of-the-art, recognized security technical implementation guidelines while developing the various microservices, so to assure, e.g., built-in authorization and authentication by means of JSON Web Tokens.

3.2.3 Advantages and Limitations of a Microservice-Based 5GC

Adopting the proposed criteria for NF decomposition is likely to bring the following specific business and functional benefits to vendors and operators.

1. *Product Flexibility* – A fine-grained modular software architecture allows a vendor to customize its solution more easily and flexibly, according to the operator's or vertical customer's requirements, addressing the heterogeneity of needs that differentiate public nation-wide network operators and private network owners. Moreover, this is a key feature in some NPN deployment, e.g., ad-hoc private

networks in PPDR use cases, characterized by the need for a compact form factor, to be deployed for instance in a van (cf. FUDGE-5G's PPDR use case [D1.1]).

2. *Forward Compatibility* – Mobile network standards are in continuous evolution. 5G has not been fully deployed yet, but the community has already started investigating the 6th generation of mobile networks (6G). A microservice-based design and development of NFs allows for a more straightforward and natural upgrade of a NF's services and functionalities.
3. *Data Segmentation* – Having separate database implementations, tailored to each microservice's needs and without redundant information, makes user and network data better isolated, thus more easily manageable. This holds even more in distributed deployments, e.g., NFs over remotely located locations or over different network slices.

Nonetheless, the proposed approach may also feature some downsides, which need to be evaluated depending on the application scenario, or at least one needs to account for some trade-offs regarding the following subjects. Considering that the plain, standard 5GC SBA is already microservice-compatible if one develops each entire NF as a single microservice, a further complexification and decomposition of the network architecture could bring concerns regarding:

1. *Development Effort* – Deeper decomposition allows a more efficient reusage of functions in different end-to-end services (in terms of flexibility, as written above). Nonetheless, it may introduce further complexity within sub-functions and how they interact with each other, e.g., too many internal interfaces to develop or excessively complex event distribution systems within the function, especially at the beginning of the decomposition process. In general, a (very) fine-grained system decomposition may increase the overall development effort, especially when systems are very complex and are made of highly numerous functions and subcomponents.
2. *Security* – 5G comes with more stringent security requirements than the previous generations, and many high-security use cases are envisioned especially for NPN deployments. While NF decomposition helps, e.g., by improving the database security thanks to segmentation and isolation, on the other hand the exposed attack surface as well as the number of potential vulnerabilities increase.

Once again, both these limitations can be mitigated by a proper adoption of the Dev(Sec)Ops approach [MB20], as well as by adopting common security standards for encrypting communications and for authentication and authorisation (cf. Section 3.1.3).

3.2.4 Examples of Decomposed 5GC NFs

Keeping in mind the identified general criteria and the whole discussion elaborated so far, in this subsection we try to substantiate them diving into the decomposition of specific 5GC NFs. When passing from theory to practice, the analysis of the multiple functionalities of each NF indicates that the NF services specified by 3GPP may not be fully independent of one another, thus complicating the design phase. It results that it is not always possible to associate them with a single *bounded context* [Eva04] in the 5G domain. The work of T2.4 that we report here considers that a proper decomposition into microservices needs to identify the bounded contexts that each functionality of a NF addresses in the 5G domain. Then, each identified bounded context can be implemented as a microservice, within the same NF. For compatibility with the standard, 3GPP-specified SBIs can eventually be exposed through another microservice, an *API Gateway* or an *Ingest API*, which also shields the internal microservices in the fashion of a De-Militarized Zone (DMZ). The Ingest API basically plays the role of a reverse proxy, whereas the API Gateway implements further logic and additional aggregation functionalities, necessary when a service request from a consumer NF is concretely mapped towards more than one of the microservices into which a producer NF is decomposed.

According to this approach, in the following of this section, we propose a possible degree of decomposition of some functionally heterogeneous 5GC NFs via:

1. The identification of independent functional or data modules for the services provided by an NF.
2. The identification of the dependencies with other NFs.
3. The identification of the relation among services within a NF's architecture.

The global complexity and numerousness of the 5GC's NFs prevents us from an exhaustive and omni-comprehensive analysis of their decomposability into microservices. As anticipated in [D2.1], we have chosen therefore to restrict ourselves to the following list for the following reasons:

- Unified Data Management (UDM) because it is a strictly necessary function even in minimum-footprint deployments and because it plays an essential role in enabling the statelessness of the other NFs.
- Authentication Server Function (AUSF) because of its tight interaction with the UDM and the UE authentication features it implements.
- User Plane Function (UPF) because it is mandatorily required in a minimum-footprint implementation of a 5G system and because of its peculiarity of providing all the user-plane functionalities at the 5GC level.
- Policy Control Function (PCF) because of its central role in establishing and managing (in collaboration with the Session Management Function, SMF) the QoS policies in PDU sessions.

- SMF, because of its tight collaboration with the UPF and the PCF.
- Network Exposure Function (NEF) because it is an important enabler of 5G advanced features, tightening and deepening the interaction between the 5GC and external applications.
- Cell Broadcast Centre Function (CBCF) because it substantially consists of a trusted Application Function (AF) and because it is utilized in the use cases considered by FUDGE-5G (e.g., the PPDR use case) [D1.1].

Overall, the selected NFs are highly representative of the main 5GC features, architectural peculiarities, and functionalities of interest for FUDGE-5G.

3.2.4.1 Unified Data Management

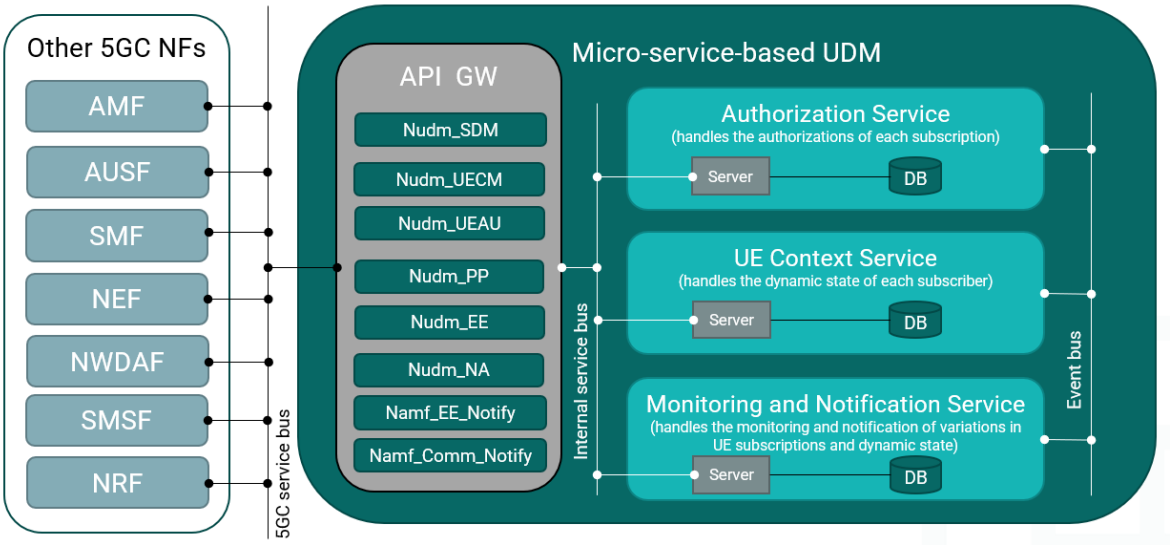

Figure 3-3 - Proposed microservice-based UDM design.

The UDM function supports several functionalities including user identity and authorization [TS23.501]. These functionalities are broken down into several services [TS23.501] used mainly by the procedures specified in [TS23.502].

As shown in Figure 3-3, the analysis of the functionalities has led to the identification of three bounded contexts [Eva04]:

- *Authorization*, to grant service authorization to positively identified users, based on operator policies, including subscription data.
- *UE Context*, to provide access to the dynamic state of the services being provided to the user.

- *Service Events*, to monitor events that may require a change in the services provided to the user and notify consumer functions who have manifested interest in those events.

The identified bounded contexts are implemented as microservices, viz.:

- Authorization Service,
- UE Context Service,
- Monitoring and Notification Service,

and are used by a fourth microservice, the API Gateway, which exposes the UDM functionality through the 3GPP-specified SBIs, ensuring the security of the underlying services at the same time.

Data is stored within the microservices, as required by a microservice-based architecture, but the use of an external Unified Data Repository (UDR), particularly for a cloud-native implementation is not excluded.

It should be highlighted that, as opposed to other UDM designs proposed in literature [OraU20], the proposed architecture ensures that the three "backend" microservices do not share data among each other.

Within T2.4's activities, ATH developed a microservice-based prototype of UDM, designed and implemented as described in this section. The functioning and design features of this prototype were validated in a dedicated demonstrational setup in ATH's R&I labs, whose results are also published in [Ish+22]. In such a setup, the microservice-based UDM is part of a proof of concept aimed at demonstrating the implementation of an on-demand provisioning procedure of a Network Slice Subnet (NSS) composed of VNFs from potentially different vendors. The demonstration includes a Network Management System (NMS) conforming to the 3GPP Service-Based Management Architecture (SBMA), an ETSI MANO orchestrator, and a NFVI. In particular, this work demonstrates the provisioning, configuration, and control of the exemplary NSS shown in Figure 3-4, composed of two NFs: the microservice-based UDM and an NRF, implemented by different developers and mimicking vendor-independence. Whereas the UDM is an agglomeration of independent microservices and is "SBMA-aware", the NRF is monolithic and does not expose SBMA-compliant management services.

Notice that for the mere purpose of validating our NSS management system, two NFs are sufficient, and the whole prototype focuses more on the configuration, management, and control features of the system, rather than on the "richness" in terms of deployed functionalities of the NSS itself. In this sense, the UDM and NRF are two good candidates for a "minimum footprint" NSS because they do not depend on the functionality of other NFs in the 5GC.

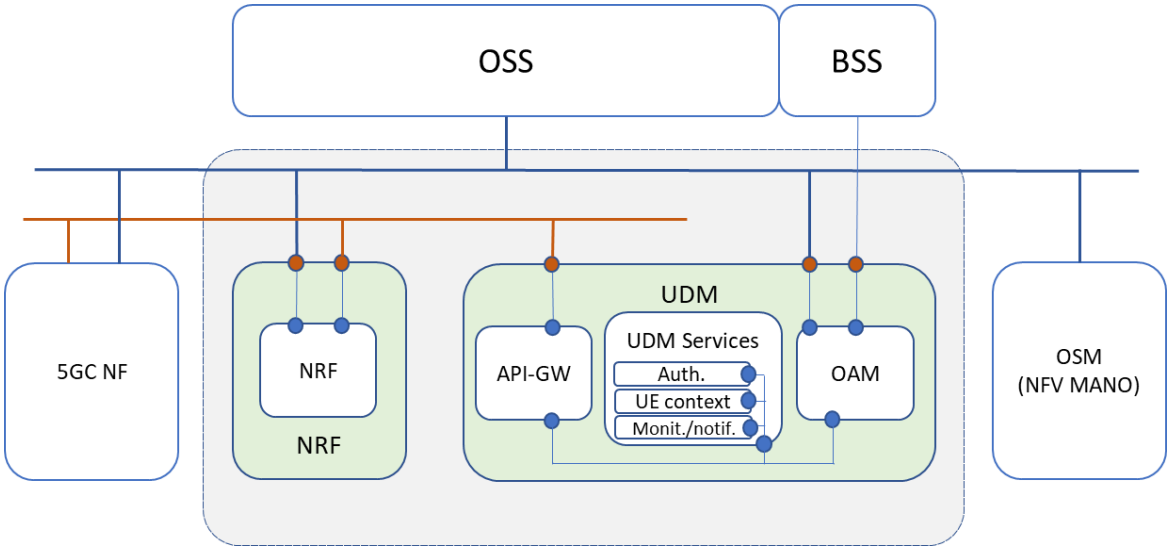

Figure 3-4 - Prototype of network slice subnet with microservice-based UDM.

The demonstration described in [Ish+22] comprises three phases: *preparation*, *commissioning*, and *operation* of the prototyped network slice.

1. In the preparation phase, the NSS, including its networking, is designed, and that information is provided to the NMS and NFV MANO. The NSS corresponds to a Network Service (NS) in NFV MANO [TS28.541], and is described in a Network Service Descriptor (NSD) [IFA014]. The NSD specifies the required VNFs – described in VNF Descriptors (VNFDs) – and their networking requirements.
   Each VNF is made up of one or more VNF Component (VNFCs) and specifies the component's *Compute Storage* and *Network* resource requirements. For the scope of FUDGE-5G's T2.4, it is important to highlight that each UDM microservice was modelled as a different VNFC, thus permitting the demonstration of the instantiation of a VNF with multiple VNFCs and validating the microservice-based approach in an OAM framework. The NSD and the constituent VNFDs are defined *yaml* files and together with the software images of the constituent VNFs are on-boarded in Open-Source MANO (OSM), the ETSI MANO orchestrator.
2. In the commissioning phase, the NSS is created via the Network Slice Subnet Management Function (NSSMF), which is part of the NMS. The NSSMF requests ETSI MANO to create and instantiate an NS instance based on the specified NSD [SOL005]. The NSSMF fetches the information on the instantiated NS from ETSI MANO, determines the instantiated VNFs, together with the allocated virtualized resources, including networking, and requests the Network Function Management Functions (NFMFs) of the VNFs to create the VNF-related Managed Object Instances (MOIs). Finally, the NSSMF creates a MOI for the instantiated NSS and notifies the NMS.
3. The NSS is now configured and operational. During the operation phase, the correct functioning of the previous procedure and the status of the VNFs that compose the

NSS can be verified via a user interface. Within the functioning system, a user can inspect the attributes of the instantiated objects. In particular, one can see which 5G services the VNFs offer and the relative Service Access Points (SAPs), what the state of those services is, and whether the UDM is registered in the NRF or not. The 5G services of the NFs can be invoked to verify the effective activation of the NSS.

3.2.4.2 Network Exposure Function

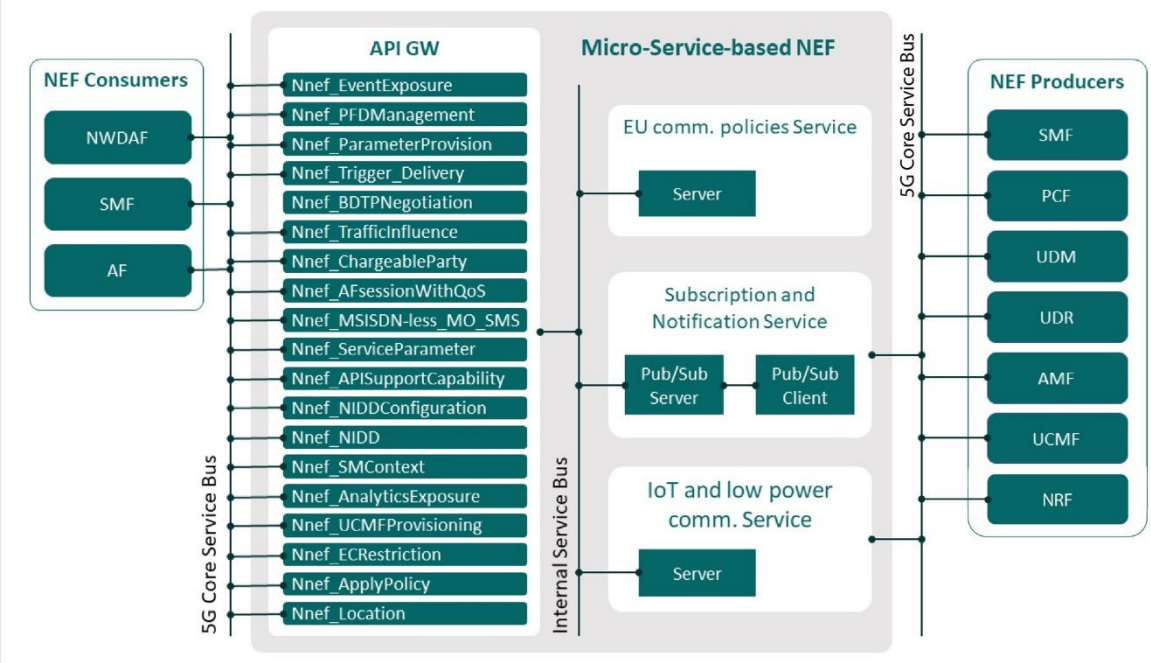

Figure 3-5 - Proposed microservice-based NEF design.

The Network Exposure Function (NEF) is located between external (AFs) and the 5GC. It is responsible for providing an access point for external applications to access the 5GC securely [TS23.501].

As shown in Figure 3-5, the envisioned microservice-based NEF encompasses a scalable and stateless API Gateway that processes RESTful requests to NEF and forwards them to the correct NEF microservice instance.

Behind it, there are three microservices, that are:

- The *UE communication policies* service.
- The *Subscription and notification* service.
- The *IoT and low-power communication* service.

The first one deals with UE policy request and configuration, e.g., QoS, traffic influence. The second one implements the publish/subscribe operations intended to monitor the network

and the UEs. Finally, the third one is responsible for operations towards IoT and low-power devices, e.g., Non-IP Data Delivery (NIDD), Background Data Transfer Policy (BDTP).

Each of the three microservices consumes the RESTful APIs exposed by the 5GC NF producers through the 5GC service bus. By having this decomposition, both a logical separation and self-compartmentalization are achieved, with greater benefits towards reliability, security, and performance.

For validation and benchmarking purposes, two sets of tests were performed to assess the performance of an implementation of such a NEF. These tests consisted in sending a high number of requests from a demo AF and a demo PCF through NEF. The setup was deployed on a testbed with the following characteristics:

Table 1 - Setup for NEF validation.

NF	Components	Instances	Resources	Extra configurations
NEF	NEF (3GPP Compliant)	1	1 vCPU (Intel Core Processor (Skylake) @ 2.40 GHz), 1024MB of RAM	Additional security layer disabled, collection of monitoring metrics enabled.
PCF	BSF, PCF and NRF (3GPP Compliant)	1	1 vCPU (Intel Core Processor (Skylake) @ 2.40 GHz), 1024MB of RAM	No extra settings.
AF	Demo AF (3GPP Compliant)	1 to 3	1 vCPU (Intel Core Processor (Skylake) @ 2.40 GHz), 1024MB of RAM	Each instance is running up to 100 AF processes in parallel to further increase the load of NEF.

The first set of tests focused only in validating that requests from AF to NF (in this case PCF) through NEF were successful. A single instance of the AF was used, and all requests reached PCF via NEF accomplishing the validation of NEF its basic functionality. The ensuing tests aimed to assess NEF its performance when a high number of AFs place multiple requests simultaneously. Running for 15 minutes plus warming up and cooling periods, the testes were split in three different stages, as depicted by Table 2.

Table 2 - Performance tests stages.

Stage	AF Instances	AF Processes Running	Average Requests/second
1	1 (with 100 AF processes running)	100	96,8
2	2 (each with 100 AF processes running)	100 per Instance	193,6
3	3 (each with 100 AF processes running)	100 per Instance	290,4

In Figure 3-6 it is possible to observe the CPU usage of NEF for the entire test duration. The results show that NEF handles the requests very well during the first stage. On the following stages, it is visible the struggle that NEF has handling the added requests, with CPU usage above 90 percent, and an average of 143,7 requests/second for the second stage and 147,8 requests/second for the last one. Concluding that there is almost no difference between stage 2 and 3 regarding requests handled.

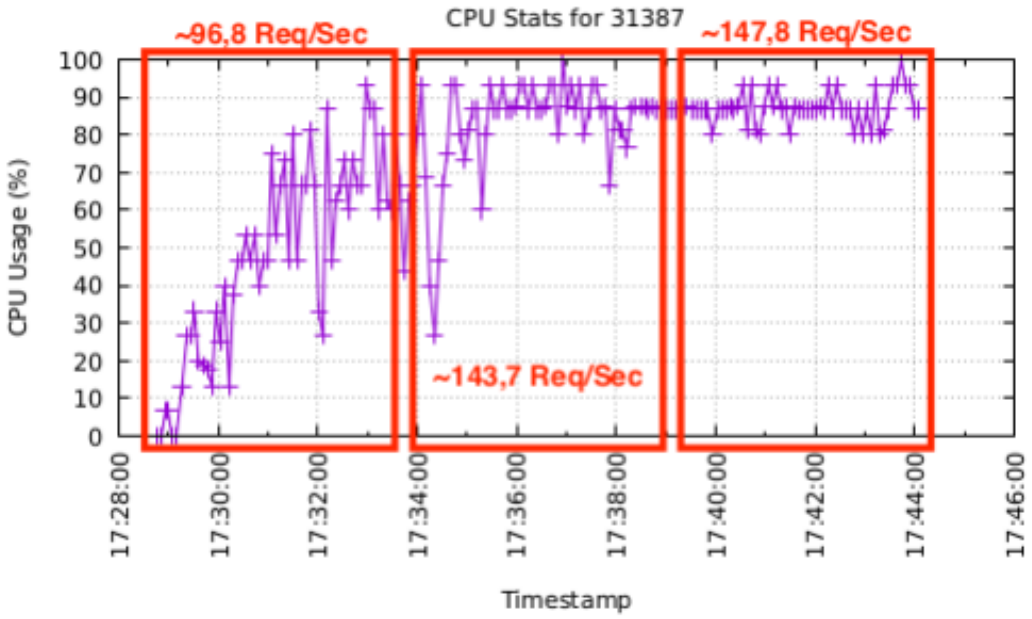

Figure 3-6 - CPU Usage for NEF performance tests.

During the performance tests, memory usage was another value measured. Observing Figure 3-7, there are no meaningful changes in memory usage along the different stages. The only noticeable difference the fact that RAM usage for NEF was changing quickly up and

down on stage 1. This allows us to conclude that NEF does not rely on high amounts of RAM, and it allocates almost the same amount for idle and high load status.

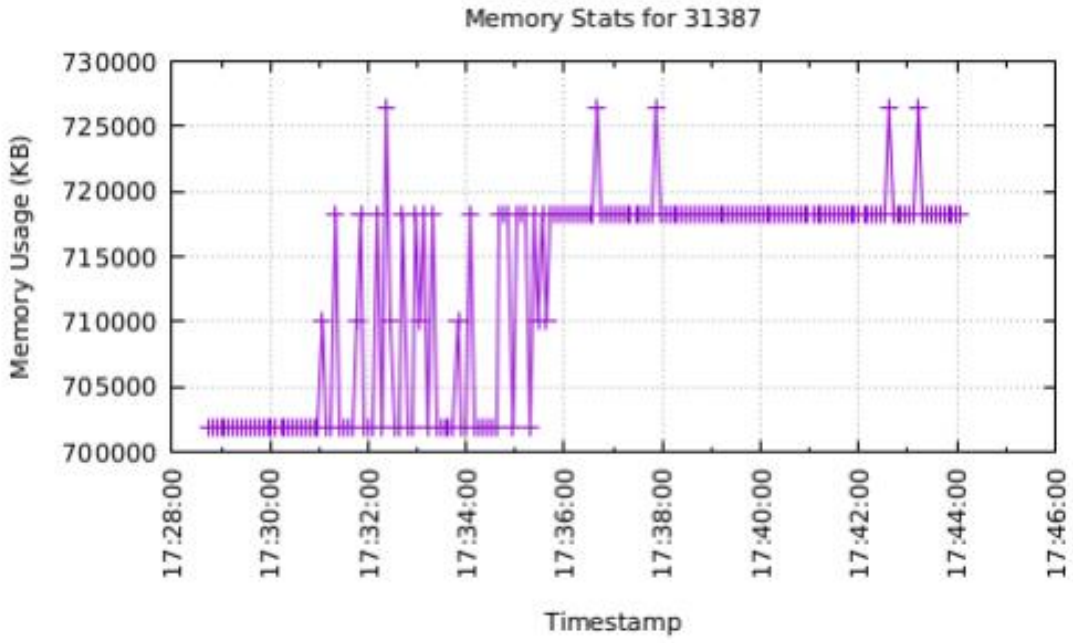

Figure 3-7 - Memory usage for NEF performance tests.

It is important to note that the testes presented are only for demonstration of the capabilities and validation of the NEF. Further testing has been done and the respective results will be reported in [D2.5].

The two images below contain an example of requests between the demo AF and PCF, made using REST and the specification from 3GPP, showing the full path in NEF's logs.

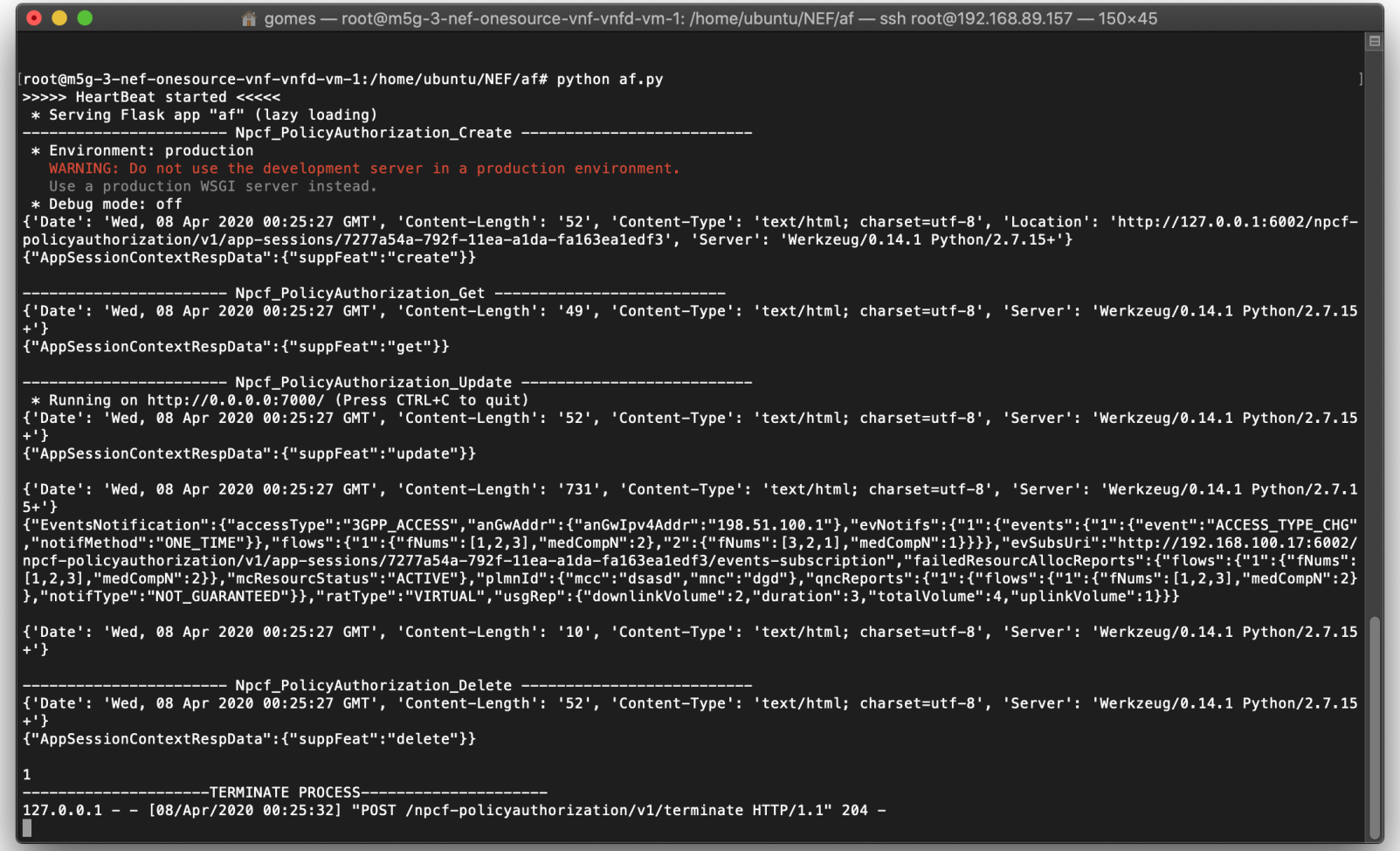

Figure 3-8- Requests from demo AF to demo PCF (via NEF).

Figure 3-9 - Request logs from demo AF to demo PCF (via NEF).

3.2.4.3 Cell Broadcast Centre Function

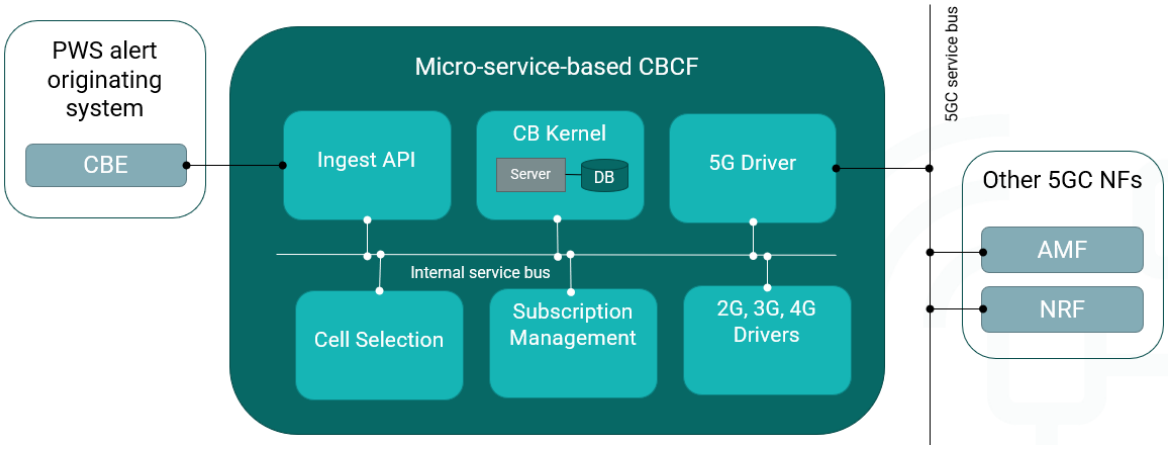

Figure 3-10 - Proposed microservice-based CBCF design.

The CBCF is an instantiation of an AF in the 5GC architecture and provides a public warning service in which government organizations can submit text messages to be broadcast to mobile devices in the alert area.

Figure 3-10 shows the architecture of a CBCF separated into microservices with only the most important microservices shown:

- Cell Broadcast (CB) Kernel,
- Cell Selection,
- 2G, 3G, 4G, 5G Drivers,
- Subscription Management.

Based on the criteria and approaches described in the previous sections, this separation of the CBCF into the proposed microservices has the advantage that depending on the deployment case, only specific microservices need to be adapted rather than the entire monolithic CBCF service, as explained hereafter.

The *Ingest API* of the CBCF receives public warning messages from a government system, i.e., a Cell Broadcast Entity (CBE). Often a government chooses a national profile of the Common Alerting Protocol (CAP) [Oas22], which implies that profiles differ for each country. The stateless Ingest API is a microservice that can easily be replaced to support another CAP-profile.

After validation by the Ingest API, the message is passed on the internal bus to the *CB Kernel* service, which requests the *Cell Selection* microservice to determine the cell sites that cover the alert area as indicated in the area component. *Drivers* supporting specific network generations (2G, 3G, 4G or 5G) are instantiated as microservices according to the operator's network deployment; they also support (static) scaling when the number of cells in any network becomes big.

Finally, a dedicated microservice manages *subscriptions* with the AMF and the NRF and renews subscriptions before they expire. This microservice cancels all subscriptions upon a graceful shutdown of the CBCF.

At the moment of writing this text, interoperability testing has been successfully performed with the three biggest network vendors. Tests with partners of the FUDGE-5G project are planned and will be reported in [D2.5].

Figure 3-11 shows the logging of an interoperability test between CBCF and an AMF. In particular:

- On line 5, a subscription request is sent by the CBCF, which is correctly acknowledged by the AMF, without causing errors.
- Lines 15 and 16 show sending a Write-Replace-Warning Request.
- The notification, which required the subscription, is shown on line 19.
- Lines 21 and 22 show a Stop Warning Request with the notification on line 25.

test_write_replace3 (1).pcap

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.154.34.10	10.239.32.70	TCP	74	40414 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1996382583 TSecr=0 WS=128
2	0.001185	10.239.32.70	10.154.34.10	TCP	74	80 → 40414 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=3592174816 TSecr=1996382583 WS=1
3	0.001236	10.154.34.10	10.239.32.70	TCP	66	40414 → 80 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=1996382585 TSecr=3592174816
4	0.001445	10.154.34.10	10.239.32.70	HTTP2	90	Magic
5	0.001577	10.154.34.10	10.239.32.70	HTTP2/…	373	SETTINGS[0], HEADERS[1]: POST /namf-comm/v1/non-ue-n2-messages/subscriptions, DATA[1], JavaScript Object Nota
6	0.002528	10.239.32.70	10.154.34.10	TCP	66	80 → 40414 [ACK] Seq=1 Ack=25 Win=65152 Len=0 TSval=3592174817 TSecr=1996382585
7	0.002716	10.239.32.70	10.154.34.10	TCP	66	80 → 40414 [ACK] Seq=1 Ack=332 Win=64896 Len=0 TSval=3592174817 TSecr=1996382585
8	0.002811	10.239.32.70	10.154.34.10	HTTP2	112	SETTINGS[0], WINDOW_UPDATE[0]
9	0.002836	10.154.34.10	10.239.32.70	TCP	66	40414 → 80 [ACK] Seq=332 Ack=47 Win=29312 Len=0 TSval=1996382586 TSecr=3592174817
10	0.002914	10.154.34.10	10.239.32.70	HTTP2	75	SETTINGS[0]
11	0.003586	10.239.32.70	10.154.34.10	HTTP2	75	SETTINGS[0]
12	0.003986	10.239.32.70	10.154.34.10	TCP	66	80 → 40414 [ACK] Seq=56 Ack=341 Win=64896 Len=0 TSval=3592174818 TSecr=1996382586
13	0.041555	10.239.32.70	10.154.34.10	HTTP2/…	328	HEADERS[1]: 201 Created, DATA[1], JavaScript Object Notation (application/json)
14	0.041584	10.154.34.10	10.239.32.70	TCP	66	40414 → 80 [ACK] Seq=341 Ack=318 Win=30336 Len=0 TSval=1996382625 TSecr=3592174818
15	0.042343	10.154.34.10	10.239.32.70	HTTP2	148	HEADERS[3]: POST /namf-comm/v1/non-ue-n2-messages/transfer
16	0.042532	10.154.34.10	10.239.32.70	HTTP2/…	770	DATA[3], WriteReplaceWarningRequest
17	0.043418	10.239.32.70	10.154.34.10	TCP	66	80 → 40414 [ACK] Seq=318 Ack=423 Win=64896 Len=0 TSval=3592174858 TSecr=1996382626
18	0.043606	10.239.32.70	10.154.34.10	TCP	66	80 → 40414 [ACK] Seq=318 Ack=1127 Win=64256 Len=0 TSval=3592174858 TSecr=1996382626
19	0.064955	10.239.32.70	10.154.34.10	HTTP2/…	215	HEADERS[3]: 200 OK, DATA[3], JavaScript Object Notation (application/json)
20	0.105054	10.154.34.10	10.239.32.70	TCP	66	40414 → 80 [ACK] Seq=1127 Ack=467 Win=31360 Len=0 TSval=1996382688 TSecr=3592174879
21	0.169613	10.154.34.10	10.239.32.70	HTTP2	113	HEADERS[5]: POST /namf-comm/v1/non-ue-n2-messages/transfer
22	0.169749	10.154.34.10	10.239.32.70	HTTP2/…	656	DATA[5], PWSCancelRequest
23	0.170744	10.239.32.70	10.154.34.10	TCP	66	80 → 40414 [ACK] Seq=467 Ack=1174 Win=64256 Len=0 TSval=3592174985 TSecr=1996382753
24	0.170844	10.239.32.70	10.154.34.10	TCP	66	80 → 40414 [ACK] Seq=467 Ack=1764 Win=64128 Len=0 TSval=3592174985 TSecr=1996382753
25	0.191644	10.239.32.70	10.154.34.10	HTTP2/…	212	HEADERS[5]: 200 OK, DATA[5], JavaScript Object Notation (application/json)
26	0.191675	10.154.34.10	10.239.32.70	TCP	66	40414 → 80 [ACK] Seq=1764 Ack=613 Win=32512 Len=0 TSval=1996382775 TSecr=3592175006
27	3.250973	10.154.34.10	10.239.32.70	HTTP2	147	HEADERS[7]: DELETE /namf-comm/v1/non-ue-n2-messages/subscriptions/9412e8c5-471a-4af8-977b-1e99c81ea3a6
28	3.272485	10.239.32.70	10.154.34.10	HTTP2	106	HEADERS[7]: 204 No Content
29	3.272540	10.154.34.10	10.239.32.70	TCP	66	40414 → 80 [ACK] Seq=1845 Ack=653 Win=32512 Len=0 TSval=1996385856 TSecr=3592178087
30	3.282630	10.154.34.10	10.239.32.70	TCP	66	40414 → 80 [FIN, ACK] Seq=1845 Ack=653 Win=32512 Len=0 TSval=1996385866 TSecr=3592178087
31	3.283766	10.239.32.70	10.154.34.10	TCP	66	80 → 40414 [FIN, ACK] Seq=653 Ack=1846 Win=64128 Len=0 TSval=3592178098 TSecr=1996385866
32	3.283824	10.154.34.10	10.239.32.70	TCP	66	40414 → 80 [ACK] Seq=1846 Ack=654 Win=32512 Len=0 TSval=1996385867 TSecr=3592178098

Figure 3-11 - Logging of CBCF-AMF interoperability test.

3.2.4.4 Authentication Server Function

The AUSF works as the front-end for the UDM to execute the authentication in the 5GC [TS29.509]. Depending on the subscription information provided by the UDM, the AUSF initiates one of the authentication procedures supported in 5G systems, 5G Authentication and Key Agreement (5G-AKA) or Extensible Authentication Protocol AKA' (EAP-AKA'). It is responsible for generating master keys that are used by the AMF to derive subsequent keys for the authentication procedure.

The AUSF interacts with the AMF (as a service consumer), the UDM (both as a service consumer and producer), and with the NRF (as a service producer) for registering and subscribing for services. For the UE registration, AMF sends authentication request message to AUSF. The AUSF talks to the UDM and obtains UE authentication information. For steering of UE, the UDM sends steering information to the AUSF to receive security contexts from AUSF and protect Steering of Roaming (SoR).

Based on the services provided, a microservice-based architecture can be proposed for the AUSF, where different AUSF instances may be instantiated within the 5GC pertaining to different services. There are three types of AUSF services specified by 3GPP:

- *Authentication* – responsible for authenticating the UE and generating master keys in the procedure for the consumer.
- *Steering of Roaming Protection* – permits the consumer to protect the steering information of UE from being tampered in a Visited PLMN (VPLMN).
- *UE Parameter update data protection* – responsible for permitting the consumer to protect the UE parameters update data from being tampered in a VPLMN.

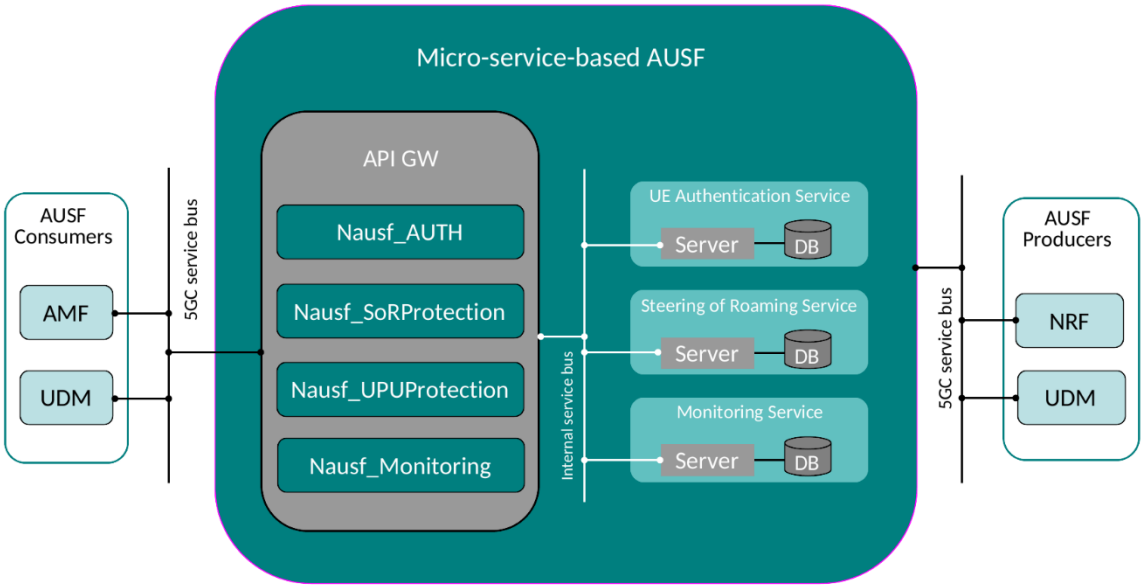

Figure 3-12 - Proposed microservice-based AUSF design.

The consumer can access each of the services by the API GW (as shown in Figure 3-12) that exposes the functionalities of AUSF through the SBIs and ensures security to the underlying services.

To realise the microservice-based AUSF architecture in FHG's Open5GCore testbed, AUSF instances are registered in NRF with the services they provide along with the NF profile. The consumers that need to access a particular AUSF service can subscribe to the NRF with the service name. As soon as the AUSF instance is registered in NRF, it sends the information of the instance to the consumer to be used in the 5G procedures. Otherwise, the consumer can request NRF (*getNFInstance*) for the details of the AUSF instance to be used for a particular service. For the authentication service the process is shown in Figure 3-13.

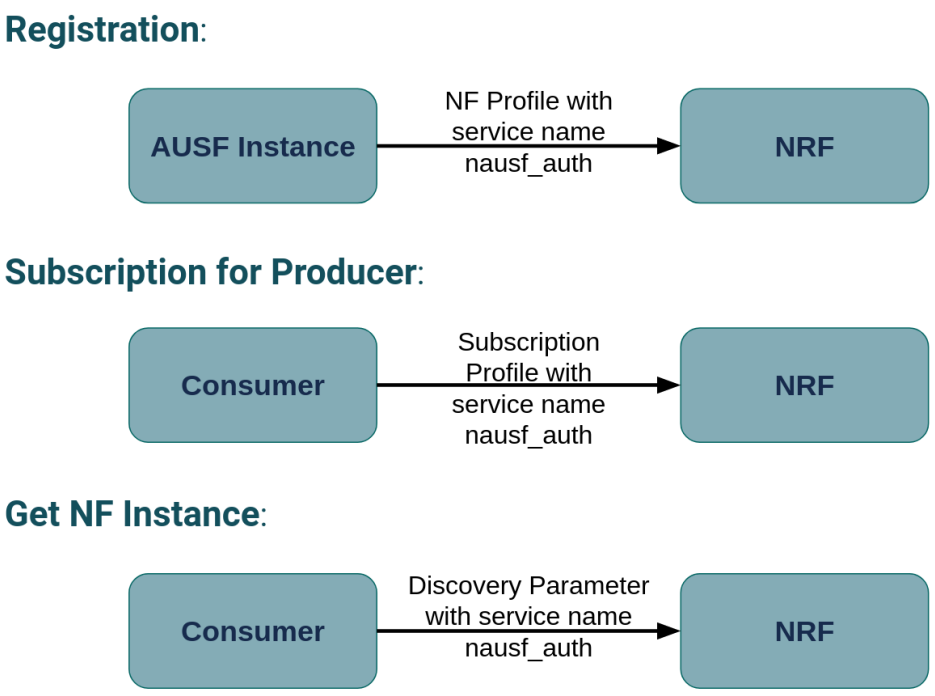

Figure 3-13 - Registration, subscription and getNFInstance functions for microservices.

This architecture also can be extended to instantiate multiple AUSF instances for the same service to adhere to scalability and improve reliability in the 5G system. The monitoring service shown in Figure 3-12 is responsible to collect the metrics that are pushed by the other AUSF services and send them to a monitoring tool (e.g., Prometheus [Prometheus]). The metrics such as the number of registration requests received by AUSF, the authentication type, percentage of success, and failures, can help in deciding the state of the system, when to instantiate a second AUSF instance for the same service and in load balancing. This will also help to have a stable system and improve performance.

3.2.4.5 Policy Control Function

The PCF in 5G differs from the Policy and Charging Rules Function (PCRF) in 4G in the functionality and interaction with other network functions. PCRF builds the Policy and Charging Control (PCC) rules based on operator policies stored in own database, UE subscription profile retrieved from the Subscriber Profile Repository (SPR), or requests coming from external AFs that allocate dedicated bearers or other policies to the data plane. As a result, the PCRF interacts with the PGW through Gx interface for setting the policy rules and receive information concerning user data traffic (e.g., charging).

PCF has several consumers to receive information (e.g., AMF, SMF, AF, Charing Function – CHF, UDR, NEF, Network Data Analytics Function – NWDAF). The UPF is not a direct PCF consumer. UPF instead will interact with the SMF. The PCF will provide the PCC Rules to the SMF that set the session-related parameters e.g. QoS Rules for the UE, QoS Profiles for the gNB, and will provide the SDF (Service Data Flow) descriptions to the UPF. The charging-related functionality previously in the PCRF now in the PCF this functionality has been moved to the 5G Charging Function (CHF) that is a PCF consumer.

Table 3 - Services produced by the PCF.

Service Name	Operation Semantics	Example Consumer
Npcf_AMPolicyControl	Request/Response/ Subscribe/Notify	AMF
Npcf_Policy Authorization	Request/Response/ Subscribe/Notify	AF, NEF, NWDAF
Npcf_SMPolicyControl	Request/Response/ Subscribe/Notify	SMF
Npcf_BDTPolicyControl	Request/Response	NEF
Npcf_UEPolicyControl	Request/Response/ Subscribe/Notify	AMF, V-PCF
Npcf_EventExposure	Subscribe/Notify	NEF, NWDAF

The PCF requires to interact with the AMF for checking the polices for the UE during the session establishment. A predefined PCC rule is by default configured in the SMF. However, a PCC rule can be activated/deactivated by the PCF, SMF shall decide what information has to be provided to the UPF to enforce the rule based on where the traffic detection filters (i.e., SDFs or application detection filter), traffic steering policy information and the policies used for the traffic handling in the UPF. PCF will request the UE profile from the UDR/UDM and will set the policies to the SMF. Finally, the SMF will communicate the SDF to the UPF through the N4 interface, dedicated to exchanges between the control plane and the user plane.

PCF can be divided into microservices. PCF serves charging and policy control purposes. It is not mandatory to implement all services. Splitting PCF design into microservices has an advantage when scaling traffic volume for services. CMC's PCF includes (cf. Figure 3-14):

- PCC Rule service,
- Charging Policy Service,
- Subscriber and Monitoring service.

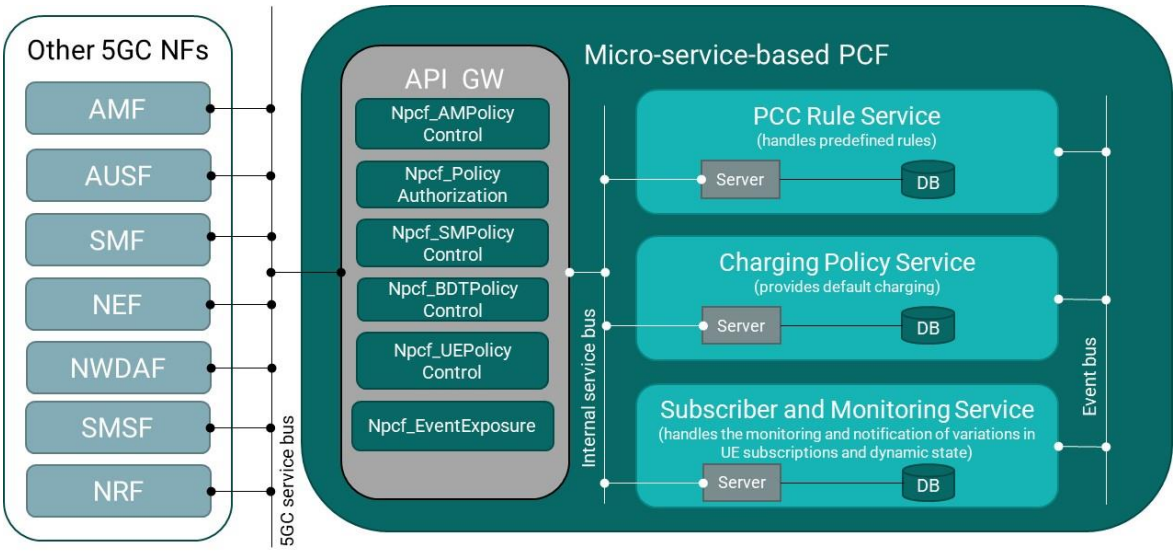

Figure 3-14 - Proposed microservice-based PCF design.

In particular, the PCC Rule Service implements the following functionalities:

- Policy control request trigger that defines the event(s) that shall cause a re-request of PCC rules for the Protocol Data Unit (PDU) Session.
- Authorized QoS per bearer (UE-initiated IP-Connectivity Access Network – IP-CAN – bearer activation/modification), defines the authorised QoS for the IP-CAN bearer (QoS Class Identifier – QCI, Guaranteed Bit Rate – GBR, Maximum Bit Rate – MBR).
- Authorized MBR per QCI (network-initiated IP-CAN bearer activation/modification), defines the authorised MBR per QCI.
- Revalidation time limit defines the time period within which the SMF shall perform a PCC rules request.
- Presence Reporting Area (PRA) Identifier(s), defines the PRAs to monitor for the UE with respect to entering/leaving.
- List(s) of PRA elements, defines the elements of the PRAs.
- Default Network-Based IP Flow Mobility (NBIFOM) access, the access to be used for all traffic that does not match any existing Routing Rule.

- IP Index, provided to SMF to assist in determining the IP Address allocation method (e.g., which IP pool to assign from) when a PDU Session requires an IP address – as defined in clause 5.8.2.2.1 of [TS23.501].
- Redundant PDU Session, indicates whether the PDU Session is a redundant PDU Session.
- Explicitly signalled QoS Characteristics defines a dynamically assigned 5G QoS Identifier (5QI) value (from the non-standardized value range) and the associated 5G QoS characteristics as defined in clause 5.7.3 of [TS23.501].
- Reflective QoS Timer, defines the lifetime of a UE derived QoS rule belonging to the PDU Session.
- Authorized Session-AMBR defines the Aggregate Maximum Bit Rate (AMBR) for the Non-GBR QoS Flows of the PDU Session.
- Authorized default 5QI/ARP defines the default 5QI and Allocation and Retention Priority (ARP) of the QoS Flow associated with the default QoS rule.
- Time Condition defines the time at which the corresponding Subsequent Authorized Session-AMBR or Subsequent Authorized default 5QI/ARP shall be applied.
- Subsequent Authorized Session-AMBR defines the AMBR for the Non-GBR QoS Flows of the PDU Session when the Time Condition is reached.
- Subsequent Authorized default 5QI/ARP defines the default 5QI and ARP when the Time Condition is reached.

The Charging Policy Service has following functionalities:

- Charging information, defines the containing CHF address and optionally the associated CHF instance ID and CHF set ID.
- Default charging method defines the default charging method for the PDU Session.
- PDU Session with offline charging only indicates that the “online” charging method is never used for PCC rules in the PDU Session.

Finally, the Subscriber and Monitoring Service has following functionalities:

- Usage Monitoring Control related information defines the information that is required to enable user plane monitoring of resources for individual applications/services, groups of applications/services, for a PDU Session.
- The PCF uses the monitoring key to group services that share a common allowed usage.
- Volume threshold defines the traffic volume value after which the SMF shall report usage to the PCF for this monitoring key.
- Time threshold defines the resource time usage after which the SMF shall report usage to the PCF.
- Monitoring time defines the time at which the SMF shall reapply the Volume and/or Time Threshold.

- Subsequent Volume threshold defines the traffic volume value after which the SMF shall report usage to the PCF for this Monitoring key for the period after the Monitoring time.
- Subsequent Time threshold defines resource time usage after which the SMF shall report usage to the PCF for this Monitoring key for the period after the Monitoring time.
- Inactivity Detection Time defines the period of time after which the time measurement shall stop, if no packets are received.
- Port number for which Port Management Information Container is provided.
- Port Management Information Container includes Ethernet port management information.
- Bridge Management Information Container includes Bridge management information.

3.2.4.6 Session Management Function and User Plane Function for Service Routing on the User Plane

This section presents a special decomposition example for a new SMF functionality which integrates with any existing 5GC due to the modular and flexible principles SBA brings, i.e., the usage of SBIs and the removal of any restrictions on what 5GC component is permitted to call another one. The design hereafter is based on the integration of Name-Based Routing (NBR) on the user plane, as described in [D1.2], and works in an over-the-top fashion of a 5G system in the sense that it uses an already established PDU session to send control-plane information to the SMF via the UPF. As the proposed solution brings SMF and UPF functionality, this section describes both NFs combined.

As illustrated in Figure 3-15, possible consumers for this SMF functionality, realised as a dedicated service, are the AF and other SMF services. The SMF_NBR then comes with an Ingest API offering three distinct set of primitives for:

- Policy Control of routing HTTP-based traffic on the user plane between UEs and DNs.
- Topology management to retrieve the topology NBR-enabled UPFs form.
- UPF property retrieval, allowing more details to obtain from all NBR-enabled UPFs around port identifiers, NBR capabilities (e.g., NBR version or Open vSwitch version) or SDN capabilities (e.g., OpenFlow or P4).

The SMF_NBR service is then composed of Path Computation Element (PCE), which performs basic topology management and rendezvous (registering and matching publishers and subscribers) tasks of NBR-enabled UPFs. A stateful database allows the two PCE components to operate in a stateless fashion.

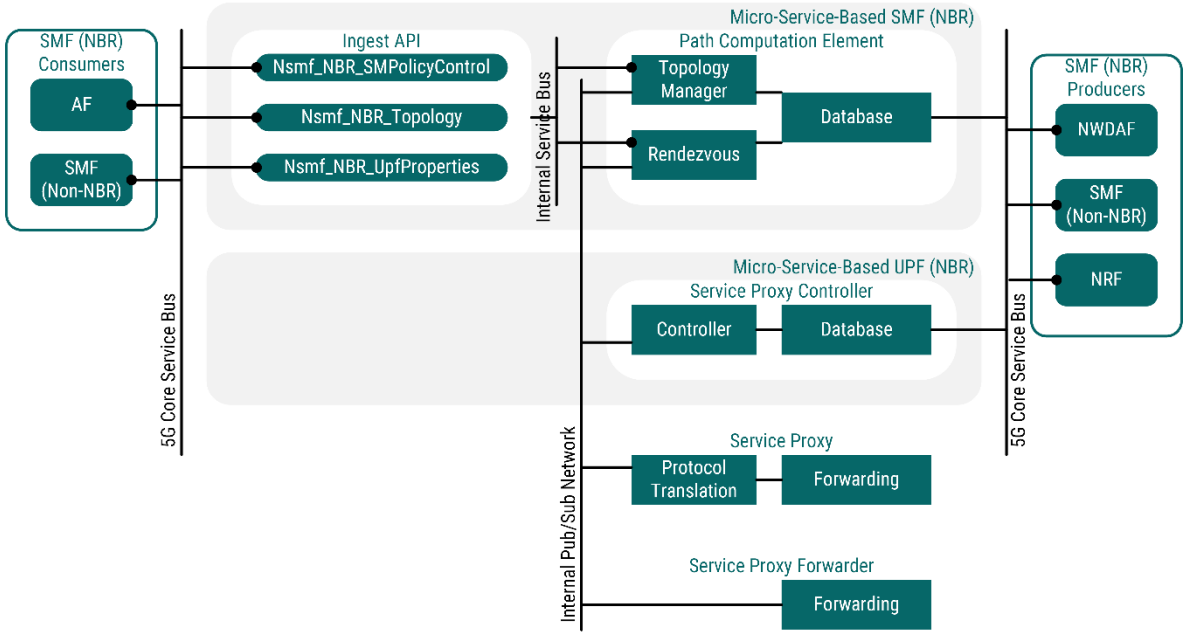

Figure 3-15 - Proposed microservice-based SMF and UPF for name-based routing integration on the user plane.

The UPF itself is decomposed into a microservice, Service Proxy Controller, that controls all UPF instances, Service Proxies (SPs) and Service Proxy Forwarders (SPFs), that do actual packet handling on the user plane.

As both SP and SPF cannot be realised in a microservice, they are not depicted under the "Micro-Service-Based UPF" box in the figure above. The communication between the PCE, SPC, SP and SPF is realised via a dedicated pub/sub protocol of NBR that operates on top of Layer 2 (802.3).

To the right of the figure, both SMF_NBR and UPF_NBR interfacing with NWDAF, SMF (non-NBR) and NRF via the 5GC's service bus.

3.3 Enhanced Service-Based Architecture in 5G and Beyond

The design patterns described in Section 3.2 demonstrate the ability to turn a monolithic software into a set of independent components. These components then use a service bus to communicate among each other, where the service bus brings the required communication methods for the stateless implementation of a component to retrieve information, without the need to figure out from which specific endpoint to retrieve it from, e.g., publish-subscribe.

What becomes apparent from these examples is the differentiation between the 5GC service bus and NF's internal service bus. While the 5GC service bus follows the SBI

specification in 3GPP (i.e., HTTP with JSON payload), the internal service bus may consist of a vendor-specific realisation (e.g., Kafka).

We recall that all 5GC service bus communication can be expected to be handled by the SCP, if the 5GC is operating under the so-called Model C or D [TS23.501], as opposed to Model A and B which refer to a direct communication where no SCP is present.

Once 5GC NFs are deployed as microservices, the importance of the SCP for the realisation of a 5GC service is likely to increase. Also, in the current design, the NRF is always queried to retrieve an IP address or FQDN of producers. Furthermore, when assessing the communication behaviour on the Internet, it must be concluded that the behaviour of explicitly addressing the SCP and then querying the NRF with a follow-up Domain Name System (DNS) query if an FQDN has been provided creates a triangular routing relationship among Consumer, SCP, NRF, and producer. When comparing this to how clients access services on the Internet, services are addressed directly (e.g., ieee.org). To remove the triangular routing relationship and to follow the semantics of how services are addressed on the internet, a new Model is proposed, Model E.

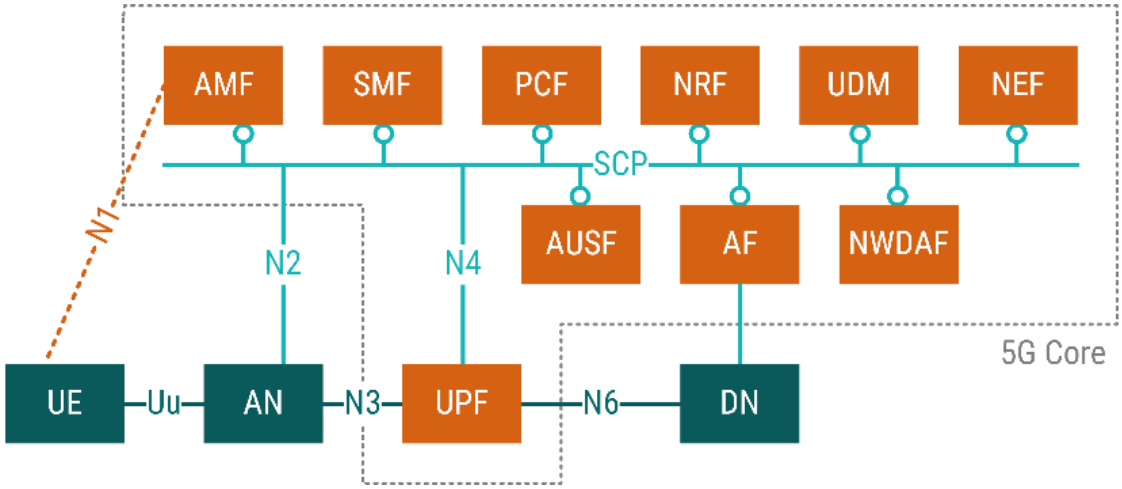

Figure 3-16 - Beyond-release-17 system architecture.

In this context, Figure 3-16 illustrates the envisaged system architecture for a beyond-5G system, where the SCP is the service bus across all examples in Section 3.2.4 and is not addressable anymore. The final description of the envisaged FUDGE-5G evolved System Architecture can be found in [D1.3]. Furthermore, the non-SBI-enabled interfaces carrying signalling traffic into a 5GC, that are, the N2 and N4, are also routed via the SCP. Even though these interfaces do not utilise HTTP, any SCP should be able to route standard IP traffic. Furthermore, it must be ensured that access networks and UPF can resolve FQDNs of AMF and SMF, respectively.

With the proposed architecture, when assessing the design patterns put into practice in the previous section, the differentiation of the service bus for inter-NF communication and the service bus for intra-NF communication may be revised. One key argument for 3GPP not to further decompose NFs and standardise their SBIs is to leave vendors the ability to differentiate themselves from their competitors through their software. However, it can be argued that if service routing (SCP) capabilities are made mandatory and combined with cloud-native 5GC orchestration capabilities, the internal service bus could leverage the SCP too. This of course requires all design patterns around naming of NFs, registration of these names against the SCP and potential isolation and QoS enforcement requirements to be properly defined and standardised, permitting multi-vendor deployments of a mobile telecommunication network.

4 Conclusions

To fully enjoy the benefits of numerous new use cases, vertical sectors request more and more the possibility to deploy and manage services that leverage 5G networks with high flexibility and increased automation, possibly exploiting tools and knowledge from the IT world on which enterprises and specialized service providers already have experience. The possibility of orchestrating vertical and networking services over cloud infrastructure answers these needs and calls for a new design of applications and network functions that fully embrace the paradigms of cloud-nativeness and microservices.

In this deliverable, we reported on FUDGE-5G's vision on such topics. We elaborated and discussed FUDGE-5G WP2's list of requirements, distinguishing characteristics, and design guidelines and constraints for orchestrable cloud-native functionalities and microservices. Further, given the specific focus of the project, this deliverable presented how the paradigm of microservices can be ported to the design of 5GC NFs, with pros and cons depending on the deployment scenarios and use cases. In particular, we highlighted how the definitions given by 3GPP of the 5GC NFs and their services are not sufficient to yield a straightforward and clear decomposition of such functions into actual microservices. As reasonable, if we think of the role and goals of a standardization body, a further design effort is left to the implementers of 5G systems. So, FUDGE-5G proposed in this document its view on the decomposition of a set of exemplary network functions, based on the reported criteria and approaches.

The presented methodologies and challenges were derived from current state-of-the-art methodologies and technologies, as well as from the direct experience of FUDGE-5G's partners on the field. Both theoretical and practical guidelines were proposed, with the aim of defining a framework that can be inherited by researchers, architects, and developers while designing microservice-based (network) functions. It is apparent that the described experiences strengthen the demand for a more open programmable and vendor-independent and telco-oriented cloud-native framework.

5 References

[Cen+22] M. Centenaro et al., "Prospects on the adoption of (micro)service-based architecture principles in 5G systems," preprint, under review for publication, 2022.

[Consul] Consul. https://www.consul.io/

[CNCF] CNCF, "Cloud native network function working group charter." [Online.] Available: https://github.com/cncf/cnf-wg/blob/main/charter.md

[D1.1] FUDGE-5G, "Technical blueprint for vertical use cases and validation framework," deliverable 1.1, Feb. 2021. [Online.] Available: https://fudge-5g.eu/download-file/493/zIrbJ2b9meNEGKKXwkEV

[D1.2] FUDGE-5G, "FUDGE-5G platform architecture components and interface," deliverable 1.2, Aug. 2021. [Online.] Available: https://fudge-5g.eu/download-file/455/UBFW5Rja2ByFBkQdYkQd

[D1.3] FUDGE-5G, "FUDGE-5G platform architecture final release," deliverable 1.3, July 2022.

[D2.1] FUDGE-5G, "FUDGE-5G technology components and platform – interim release," deliverable 2.1, Aug. 2021.

[D2.2] FUDGE-5G, "FUDGE-5G unified service based architecture platform," deliverable 2.2, July 2022.

[D2.5] FUDGE-5G, "FUDGE-5G technology components and platform final release," deliverable 2.5, Nov. 2022.

[dDWZ20] R. de Jesus Martins, A. Galante Dalla-Costa, J. A. Wickboldt, and L. Zambenedetti Granville, "SWEETEN: Automated network management provisioning for 5G microservices-based virtual network functions," 16th International Conference on Network and Service Management (CNSM), 2020.

[DWWN20] K. Du, X. Wen, L. Wang, and T.-T. Nguyen, "A cloud-native based access and mobility management function implementation in 5G core," 2020 IEEE 6th International Conference on Computer and Communications (ICCC), 2020.

[Eva04] E. Evans, "Domain-Driven Design," Addison-Wesley, 2004.

[KVM] Linux, Kernel Virtual Machine. https://www.linux-kvm.org/page/Main_Page

[IFA014] ETSI, "Network Functions Virtualisation (NFV) Release 3; Management and Orchestration; Network Service Templates Specification," ETSI GS NFV-IFA 014. [Online.] Available: https://docbox.etsi.org/isg/nfv/open/Publications_pdf/Specs-Reports/NFV-IFA%20014v4.2.1%20-%20GS%20-%20Network%20Service%20Templates%20Spec.pdf

[Ish+22] A. Ishaq et al., "Service-based management architecture for on-demand creation, configuration, and control of a network slice subnet," 2022 IEEE International Conference on Network Softwarization (NetSoft), 2022.

[MB20] R. W. Macarthy and J. M. Bass, "An empirical taxonomy of DevOps in practice," 2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 2020, pp. 221-228.

[Mic22] Microsoft Developer Division, .NET, and Visual Studio product teams, "Architecting Cloud Native .NET Applications for Azure," 2022. [Online.] Available: https://docs.microsoft.com/en-us/dotnet/architecture/cloud-native/

[NFV12] ETSI NFV ISG, "Network Functions Virtualisation An Introduction, Benefits, Enablers, Challenges & Call for Action," SDN and OpenFlow World Congress, Darmstadt, Germany, Oct. 2012. [Online.] Available: https://portal.etsi.org/NFV/NFV_White_Paper.pdf

[NGMN22] NGMN Alliance, "Experience on cloud native adoption," Final Deliverable (Approved), version 1.1, Jan. 2022. [Online.] Available: https://www.ngmn.org/wp-content/uploads/220128-Experience-on-Cloud-Native-Adoption-v1.1-Final.pdf

[Oas22] OASIS Open, "Common Alerting Protocol" v1.2, 2022. [Online.] Available: http://docs.oasis-open.org/emergency/cap/v1.2/CAP-v1.2-os.doc

[Ora20] Oracle, "Oracle Communications Cloud Native Core Solution," 2020. [Online.] Available: https://www.oracle.com/a/ocom/docs/industries/communications/5g-core-cloud-solution-br.pdf

[OraU20] Oracle, "Unified Data Manager (UDM) User's Guide," 2020. [Online.] Available: https://docs.oracle.com/communications/F25434_01/docs.10/UDM%20User%27s%20Guide/GUID-BB291E7E-22E2-4484-A902-45F7670A53A2.htm

[Prometheus] Prometheus. https://prometheus.io/

[Res18] E. Rescorla, "The Transport Layer Security (TLS) protocol version 1.3," IETF, 2018.

[Sam20] Samsung, "Cloud Native 5G Core," 2020. [Online.] Available: https://images.samsung.com/is/content/samsung/p5/global/business/networks/insights/white-paper/cloud-native-5g-core/Cloud-Native-5G-Core-Samsung-5G-Core-Volume-2.pdf

[Soe+18] T. Soenen et al., "Insights from SONATA: Implementing and integrating a microservice-based NFV service platform with a DevOps methodology," IEEE/IFIP Network Operations and Management Symposium (NOMS), 2018.

[SOL005] ETSI, "Network Functions Virtualisation (NFV) Release 3; Protocols and Data Models; RESTful protocols specification for the Os-Ma-nfvo Reference Point," ETSI GS NFV-SOL 005. [Online.] Available : https://www.etsi.org/deliver/etsi_gs/NFV-SOL/001_099/005/03.05.01_60/gs_NFV-SOL005v030501p.pdf

[TS23.501] 3GPP, “System architecture for the 5G System (5GS),” TS 23.501. [Online.] Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3144

[TS23.502] 3GPP, “Procedures for the 5G System (5GS),” TS 23.502. [Online.] Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3145

[TS28.541] 3GPP, “Management and orchestration; 5G Network Resource Model (NRM),” TS 28.541. [Online.] Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3400

[TS29.509] 3GPP, “Authentication Server Services,” TS 29.509. [Online.] Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3343

[Wig] A. Wiggins, “The 12 Factor App Methodology.” [Online.] Available: https://12factor.net