5G Vertical Application Enablers Implementation Challenges and Perspectives

Dimitrios Fragkos, George Makropoulos, Panteleimon Sarantos, Harilaos Koumaras
Institute of Informatics and telecommunications
NCSR Demokritos
Athens, Greece
{dfragkos, gmakropoulos, psarantos, koumaras}@iit.demokritos.gr

Anastasios-Stavros Charismiadis, Dimitris Tsolkas
Fogus Innovations & Services P.C.
Athens, Greece
{charismiadis,dtsolkas}@fogus.gr

***Abstract*—5G system aims to enable vertical industries utilizing network programmability to its full extension. 3rd Generation Partnership Project (3GPP) has already established the foundations to provide 5G Core's capabilities to third parties with Common API Framework (CAPIF) and Service Enabler Layer Architecture (SEAL). Vertical application developers communicate implicitly, through the Network Exposure Function (NEF), to unlock the available services. As more third parties move towards 5G ecosystem, their vertical applications need to be scalable, robust and more secure. Cloud native approach is the key enabler to fulfill those requirements and empower the inherent cloud-native characteristics of the 5GC. This paper investigates how the 5G architecture will support Vertical Application Enablers (VAEs) currently studied in 3GPP Rel.17, and proposes cloud native alternatives regarding VAEs implementation.**

Keywords— 5G, SEAL, NEF, VAE, programmability

I. INTRODUCTION

The telecom industry was upended in 2007 with the introduction of a new smartphone platform that was welcomed by customers and developers alike, following previous attempts to open cellphone handsets to third parties. Third-party developers were given easy-to-use, programmable Software Development Kits (SDKs) and Application Programming Interfaces (APIs) to create new applications for the smartphone, while unit sales soared thanks to effective marketing and praised user experience. The app-store for these applications launched the following year, and other smartphone manufacturers quickly followed, inviting programmers to create and innovate on their platforms as well. Superior user interface, faster data rates, and a business model that allowed programmers to reach potentially millions of customers, propelled the mobile app ecosystem to multi-billion-dollar status and accelerated the mobile operating system's openness.

5G promises even more disruption in app programmability, combining the untapped capacity of multiple simultaneous network features and promising a new generation of applications that deliver an unprecedented user experience. The business potential with 5G openness is high, considering that opening the OS of a mobile phone to external developers impacted the mobile market, then the potential by opening up a whole mobile network is enormous and is expected to disrupt the vertical industries.

5G Core (5GC) network is realizing this opportunity by securely exposing standard APIs. External third parties with permission, such as industries, platform developers, and designers, may use those standard APIs for building network-aware (5G-enabled) applications, which establish a bi-directional communication with the 5GC, retrieving network statistics, but also triggering specific policies and commands to the network.

The above-mentioned exposure capability is materialised through the Service Based Architecture (SBA), adopted by the 5GC network. Indeed, the 5GC control plane Network Functions (NFs) communicate through API-calls that define the related Service Based Interfaces (SBIs).

In this context, the Network Repository Function (NRF) allows other NFs to register their services, which can then be discovered by other NFs. This allows for a versatile implementation, in which each NF allows other approved NFs to access resources. In addition, the Network Exposure Function (NEF), provides a set of northbound APIs for exposing network data and receiving management commands. More precisely, NEF provides adaptors for connecting the southbound interfaces with the SBA to an exposure layer with northbound interfaces offered to third-party developers. In this way, NEF facilitates the safe disclosure of network resources to third parties, such as network slicing, edge computing, and machine learning, allowing for the monetization of network assets and business innovation. The functionality provided by NRF and NEF to third parties, enables programmability and adaptability of the 5G connectivity services, and creates a new ecosystem where third parties' developments bridge 5G exposed capabilities and service requirements/potentials from the vertical industries.

In this framework, 3rd Generation Partnership Project (3GPP) introduced the concept of Vertical Application Enablers (VAEs) in Rel. 16, enabling the efficient use and deployment of vertical apps over 3GPP systems. The specifications and the architecture are based on the notion of the VAE layer that interfaces with one or more Vertical apps. VAEs communicate via network-based interfaces that are well-defined and version-controlled. The focus of VAEs is to provide key capabilities, such as message distribution, service continuity, application resource management, dynamic group management and vertical app server APIs over the 5G system capabilities, as specified in [1].

From the VAE implementation perspective, cloud-native deployment procedures in network programmability allow the 5GC to deliver the benefits of cloud technology and be ultra-robust, secure, and scalable. Decomposing applications into smaller, manageable parts as loosely coupled stateless

services and stateful backing services is a basic concept of cloud-native implementation. This is typically achieved by the use of a microservice architecture, in which each part can be deployed, scaled, and upgraded independently. Thus, a rapid and low-cost implementation of new services can be achieved.

The rest of the paper is organized as follows: Section II briefly presents the 5G architectural functions that support the VAEs. Section III presents the cloud-native architectural alternatives of the VAEs, while Section IV performs a qualitative comparison of them and proposes the most suitable approach. Finally, Section V presents the future work and concludes the paper.

II. 5G Architecture Supporting VAEs

3GPP has already established the foundations to provide 5GC Network capabilities to vertical industries. The key concepts that has emerged are the Common API Framework (CAPIF) and the Service Enabler Architecture Layer (SEAL) together with NEF, as explained below.

A. CAPIF

1) CAPIF Architecture

CAPIF was introduced in 3GPP Rel. 15 [2], to enable a unified approach between 5GC's northbound APIs framework and vertical apps. The key concept is the standardization and development of the common supporting capabilities (e.g., authentication, service discovery, charging policies) that are applicable to northbound APIs in order to facilitate the development of vertical apps. CAPIF consists of the CAPIF Core Function (CCF), API Invokers and API provider domain which comprises API Exposing Function (AEF), API Publishing Function, (APF) and API Management Function (AMF). The architectural model adapted from [3] is presented in Fig. 1 and the functional entities are briefly described as follows:

- CCF, acts as an orchestrator that manages the interaction between service consumers (vertical apps) and service providers (e.g., NEF, SEAL). The main responsibilities of CCF are authentication of the API invoker, authorization of the API invoker to access the available service APIs, monitoring the service API invocations.
- API Invoker, represents the vertical app which consumes the service APIs utilizing CAPIF. API Invoker provides to the CCF the required information for authentication, discovers and then invokes the available service APIs.
- AEF, is responsible for the exposure of the service APIs. Assuming that API Invokers are authorized by the CCF, AEF validates the authorization and subsequently provides the direct communication entry points to the service APIs. AEF may also authorize API invokers and record the invocations in log files.
- APF, is responsible for the publication of the service APIs to CCF in order to enable the discovery capability to the API Invokers.
- AMF, supplies the API provider domain with administrative capabilities. Some of these capabilities include, auditing the service API invocation logs received from the CCF, on-boarding/off-boarding new API invokers and monitoring the status of the service APIs.

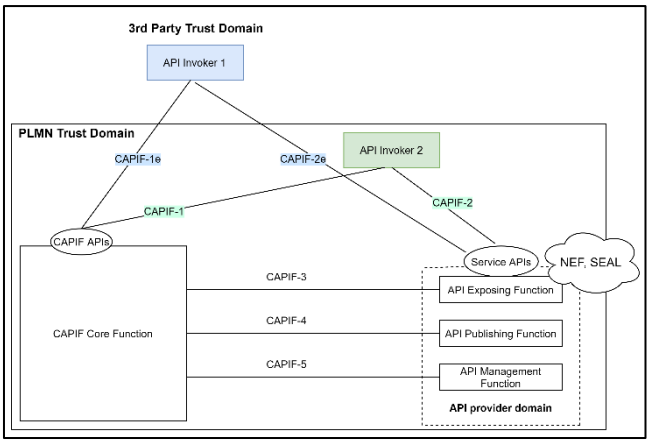

Fig. 1. Simplified CAPIF Architecture

3GPP considers two main architectural deployment models, centralized, when the CCF and API Provider domain functions are co-located, and distributed (Fig. 1), when CCF and API Provider domain functions are not co-located and they are interacting through CAPIF-3/4/5 interfaces. Therefore, multiple CCFs can be deployed in the same PLMN trust domain [3].

CAPIF is located within the PLMN operator network. Thus, there are two functional options for API Invokers; usually 3rd party applications, which have service agreement with PLMN operator, represent API invokers (i.e., API Invoker 1) but they may be co-located within the same PLMN trust domain (i.e., API Invoker 2). Whether third parties have business relationship with PLMN, they can provide their own service APIs to CCF through CAPIF-3e/4e/5e interfaces, but they need to act in accordance with the functionalities of API provider domain. In order to be compliant with the overall architecture (see Fig. 1), NEF and SEAL (i.e., SEAL server) support the CAPIF's API provider domain capabilities, as specified in [4] and [5].

2) CAPIF Services

The available CAPIF services and their respective APIs according to [3] are listed hereby. Services are divided into four categories, common, security, management and internal connectivity services:

Common Services

- Discover (CAPIF_Discover_Service_API): This service enables API Invokers to retrieve the available services that have been registered in CCF.
- Publish/Unpublish/Update (CAPIF_Publish_Service_API): APF consumes this service to publish/unpublish a service API to the CCF. The publication includes details about the specific service API. APF can also update already published services.

Retrieve (CAPIF_Publish_Service_API): APF requests from CCF information related with previous published services. When a publication occurs CAPIF registers all the related information in a repository (i.e., API registry).

Management Services

- Logging (CAPIF_Logging_API_Invocation): Upon invocations (i.e., from API Invokers), CCF may store valuable information such as API invoker's ID, IP

address, service API name etc. AEF utilizes this service to access the potential log files that have been stored in CCF.
- Auditing (CAPIF_Auditing): This service can be used to control CAPIF interactions with API Invokers (e.g., invocation events, onboarding events, authentication), which are stored in CCF. AMF initiates a request to fetch the respective log files.
- Charging: AEF can use this service to retrieve charging related information flows from the CCF.
- Monitoring events (CAPIF_Monitoring): Monitoring event service is used by AMF in order to get notified whether an event occurs in the CCF. Some of the events are the availability of service APIs (e.g., active, inactive), changes in service APIs (e.g., after an update), service API invocations, API invoker status (e.g., onboarded, offboarded) and performance related events (e.g., load conditions).

Security Services

- Authentication (CAPIF_Security / AEF_Security_API): An API Invoker can be authenticated from the CCF or the AEF. The former service enables invoker to initiate a direct request to the CCF. Otherwise, AEF authenticates an invoker with assistance from CCF. The authentication occurs prior or upon an invocation.
- Authorization (CAPIF_Security / AEF_Security_API): After authentication occurs, API Invokers initiate requests to retrieve service APIs. AEF checks whether the invoker is authorized to do so. If the AEF does not have the required information for authorization, AEF inquires CCF. Thus, AEF and CCF can invalidate invoker's configured authorization at any moment.
- Access control policy (CAPIF_Access_Control_Policy): This service enables AEF to obtain the configured policies to perform access control on the service API invocations.
- Registration of provider domain: This service enables AMF to register the API provider domain functions to CCF in order to be authorized and use CAPIF's functionalities
- On/off boarding (CAPIF_API_invoker_management): This service enables API Invokers as recognized users of the CAPIF. Invokers initiate the on-boarding process by sending a request to the CCF. If the enrolment information provided is valid, CCF on boards invokers and creates a new profile, which is sent back upon the response. API Invokers can also cancel their on-board status.

Internal connectivity

- CCF interconnection (CAPIF_Discover_Service_API / CAPIF_Publish_Service_API): This service enables the interconnection between multiple CAPIF providers. Each CAPIF provider has a CCF which utilizes publish and discover services in order to interchange its APIs.
- Topology hiding (CAPIF_Routing_Info): This service enables hiding the topology in the functional scenario where CAPIF includes PLMN trust domains, third party domains and API invokers access the service APIs from outside both the PLMN and third-party trust domains. In this case, API invokers access an AEF which acts as an entry point. Thus, the information for the entry AEF is shared with API Invoker in the discovery service. Then, subsequently, AEF resolves the actual destination address of the requested service API and forwards the initial request.

The abovementioned services need to fulfill the authentication and authorization prerequisites. The capabilities of the services are presented under the assumption that API provider domain functions (i.e., AEF, APF, AMF) and API Invokers are already authorized by the CCF and they are active. The detailed security aspects are specified by 3GPP in [6].

B. SEAL

1) SEAL Architecture

SEAL was introduced in Rel. 16 to support easier and faster development and deployment of vertical apps [7]. While the demand to develop vertical app standards for different types of industries was continuously increasing, it became obvious that many auxiliary services, such as location management, are needed across multiple vertical apps. As a result, capturing these commonly used auxiliary services and offering them to verticals as a common service layer, will benefit both verticals, allowing them to focus only on the core features and functionality of the vertical app, and operators, saving them from enormous efforts and time to develop the corresponding services for each vertical. The above concept became reality with the standardization of SEAL architecture [5]. SEAL architecture enables these common services to be consumed by vertical apps over 3GPP, CAPIF compliant, northbound APIs. SEAL architecture supports two functional models: on-network (i.e., SEAL-Uu), when the UE connects to the 3GPP network system to consume the service, and off-network (i.e., SEAL-PC5), when UEs connect to each other directly. The functional architecture is depicted in Fig. 2. For simplification, we consider only the on-network model.

The main functional entities of SEAL architecture are the following:

- Vertical Application Layer Client (VAL client): This entity provides the client-side functionalities of the corresponding vertical app (e.g., Vehicle to Everything (V2X) client).
- Vertical Application Layer Server (VAL server): This entity provides the server-side functionalities of the corresponding vertical app (e.g., V2X application server). If CAPIF is supported, VAL server acts as an AEF to provide the service APIs to the Vertical Application Server (VAS) or another VAE server. It can also act like an API Invoker to consume the service APIs, whether they provided by another VAL server.
- SEAL Client: This entity provides the client-side functionalities corresponding to a specific SEAL service (e.g., Location Management client)
- SEAL Server: This entity provides the server-side functionalities corresponding to a specific SEAL service (e.g., Location Management server). It can act as CAPIF's API exposing function.

Various deployment scenarios have been proposed in SEAL architecture, concerning the domain in which SEAL servers are deployed. According to [5] the SEAL servers can be deployed: a) in a single PLMN operator domain (centralized deployment), b) in multiple PLMN operator domains, as distributed function, with or without

interconnection between the SEAL servers, c) in the VAL service provider domain or d) in a separate SEAL provider domain.

2) SEAL Services

The following section describes the common set of SEAL services designed to be used by vertical apps.

- Location Management : Enables the vertical app to have access to network location information of its corresponding UEs. More specifically, this service can send reports on-demand to a VAS about the location of its UEs, subscribe the VAS so as to receive notification when location information of UEs changes, share UE location information etc.
- Group Management: Allows vertical apps to group UEs, thus enabling group management operations, such as enforce group policies, edit group configurations etc. The service also allows the vertical app to subscribe for and receive notifications when group information or status is modified.
- Configuration Management: Enables the vertical app to create and manage configuration on its UEs (provide initial configuration, edit configuration, notify server when configuration changes etc.)
- Identity Management: This service is responsible for the authentication and authorization procedures of a vertical app user.
- Key Management: Enables a vertical app to support secure transfer of data by providing and storing encryption keys.
- Network Resource Management: Allows a vertical app to manage network resources by managing (create, modify, delete) unicast and/or multicast bearers.

C. VAE Layer

VAE layer acts as a support layer between SEAL and a specific vertical application layer (e.g., V2X application client and server). VAE layer, by utilizing SEAL/NEF APIs and translating all the underlying network data to vertical application specific, enables the deployment of the actual vertical app. The functional model of VAE layer is depicted in Fig. 2. Similarly, to SEAL architecture, VAE supports both on-network and off-network model. Note that, both VAE Client and VAE Server are mutually-exclusive with VAL Client (SEAL) and VAL Server (SEAL), respectively.

The most important entities of the VAE architecture are the following:

- Vertical application specific client: Provides client-side functionalities corresponding to a specific vertical app (e.g., a platooning client in V2X use case).
- Vertical application specific server: Provides server-side functionalities corresponding to a specific vertical app (e.g., a platooning server in V2X use case). As mentioned, vertical app can act as an API invoker, if CAPIF is adapted. Specifically, vertical app's server side represents the invoker [1].
- VAE client: Provides the client-side support functions for a specific vertical app (e.g., deliver application messages to vertical app clients, receive monitoring reports from VAE server, provide location information to VAE server etc.)
- VAE server: Provides the server-side support functions for a specific vertical app (e.g., communicate with the underlying network, provide service discovery, support resource adaptation etc.).

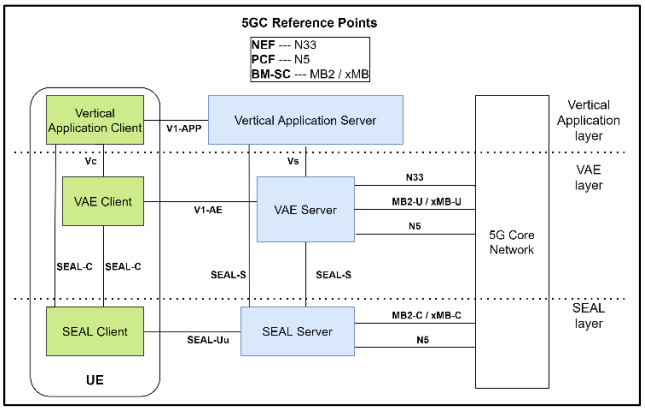

Fig. 2. VAE-SEAL Functional Model

According to [1], VAE server can be deployed either in a centralized manner, in which one VAE server supports one or more vertical app specific servers, or in distributed manner, in which one or more VAE servers (with or without interconnection between them) support one vertical application specific server. Furthermore, the VAE server can be deployed either in a PLMN operator domain or in a vertical service provider domain.

As mentioned, VAE layer, utilizes the capabilities of the underlying SEAL, thus it provides additional vertical specific capabilities to enable the applications. 3GPP has already specified the VAE architecture for Vehicle to Everything (V2X) services [1]. Procedures and information flows of services are already described and some of them are offered as APIs. Interestingly, some examples for V2X services are V2X UE registration, application-level, location tracking, file distribution, V2X application resource management etc. Work and studies are ongoing also for Factories of the Future (FOF) and for Unmanned Aerial Systems (UAS), in TR 23.745 and TS 23.255 respectively.

III. VAE Implementation Alternatives

To unlock the full potential of the 5G, the transition to a cloud native 5GC is an auspicious approach. However, the cloud-native deployment does not only refer to the 5GC, as it can have a direct impact to vertical specific applications as well, leveraging specific features in order to meet the requirements of the industry verticals. In the light of the above, this section describes three different cloud native approaches towards the deployment of the VAE, where the software is built upon microservices that can act independently.

A. Container-based Deployment

In the case that the VAE makes use of multiple processes, the deployment could be realized using a Container Platform. The architectural approach of the proposed deployment is depicted in Fig. 3. The specific implementation provides an advantage in terms of flexibility to the developer during the coding process by allowing the use of simple tools towards the "packing" of the VAE to an image. The implementation of a REST API for the callback updates from the 5GC is also

deemed necessary and the latter can also act as the endpoint to receive requests from the vertical app. To that end, a common library or framework can be utilized, so as to avoid coding the REST API backend from the beginning.

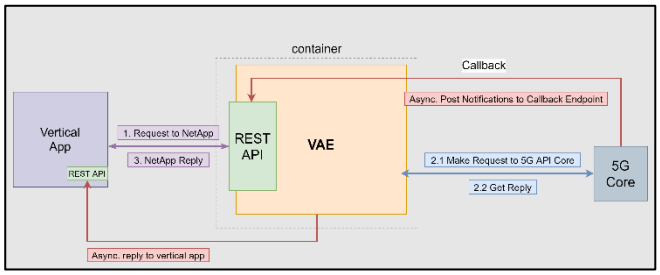

Fig. 3. Container-based VAE realisation

B. *Function as a Service (Faas) Deployment*

When the VAE acts as a micro service (i.e. exposing Southbound APIs to other services and applications), the utilization of the Function as a Service (FaaS) ecosystem allows the deployment of the VAE in a serverless environment [8]. Fig. 4 represents the reference architecture of the proposed FaaS approach. By adopting a serverless approach, the VAE automatically excludes the option to have a built-in API backend, thus the vertical app can use the REST API endpoints of the FaaS Platform to communicate with the VAE. The VAE can also configure "on the fly" endpoints on the FaaS API in order to receive asynchronous callbacks from the 5GC. An optional database can be used from the VAE to empower a stateful approach. By this means, the VAE will be enhanced in terms of efficiency and can be treated as a standalone extension of the vertical app. The developer of the vertical app has the liberty to add processes and increase the complexity of the system, by offloading additional functions of the vertical app to the VAE framework. Using a FaaS approach is a straightforward process to horizontally scale across multiple end-devices, either statically or dynamically. The term dynamically, refers to the case that the administrator of the FaaS platform can configure limits that will increase or decrease the replicas of a VAE ad-hoc, based on the number of requests (or any other resources).

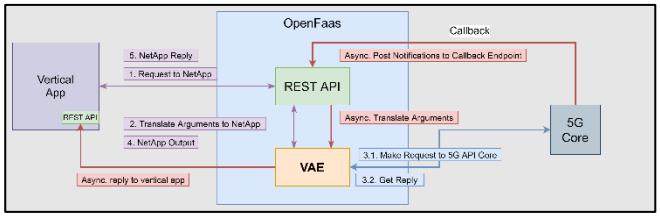

Fig. 4. FaaS-based VAE realisation

C. *Container Deployment with a Message BUS*

When the VAE acts as micro-services, an alternative approach beyond the containerised deployment, enhancing the overall architecture, is the utilization of a communication framework between the VAE and the vertical application/service. A suitable candidate technology-enabler is a common Message Bus channel using message queue service, that can be applied to handle all the asynchronous interactions between the actors of the ecosystem (i.e., vertical app, VAE, clients). The architecture is depicted in Fig. 5.

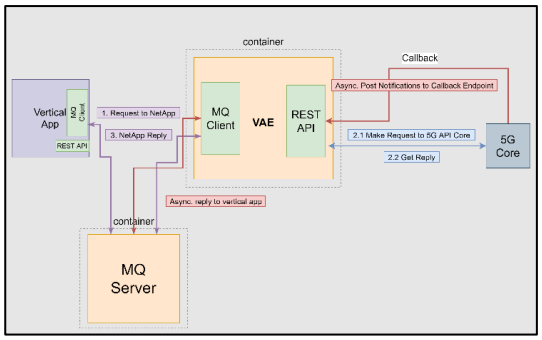

Fig. 5. Container with a message bus-VAE

The bridging of the vertical app with VAE can be achieved via the publish/subscribe mode that the protocol provides, acting as an alternative solution to the traditional client-server communication model with endpoints. In such case a Message Queuing Telemetry Transport (MQTT) broker acts as an intermediate for the Message Queuing (MQ) client sending messages and the subscriber (vertical app) who is receiving those messages. However, the VAE requires a REST API backend in order to support the communication with the 5GC. Additional services (e.g., databases) could also be supported by the proposed architecture, thus fulfilling the MQ requirements.

IV. Qualitative Comparison of the Different VAE Implementation Alternatives

With the aim to highlight the pros and cons and evaluate the efficiency of the proposed cloud-native implementations, this section provides a qualitive comparison towards a list of KPIs that were deemed suitable for all the aforementioned use cases. The defined KPIs, in order to cross-evaluate the three different approaches, include efficiency in terms of coding, scalability and agility, as well as communication and function decoupling.

A. *Qualitative Evaluation of VAE Implementation Alternatives*

1) *Container-based Deployment Qualitative Evaluation*

Due to the fact that all the libraries for the VAE implementation can be bundled within the selected image, deploying the VAE as a Container is a simple and effective approach. The dependencies between the components of the system can be isolated, allowing for the inclusion of several secondary applications, such as a database, allowing the provision of more complex VAEs and the support of advanced vertical services and applications. Since most container engines now operate across several platforms, the process requires a smaller set of programming skills, facilitating significantly the development efforts, as well as the technical expertise needed to deal with the programming task.

However, the underling container platform adds a noticeable overhead in terms of performance, but it can be discounted compared to the management benefits that provides. It is worth noting that the scalability process requires both the VAE, as well as any additional applications, such as databases, to conform with horizontal scaling. Furthermore an additional load balancer may be required, as the VAE implementation increases the overhead effect, thus affecting both management and performance capabilities.

2) *FaaS-based Deployment Qualitative Evaluation*

The FaaS model deployment allows the automatic and independent scaling in horizontal manner, thus representing an efficient implementation towards the service of an application. Moreover, it utilizes effectively system's resources, either on demand (dynamically) or with a predefined process. However, a drawback towards the specific deployment, is the fact that the developers have to establish the implementation of the VAE via the FaaS framework, based on the learning curve that the platform indicates. The management of the overall system is highly dependent on the maturity and the abstraction level of the FaaS platform. In order to support a production environment, the use of a FaaS platform, promoted by a wide community, is recommended. As far as the performance of the system is concerned, since a FaaS deployment uses an underling container platform, overhead is being added due to the provision of the Docker Engine. Most FaaS implementations bring up to the surface the cold start issue, which has an effect on certain types of applications, where latency is critical.

3) Container Deployment with a Message BUS Qualitative Evaluation

The Message Bus implementation aims to simplify the front-end interface by using an Message Queue (MQ) communication protocol. Adopting this approach can result in a more efficient adaptation and consumption of the request/replies by the vertical app. Most MQ communications are implemented in a publish/subscribe manner, thus various aspects of the vertical app can be subscribed or publish to the VAE. This method could enable the vertical app to expose the VAE API to the clients as well via direct communication. Moreover, the asynchronous communication enabled by the message queues, optimizes the data flows between the components, resulting in better performance of the system. With respect to the overall management of the architecture, the queues reduce the dependencies between the involved components leading to coding simplicity.

B. Cross Comparison Score and Technology Selection

Table I comprises the qualitive cross comparison KPIs for the proposed VAE implementations, according to the aforementioned qualitative analysis.

TABLE I. QUALITATIVE COMPARISON OF KPIS

Criterion/KPI	Implementation Approach		
	Isolated Container	***Container with FaaS platform***	***Containers with Message Bus***
Vertical Scaling	*	***	**
Horizontal Scaling	***	**	**
Technology Maturity	***	*	**
Orchestration Options	***	*	**
Code management	***	**	**
Performance	*	**	**
Data and control decoupling	**	**	***
SCORE	**76% (16/21)**	**62% (13/21)**	**71% (15/21)**

According to Table 1, the various KPIs are listed, considering the agility of the deployment option, as well as the scaling easiness, the technology maturity, but also the orchestration options, together with coding requirements and performance aspects, as well as data and control decoupling. The comparison shows that all alternatives achieve a satisfactory score. However, the isolated container approach seems to be more preferable, since its simplicity contributes to its easiness in terms of code management, scaling and orchestration options. Also technology maturing plays a significant role, especially for deployments within a production environment, such as a 5G network of a mobile operator. The additions of a message BUS, seems also to be a highly preferable solution, since it decouples the control plane from the data plane, allowing policies enforcement and prioritization in the message management and the executions of functions. Finally, the FaaS framework, also scores high, and it seems to be the rest preferable approach, mainly due to the extra complexity that it introduces, which affects also the performance and the scaling capabilities of the overall system, considering also its impact on the orchestration. Moreover, the FaaS framework may cause also a technology locked-in, which will affect the smooth evolution of the platform in case that the FaaS framework does not continue to fost and grow as expected.

V. CONCLUSIONS

This paper presented the SBA of the 5GC Network, and 3GPP's pioneering frameworks that enable data exposure to third parties through the northbound APIs. CAPIF, SEAL and VAE are undoubtedly the key enabling frameworks to unlock the capabilities of a programmable 5G network. Three cloud native architectural approaches for the communication between the network interfaces and the VAE were proposed and a qualitive comparison was performed in order to showcase the most efficient solution.

ACKNOWLEDGMENT

The work in this paper has been funded by the H2020/5G-PPP Research Projects EVOLVED-5G (Grant Agreement no. 101016608) and 5G!Drones (Grant Agreement no. 857031).

REFERENCES

[1] 3GPP TS 23.286, "Application layer support for Vehicle-to-Everything (V2X) services", Release 17, V17.1.0, April 2021

[2] N. D. Tangudu, N. Gupta, S. P. Shah, B. J. Pattan and S. Chitturi, "Common Framework for 5G Northbound APIs," *2020 IEEE 3rd 5G World Forum (5GWF)*, Bangalore, India, 2020, pp. 275-280, doi: 10.1109/5GWF49715.2020.9221161

[3] 3GPP TS 23.222, "Common API Framework for 3GPP Northbound APIs", Release 17, V17.4.0, April 2021

[4] 3GPP TS 23.501, "System architecture for the 5G System (5GS)", Release 17, V17.0.0, March 2021

[5] 3GPP TS 23.434, "Service Enabler Architecture Layer for Verticals (SEAL)", Release 17, V17.1.0, April 2021

[6] 3GPP TS 33.122, "Security aspects of Common API Framework (CAPIF) for 3GPP northbound APIs", Release 16, V16.3.0, July 2020

[7] S. P. Shah, B. J. Pattan, N. Gupta, N. D. Tangudu and S. Chitturi, "Service Enabler Layer for 5G Verticals," *2020 IEEE 3rd 5G World Forum (5GWF)*, Bangalore, India, 2020, pp. 269-274, doi: 10.1109/5GWF49715.2020.9221425

[8] E. V. Eyk, A. Iosup, S. Seif and M. Thömmes, "The SPEC cloud group's research vision on FaaS and serverless architectures", *Proceedings of the 2nd International Workshop on Serverless Computing (WoSC '17)*, New York, USA, December 2017, pp. 1-4, doi: 10.1145/3154847.31548