UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

TRABAJO FIN DE MÁSTER

Edge selection mechanisms and open source APIs in the context of 5G networks

Máster Universitario en Ingeniería de Telecomunicación

Author: Ricardo Serrano Gutiérrez

Advisor: David Moro Fernández
Lecturer: Jorge E. López de Vergara Méndez

February 2024

EDGE SELECTION MECHANISMS AND OPEN SOURCE APIS IN THE CONTEXT OF 5G NETWORKS

Ricardo Serrano Gutiérrez

ADVISOR: David Moro Fernández
LECTURER: Jorge E. López de Vergara Méndez

Department of Electronics and Communications Technology
Escuela Politécnica Superior
Universidad Autónoma de Madrid

February 2024

A mis padres y a mi hermano.

A mi tutor, David, por ser un ejemplo a seguir en lo profesional y a todos los que me han acompañado en mi etapa laboral.

A mi familia, a mis amigos y a todo aquel que sienta orgullo por todo lo que he conseguido.

Deo gratias

Summary

Abstract

In the context of 5G networks, Telco Edge emerges as a new paradigm in terms of low-latency cloud computing. The ability to place cloud computing nodes as close as possible to the internet access point, enabled by the architecture of a telecommunications operator, implies a significant improvement over traditional cloud computing architectures.

The integration of this type of platform with the traditional architecture of the telecommunications operator, particularly following the philosophy of open infrastructures proposed not only by 5G standards but also driven by industry forums such as GSMA, raises the need for this type of solution to be operable by non-specialized developers through the use of standard interfaces.

This project aims to fill the gap in the state of the art when it comes to managing the deployment and implementation functions of applications in Telco Edge. Specifically, it seeks to implement one of the fundamental interfaces for reducing latency in these specific architectures, which is the selection of the most optimal node.

Key words

5G networks, Telco Edge, Low-latency, Cloud computing, Telecommunications operator, open infrastructures, GSMA, standard interfaces, developers, deployment, implementation, applications, Edge site selection.

Resumen

En el contexto de las redes 5G, el concepto de *Telco Edge* emerge como un nuevo paradigma en la computación en la nube con baja latencia. La capacidad de colocar nodos de computación en la nube en las proximidades del punto de acceso a internet, facilitada por la arquitectura de un operador de telecomunicaciones, representa una mejora significativa en comparación con las arquitecturas tradicionales de computación en la nube.

La integración de esta plataforma con la arquitectura tradicional de un operador de telecomunicaciones, especialmente en línea con la filosofía de infraestructuras abiertas promovida no solo por los estándares de 5G, sino también respaldada por foros de la industria como GSMA, plantea la necesidad de que dicha solución sea operable por desarrolladores no especializados, mediante el uso de interfaces estándar.

El objetivo de este proyecto es abordar las limitaciones actuales en el estado del arte relacionadas con el manejo de las funciones de despliegue e implementación de aplicaciones en *Telco Edge*. Específicamente, se centra en implementar una de las interfaces fundamentales para reducir la latencia en estas arquitecturas: la selección del nodo más óptimo.

Este enfoque busca contribuir al desarrollo de soluciones más accesibles y eficientes, promoviendo la adopción de *Telco Edge* y facilitando su implementación incluso para aquellos desarrolladores que no son especializados en la materia.

Palabras Clave

Redes 5G, Telco Edge, baja latencia, computación en la nube, operador de telecomunicaciones, infraestructuras abiertas, GSMA, interfaces estándar, desarrolladores, implementación, aplicaciones, Selección de nodos de Edge.

Contents

List of Figures

List of Tables

Preamble

5G Architecture

The concepts described in this project are closely related to the architecture defined in the 5G standards. A non-specialized reader may find the understanding of these documents quite complex. Therefore, this section aims to give an overview of the 5G architecture to facilitate the reading and comprehension of the rest of the work. It is important to note that there are different operating planes within the 5G architecture, particularly in what is known as the Core of the network. The Core is divided into three distinct domains: the data Core (responsible for providing internet connectivity services to users), IMS Core (IP Multimedia Subsystem, in charge of providing voice and video services to users), and the Packet Core (responsible for packet routing, mobility management, and network functions like authentication and authorization). Additionally there is also the Radio Access Network consisting of the different radio link cells (in 5G named after gNodeB) to which the mobile devices attach themselves to transmit and receive information. For simplicity and consistency with the rest of the report, this section will primarily focus on the 5G data Core. Additionally, references will also be made to the 4G architecture due to the current context in which 5G networks are mostly not fully deployed.

The 5G data Core relies heavily on the concept of Network Functions (NF), which are modular software elements that perform specific tasks within the network. These functions can include but are not limited to, traffic routing, security, user mobility and session management. Understanding the role of NFs is crucial in comprehending the overall functionality and efficiency of the 5G data Core and by extension are also relevant to this project. The main NFs of the data Core are detailed as follows:

- **User Plane Function (UPF)**: The UPF deals with the actual data transfer in the user plane. It is responsible for packet routing and forwarding, as well as handling quality of service (QoS) for user data. This element is also part of the Packet Core and works a data gateway.
- **Access and Mobility Management Function (AMF)**: The AMF manages the mobility of users within the 5G network. It handles tasks such as registration, authentication, and connection setup for devices moving between cells.
- **Session Management Function (SMF)**: The SMF is responsible for session-related functions, including the establishment, modification, and termination of sessions. It ensures efficient use of network resources and manages the UPF the user shall connect among other functions.
- **Authentication Server Function (AUSF)**: The AUSF authenticates and authorizes users, ensuring secure access to the 5G network. It works in coordination with the subscriber's home network to verify user identity.
- **Unified Data Management (UDM)**: The UDM manages user-related data, including subscriber profiles, authentication credentials, and subscription information. Additionally it plays a crucial role in user authentication and authorization.

- **Unified Data Repository (UDR)**: Data base where user data is stored. It is mainly accessed by UDM.
- **Network Exposure Function (NEF)**: The NEF enables external applications and services to interact with the 5G network. It provides APIs for third-party developers, facilitating the creation of innovative applications and services.
- **Policy Control Function (PCF)**: The PCF controls and enforces policies related to network resource usage, QoS, and other parameters. It ensures that the network operates in accordance with defined policies and service agreements.
- **Service Communication Proxy (SCP)**: The SCP dynamically distributes signaling loads, ensuring balanced utilization of network resources, and actively manages overload situations to prevent disruptions. Additionally, the SCP optimizes signaling routing, directing messages through the most efficient paths to maintain reliable and timely communication
- **Network Slice Selection Function (NSSF)**: The NSSF plays a role in selecting and steering traffic to the appropriate network slices based on the requirements of the services and applications being used. Network slicing is a particular 5G feature based in Radio Resource Partitioning (4G) which allows to segment the traffic of one ore multiple users, but is not part of the scope of this project.

The 5G Core components, including the Service Management Function, User Plane Function, and others, work collaboratively to ensure efficient network operation. Key functions such as mobility management, session control, authentication, and policy enforcement contribute to a secure, high-performance 5G experience. The architecture's adaptability is highlighted by elements like the Network Exposure Function, allowing external applications to integrate seamlessly. This overview provides a basis for understanding the intricacies of 5G networks and their evolving role in delivering advanced services and connectivity. Figure 1 shows the different elements mentioned and the interfaces among them. This project will primary focus in the aforementioned NFs: UPF, SMF and NEF. [3]

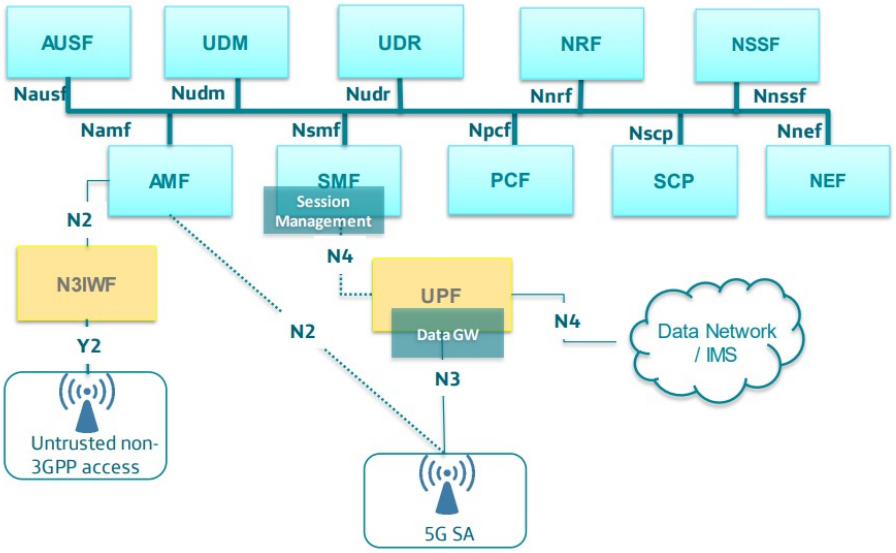

Figure 1: Standard 5G architecture

1 Introduction

1.1. Motivation

In the current times, technology companies are keen to exploit the vast computing potential of Cloud platforms. The software functionalities, operational procedures and terabytes of data that such companies handle nowadays are gradually posing a challenge that needs to be tackled in a decentralized manner. Cloud computing has emerged as a response to such demands. There is now no need for any company to spend large sums of money to purchase hardware for such tasks. Almost any company worldwide can now connect to such services and execute almost any type of software at a lower cost than if they were to construct their own infrastructure.

Nevertheless, only a few companies worldwide are offering these services due to the substantial investment required to provide them in an efficient way. Furthermore, the Cloud sites are usually situated in precise geographic locations, leading to potential latency issues. At this juncture, Telco Edge Cloud proves advantageous. Placing high-capacity compute nodes close to the endpoint of computational tasks leads to significant reductions in latency. Telecommunications operators thus have an opening to innovate their business and use their position as the gateway to the Internet to place these powerful computing nodes where the demand requires [4].

Anyhow, if there is a desire to minimise latency, it may be needed to reduce it to the lowest achievable level. It is therefore necessary to establish the different mechanisms and procedures that a user or a developer should follow in order to connect to the site that offers the lowest latency at any given moment. The establishment of these connection procedures to the best Edge site available in the geography of the telecommunications operator's architecture is highly synergistic with the new capabilities that 5G is beginning to expose. Since these procedures can be exposed to developers and users via simple, open source and standard Application Programming Interfaces (API), the 5G network can be used as a platform for the development and deployment of new functionalities this kind.

The primary motivation of this project is to explore the application deployment process in the Telco Edge Cloud and particularly, Edge selection techniques so to be able to design and implement a standard API that enables developers to access the inherent capabilities of a real telecommunications network. It is expected in this project to showcase the significant role of integrating Edge Cloud technology into telecommunications networks as a novel technological and business paradigm.

1.2. Objectives

The implementation of 5G networks represents a novel paradigm concerning the exposure of network capabilities through streamlined APIs. The disclosure of Edge Cloud service mechanisms is no exception in this context.

As previously outlined, a primary objective of the project is not only to comprehend and study the workings of Edge Cloud service deployment mechanisms but also to provide a standard API that exposes these functionalities through the 5G network to developers and users who require them. To achieve these final objectives, the following goals will be pursued:

- Study and comprehension of the benefits of Cloud computing, its usage by telecommunications operators and the analysis of Telco Edge working architectures.
- Investigation and presentation of the different standardization organisms in the 5G networks paradigms such as 3GPP (3rd Generation Paternship Project) GSMA (Global System for Mobile Communications Association) and CAMARA as well as industry projects such as Open Gateway
- Study and comprehension of the various components that constitute a 5G infrastructure and their potential impact on the exposure of capabilities via APIs. Also, recognition of key elements within the 5G architecture for exposing the aforementioned capabilities and designing API flows around these elements.
- An overview of the security protocols and procedures used in a typical API consumption environment.
- Define and validate a Telco Edge API for application onboarding and lifecycle management and Telco Edge API for Edge selection

Following all these sub-goals, the expected deliverables are as follows:

- Python code implementing a standard Edge Selection API. This API would have been deployed and tested on the architecture of a real telecommunications operator.
- API definition in Yaml format regarding application onboarding and lifecycle management to be used as a contribution in standardization organisms such CAMARA.
- A Python-based OAuth server as well as SSL/TLS certificates in order to comply with essential security features.
- Configuration and test scripts for the API in the context of a specific telecommunications operator architecture.

1.3. Methodology and working plan

The project alternates successively between study and analysis phases, design phases, implementation phases and documentation phases. Therefore, the work has been divided into 9 task blocks, most of which include all of these development phases.

The first three blocks correspond to tasks of study and understanding of the state of the art. Blocks 4 and 5 correspond to analysis and design stages. Blocks 6 and 7 correspond to implementation and deployment phases. Finally block 8 corresponds to the testing stage which in some cases occurs in parallel with the implementation tasks for obvious reasons. Block 9 corresponds to a documentation phase. The approximate time taken for each of the tasks is shown in Figure 1.1 as a Gantt diagram.

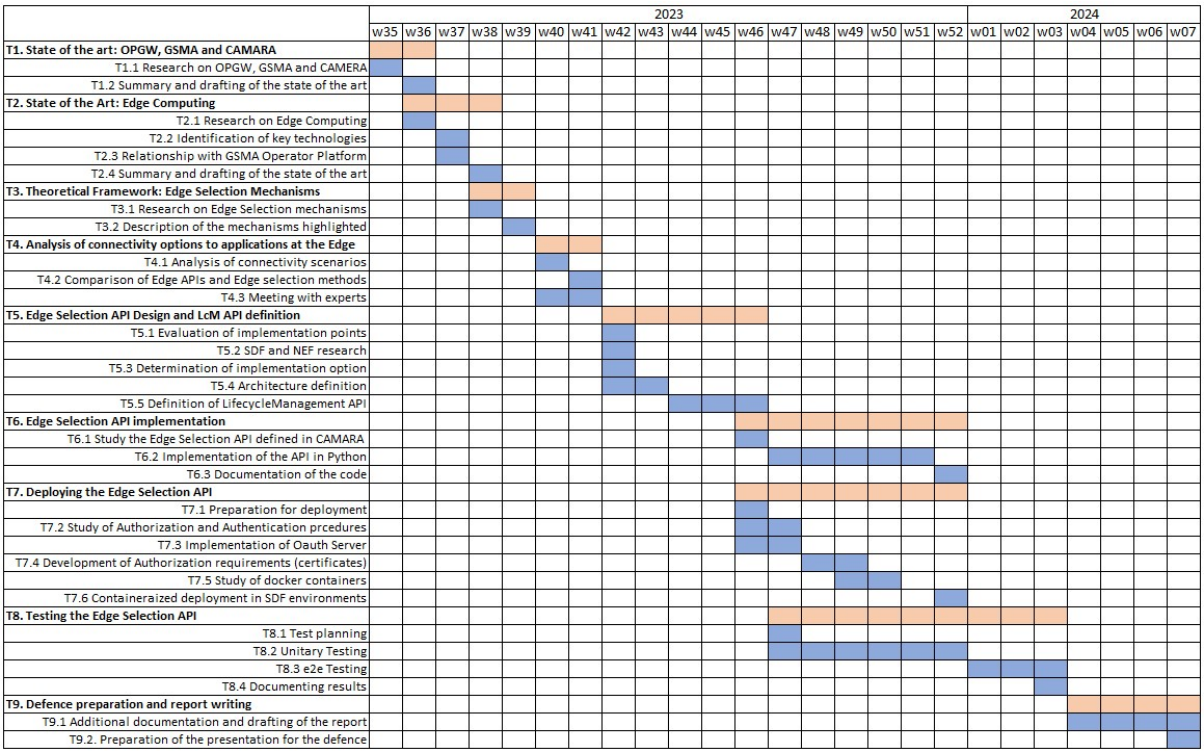

Figure 1.1: Gantt diagram of the project

1.4. Report structure

The report will be structured in the following chapters:

- **Chapter 1: Introduction.** General concepts of the project will be discussed in this chapter. It includes the motivation, the methodology and the main objectives of the project.

- **Chapter 2: State of the art.** This chapter explores the current landscape of Edge computing within the context of 5G, covering Cloud Edge, standardization organisms, and various use cases. Also provides the conceptual foundation necessary for understanding the implementation of a standard Edge Selection API within the 5G architecture and the current gaps in the state of the art.

- **Chapter 3: Analysis.** The problem to be solved will be presented and the challenges this problem enhances will be described. This chapter will relate the main intents that a developer would need in order to deploy a service (containers, Virtual Machines, software etc.) in the Telco Edge Cloud.

- **Chapter 4: Design.** A Yaml format API definition for the complete working flow will be delivered. As the implementation chapter will focus in development of the Edge Selection API implementation, insights regarding the options of deployment and the edge selection mechanisms will also be given in this chapter. The authorization and authentication procedures for the API consumption will be introduced in this chapter.

- **Chapter 5: Implementation.** The development insights, security protocols, deployment process and architecture of the Edge Selection API will be detailed in this chapter.

- **Chapter 6: Testing.** This chapter delves into the testing phase of the Edge Selection API, covering the testing architecture, segmented testing of systems (unitary testing), end-to-end testing, and concluding with findings and conclusions.
- **Chapter 7: Conclusions and future work.** The final chapter summarizes project conclusions and outlines potential future directions for research and development in Edge computing and telecommunications.

2

State of the Art

2.1. Introduction

One of the main advantages that 5G networks bring, as mentioned in the introductory chapter, is the ability to expose their capabilities through standard interfaces. This includes not only the information that a given telecommunications operator may have about its own users but also the development of functional architectures that bring value to the technology industry. This approach of course applies to the development of the Telco Edge. Throughout this chapter there will be presented the different use cases and technologies that are expected to be developed in the new context of 5G networks.

The development of these technologies should occur within the context of industry alliances. This promotes the standardisation of interfaces and provides an advantage to developers who wish to utilise them, as well as resulting in significant cost savings for the companies involved. In this sense, industry trades such as GSMA, which represent the interests of mobile network operators worldwide, play a crucial role in standardisation. Under the umbrella of the GSMA comes the Open Gateway project, which aims to develop the standard interfaces mentioned above within the CAMARA forum, an open source project within Linux Foundation to define, develop and test the 5G APIs.

As far as Edge Cloud is concerned, CAMARA has a project group for the development and implementation of this family of APIs. This is the fundamental segment on which this project will focus. In short, this chapter will focus on showing the state of the industry and technology as well as presenting the gaps that exist in the state of the art and where this project will make a contribution.

2.2. Telco Edge Cloud

The market for Cloud computing is expanding rapidly as it enables companies and individuals to save on hardware infrastructure spending. Virtualization advancements have simplified the deployment of programs, services, and applications in a Cloud environment. There are numerous businesses that provide Cloud computing services. Any program that can run on CPUs or GPUs with the architectures available in the Cloud site can be deployed in this type of environment

and accessed from anywhere in the world.

One disadvantage of this infrastructure type is that it is typically deployed by a limited number of sites, usually one or two per continent. This can result in challenges related to manageable data bandwidth and, more significantly, latency. As a response to this issues, Edge Cloud solutions are emerging as computing nodes deployed in more geographically localised points. These solutions do not require a telecommunications operator and are generally less performant than the typical Cloud solutions for cost reasons. They are typically deployed in the order of several sites across a country.

Anyhow, in typical Edge Cloud deployments, a challenging issue to overcome is the potential change in latency values for the user of the Cloud service as they move around the territory. This change can occur due to the difficulties to adapt the user necessities to the current deployment of different Edge sites and can negatively impact the user experience.

Telco Edge Cloud refers to a Edge Cloud solution, particularly, "reduced" (compared to other Cloud deployments) computing clusters deployed within the telecommunications operator's network [5]. This not only offers lower latencies as the average Edge Cloud solutions, but also proves advantageous in terms of mobility. Telco Edge Cloud not only efficiently manages user mobility between Edge sites but also enables developers to adjust the connectivity parameters required for their applications, thanks to technologies such as Slicing. This is possible due to the integration of Edge Cloud clusters with the telecommunications operator's architecture. As a mobile network operator manages user mobility throughout the network, for instance changing radio nodes, the new 5G features provide a new way to adapt connectivity to user requirements.

This level of versatility allows for a wide range of applications. The evolution towards holographic calls and other volumetric video scenarios will require high computational load and reduced latencies. Additionally, artificial intelligence models can be deployed on the Edge and accessed from almost any part of a territory, such as in autonomous driving scenarios. Furthermore, mobile terminals can benefit from improved battery usage as applications can be deployed directly on the Edge without the need for the terminal to run too complex computational processes.

Given that one of the main advantages discussed with respect to the Telco Edge Cloud is the fact that the user is always offered the most optimal site despite the user's own mobility, it is necessary to explore the technical rationale behind this capability. The need to develop Edge selection mechanisms arises.

2.3. Standardization Organisms and Industry Projects

Standardization of technology is a crucial issue in the telecommunications industry for innovation. There are multiple standards that must be met in this project with regards to the industry as a whole. However, when it comes to standardization organizations for Telco Edge Cloud, such as GSMA and CAMARA, as well as industry projects like Open Gateway, they play a crucial role in shaping the landscape. The collaborative efforts of these organizations define the current state of the art and pave the way for future advancements in 5G networks and Telco Edge solutions. By working together, they contribute to the establishment of a robust framework that benefits developers, operators, and end-users alike.

2.3.1. GSMA

The Global System for Mobile Communications Association (GSMA) plays a pivotal role in the standardization and development of mobile communication technologies globally. As

an influential trade organization, GSMA represents the interests of mobile network operators worldwide. Within the context of 5G networks and Telco Edge, GSMA is actively involved in shaping industry standards and promoting collaboration among industry actors. GSMA's initiatives contribute to the establishment of common interfaces and frameworks that facilitate interoperability and seamless integration within the telecommunications ecosystem.

The GSMA's Operator Platform is a central initiative, providing mobile network operators with a comprehensive toolkit to empower them. The Operator Platform simplifies 5G adoption by providing standardized interfaces, promoting interoperability, and ensuring a consistent user experience. In the field of Telco Edge, this platform is becoming central, providing operators with the tools to optimise networks, deploy advanced services and facilitate the integration of Telco Edge solutions through standardised APIs and protocols. GSMA's commitment to evolving the Operator Platform ensures that it remains a key enabler for operators to unlock the full potential of 5G networks and Telco Edge technologies[6].

2.3.2. CAMARA

CAMARA, an open-source project under the Linux Foundation, focuses on defining, developing, and testing 5G APIs. Within CAMARA, the Telco Edge Cloud is a significant area of exploration. The project serves as a collaborative platform where industry players can engage in discussions, contribute to the development of APIs, and address challenges related to the deployment of Telco Edge solutions. CAMARA's efforts, particularly in the Telco Edge domain, aim to standardize interfaces and promote a unified approach to enable the widespread adoption of 5G technologies.

Another important subgroup in relation to the harmonization of CAMARA APIs is the Commonalities Subgroup. Its focus is on identifying and promoting shared elements and best practices across diverse 5G APIs. This includes establishing guidelines for consistent naming conventions, data formats, and communication protocols to enhance interoperability and reduce fragmentation. The philosophy of the subgroup is to align all the 5G APIs defined in CAMARA and provide them with a common structure. Additionally, the subgroup aims to define which technical features should be considered or excluded within the overall APIs.

2.3.3. Open Gateway

The Open Gateway project arises under the umbrella of GSMA. It is a significant initiative within the CAMARA forum. Its objective is to develop standard interfaces, including Telco Edge interfaces, by focusing on open standards. The project aims to enhance the capabilities of 5G networks, making them more accessible and adaptable to diverse use cases. The project contributes to the development of APIs that are essential for realizing the full potential of Telco Edge solutions through collaborative efforts. The involvement of industry stakeholders in the Open Gateway project signifies a commitment to driving innovation and ensuring the seamless integration of Telco Edge technologies.

The Open Gateway project is essential to the evolution of the Operator Platform. It contributes to the development of standardized interfaces that enable the effective integration of Telco Edge solutions. The collaboration between the Open Gateway project and the Operator Platform within the GSMA ecosystem highlights a joint effort to promote innovation, establish industry-wide standards, and facilitate the widespread adoption of 5G technologies and Telco Edge solutions by mobile network operators.

2.4. Gaps in the state of the art and contributions

This project focuses on the development of Telco Edge Cloud APIs. It has already been introduced that the organisation responsible for the discussion, development and definition of APIs is the CAMARA project. In this forum and in relation to the Telco Edge Cloud, three families have been developed for the Edge exposure: Traffic Influence APIs, Simple Edge Selection APIs and Resource Management APIs. At the time this project was initiated, the status of the CAMARA standard for Edge Cloud APIs was as follows:

2.4.1. Traffic Influence APIs

The Traffic Influence API is an integral part of the OPAG-CAMARA system and aims to optimise traffic routing in terms of latency within specific geographical areas. It facilitates the establishment of the best routing paths between user devices (e.g. smartphones) and optimal Edge Cloud site instances. This optimisation is crucial for scenarios where a service is offered by both Cloud and edge instances, allowing the Traffic Influence API to influence traffic routing towards edge instances to improve latency, possibly in combination with other CAMARA APIs such as Quality on Demand [7].

Key components include the management of Traffic Influence resources, each of which specifies the desired intent for routing optimisation. Developers can use the API to create, query, modify and delete these resources, with asynchronous notifications providing updates on resource status. The API's flexibility allows developers to specify geographic locations, application instances and other parameters to tailor routing optimisations, making it a valuable tool for efficient traffic management in Telco Edge environments. [8]

The Traffic Influence API is already defined in the CAMARA repository. However, it is still under discussion due to the complexity of its implementation, as the operators' networks are not yet ready for deployment. Full 5G integration is required, as well as specific 5G functionalities such as Slicing and NEF (Network Exposure Function) deployments. Slicing allows for the creation of virtual network segments to meet various service requirements, while NEF enhances the adaptability and efficiency of the Traffic Influence API by exposing network capabilities. Features that are critical for the deployment of this API.

2.4.2. Edge Selection APIs

The Edge Selection API retrieves the most optimal Edge site name for the user's device from which the request was made. This information is typically parsed from the device and sent to the Application Provider endpoint for URL creation. The resulting URL will connect the user to the Telco Edge site where the application instance is running and that is closest to the user's device.

The Edge Selection API differs from the previous Traffic Influence API in that it is limited to identifying the most optimal Edge site based on the current network configuration. The API returns the name of this Edge site. Its function is to check the UPF to which the user is connected and establish a relationship between that UPF and the closest Edge site. On its part, the Traffic Influence API can determine if the user is connected to the UPF with the lowest latency and, if requested, switch the user's connection to a more optimal UPF. Thus, Traffic Influence API is able to make changes in the operator network.

This API is also already standardised in the CAMARA repository. The implementation of this API is simpler than the previous one and, as mentioned before, it is necessary that each CAMARA API is implemented by at least two operators in order to reach the highest CAMARA

version API and to validate the definition so that it can be used by developers and other partners in the industry. In this sense, this project will contribute to the implementation of this API to achieve these goals.

2.4.3. Resource and Lifecycle Management APIs

The application lifecycle management, the handling of resources in the Telco Edge platforms and in general the management of software instances and hardware in the Telco Edge need to be done through standard interfaces and therefore need to be developed in CAMARA. At the time this project started, there are some proposals from different actors in the repository regarding this topic, but the proposals are quite different from each other. It is necessary to harmonise the current definitions before any implementation of this API family.

The aim of this project is to establish a standard definition for lifecycle management APIs in order to resolve the current issues faced by the CAMARA subgroup. The aim is to establish a standard API for implementation, thereby enabling the versioning flow of CAMARA to continue. This has been hindered by conflicts between the proposed APIs.

2.5. Conclusions

In summary, this chapter outlines the current landscape of 5G networks and Telco Edge, emphasizing the crucial role of standard interfaces and collaborative industry efforts. The Telco Edge Cloud field has been explored, highlighting its advantages in terms of lower latencies, efficient mobility management, and adaptability to diverse applications.

Although there have been advancements, significant gaps have been identified in the state of the art within the Telco Edge domain. Key Telco Edge Cloud APIs, such as Traffic Influence APIs face deployment challenges due to ongoing discussions and operators' networks being unprepared. The integration of full 5G and specific functionalities such as Slicing and NEF deployments adds complexity to their implementation. Therefore, the development of the Traffic Influence API is outside the scope of this project.

The Edge Selection API, although simpler, requires validation through widespread adoption by at least two operators, implementation and deployment of Edge Selection API poses a challenge that the project aims to address. Furthermore, there are discrepancies in proposals for standardization that need to be harmonized in Resource and Lifecycle Management APIs. This project aims to address this gap in the current state of the art. Definition of an Application Lifecycle Management API for Telco Edge.

3 Analysis

3.1. Introduction

Following the principles of the Open Gateway initiative, it is crucial that the deployment of services in the Telco Edge Cloud is easily manageable by any developer through standard APIs. In this sense, it is necessary to expose things like the selection of the most optimal Edge site, as well as the deployment of applications (VMs and container services among others) on the Edge sites and the management of the tasks and sessions occurring on these Edge sites through these sort of interfaces.

3.2. Challenges

Telco Edge Cloud technology is, for the moment, just beginning to blossom. As stated above, it is important that this technology is born *API-fied* to facilitate access to as many users as possible. In this way, the key points that the proposed API should be able to cover are described in this section. Figure 3.1 illustrates the various flow sections of the API working together.

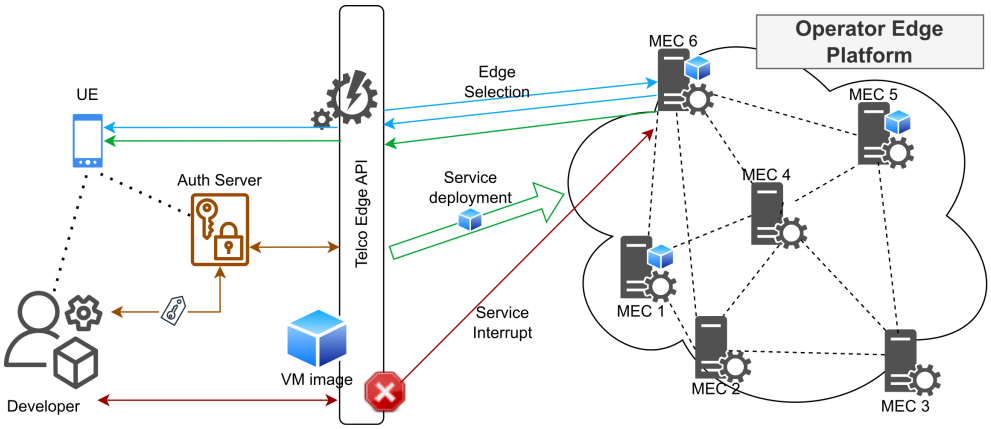

Figure 3.1: Telco Edge API working flow

3.2.1. Security

Security is an important part of the API's operational flow. The user authentication method stipulated by CAMARA and the GSMA for API implementation is OAuth 2.0. Summarizing, OAuth 2.0 is an authorization standard protocol that enables a secure way for applications to access and interact with a user's data on a third-party service, without exposing the user's credentials. It allows users to grant limited access to their resources (such as information, profiles, or data) to other applications, without sharing their passwords. OAuth 2.0 works by facilitating the issuance of access tokens by an authorization server, which the client application can then use to access protected resources on behalf of the user, following an authentication and authorization process [9].

As stated in the GSMA TS.43 Standard [10], There is an element within the telecommunication operators' architecture (generally known as Entitlement Server) that can serve as IdP (Identity Provider) for OAuth 2.0 token management and thus function as an authorization server. Its main role is to check whether a device (and user) is allowed or not to access a certain operator's service (Voice-over-WiFi, Voice-over-cellular, E-sim provision, etc.), but it has access to the Operator Core and can identify the user in both conexts, Data conection and wifi connection. However, it is also being examined this element's application in additional authorization and authentication scenarios.

In summary, it is important that the API developed complies with what is expected of it in terms of authorization. Due to the complexity of elements like the Entitlement Server and the lack of architecture available in the laboratory environment in which this project is carried out, it will be necessary to implement an adhoc Authorization Server to complete the security flows of the API.

Regarding authentication, it is crucial for the user to ensure they are connecting to the correct server to prevent attacks such as Man In the Middle or phishing, which can result in the loss of sensitive information. Therefore, a security layer based on SSL/TLS must be provided to the service to enable the user to verify our service's identity. There are open-source tools available for generating SSL certificates, such as OpenSSL [11], which can be used to complete this challenge.

3.2.2. Application onboarding and instantiation

To ensure effective application onboarding and lifecycle management, it is crucial that the standard interface adheres to the following requirements:

- Retrieval of the operator's Telco Edge and their status, sorted by location and filtered by status (active/inactive/unknown).
- Requesting the operator to instantiate the application instance to the most suitable Telco Edge site for a particular terminal, considering factors like connectivity, resources (vCPU, Memory, network interfaces, storage, GPU, etc.).
- Request for the operator to store application manifests. An application manifest is an archive file that contains all the essential information required to deploy an application to its run-time environment. This single application manifest moves from the design phase to run-time, encapsulating all the crucial bundles and metadata necessary for the successful deployment and execution of an application. This concerns the storage of container images or VM images and manifests detailing essential resources, Helm charts, and comparable items. It is also known simply as "application".

3.2.3. Optimizing Edge site selection for User Equipment

One of the primary benefits of Telco Edge Cloud technology is its ability to greatly reduce latency levels, a distinct advantage over traditional Cloud computing solutions. To achieve this, it is crucial for developers to connect to the most optimal Edge site, as outlined in section 2.2. This section concerns determining the optimal application service endpoint for a particular terminal, solely considering the shortest network path based on the network topology. Ultimately, this section applies to Edge Selection. As outlined above, this API is already defined in the CAMARA Standard but needs to be implemented.

The literature acknowledges the significance of selecting the most suitable edge site [12]. However, the referenced article suggests that this decision should also consider other parameters, such as measured latency or throughput. Additionally, the article proposes the use of Dijkstra-based algorithms. This project will solely concentrate on the established network topology and will consider only the best possible effort in these terms. The focus will be on the core elements of the network that provide service in this type of architecture to reduce latency.

3.2.4. Application and instance termination in the Telco Edge environments

Effective management of Edge Cloud services within the Telco Edge Cloud environment heavily relies on session termination. The API development should include the ability to request the operator to terminate a specific application's running instance, which ensures seamless resource management and termination of associated sessions. Furthermore, the system should enable the removal of current application manifests stored within the Telco Edge Cloud infrastructure. This feature enhances resource utilization efficiency and preserves a concise atmosphere by eliminating irrelevant or obsolete applications, leading to an optimized and proficient management of the Telco Edge Cloud ecosystem. Since the usage of such services typically requires a monetary investment from both users and developers, it is essential to have this function in place to manage resource utilization from the operator's perspective.

3.3. Conclusions

As mentioned in the introductory chapters, the CAMARA standard currently only defines the Edge Selection API. This API must be implemented to validate the definition's viability and provide comments in the industry forum. Furthermore, one of CAMARA's objectives is to define application lifecycle management in the Telco Edge through APIs.

In order to fill these gaps in the state of the art and in the standard (section 2.4), this project aims to develop and implement the Edge selection API. This task involves creating Python code, deploying it, and testing it within a Telecommunications operator's architecture. The API development includes implementing the authorization protocols described in this section, OAuth 2.0, and the authentication developments based on SSL/TLS.

As a part of this contribution, it is also foreseen to define, in Yaml format, an API for the complete lifecycle management of a given application at the Telco Edge and propose it to CAMARA edge subgroup as part of the activities done in this project. Note that the validation of this API could only happen once that the different partners agree on the contribution and therefore is not within the objective o this project . The definition should cover the main points outlined in this chapter.

By addressing the challenges presented in this report, the proposed developments and implementations are expected to meet the requirement for standardised interfaces for implementing

and deploying Telco Edge Cloud services. Furthermore, these programs ought to aid operators in integrating these interfaces into their architectures.

4 Design

4.1. Introduction

This chapter will provide details of the complete functional architecture and interfaces of a Telco Edge Cloud service, including, as defined in the analysis chapter, the application instantiation for the given service, the selection for the optimal site and the termination of the service, as well as the authorization and authentication procedures.

Firstly, this section will present implementation options for the Edge selection working track. This will involve discussing the most appropriate Edge Selection mechanism, as well as the deployment options of the service. Secondly, a functional architecture description will be provided, followed by the definition of the interfaces and methods that will make up the Edge Cloud API. In this working line, a Yaml file containing this definition will then be presented. Lastly, the security procedures involving the API consumption will be introduced and reviewed.

The solution will primarily focus on 5G architectures, while still acknowledging and referencing 4G architectures. For the sake of practicality and consistency with other sections, the functional architecture of the API will be modelled after Telefónica Spain's architecture, which may result in variation depending on the telecommunications operator.

4.2. Edge Selection Mechanisms

One of the primary concerns within the operator's realm is identifying the most suitable Edge site from the user's vantage point. It is imperative to identify the cluster that will provide the best latency to the user in order to meet the expectations of this service. Several mechanisms are expounded below for evaluation in this section, in order to justify which is more likely to be used given the requirements.

To give sufficient context, and referring to latter sections of this chapter, in order to reduce latency it is important to place the Edge sites as close as possible to the internet gateways. In the operator's domain, this element is known as the UPF, the element responsible for providing the internet gateway to operator's customers. This way, in general, the closest the Edge site to a UPF the lowest latency will be achieved.

4.2.1. Mechanisms based in latency measurements between Edge sites

The solution regarding latency reduction that first comes to mind is to measure the latencies between the various Edge sites and the terminal, and select the site with the lowest values. Latencies are typically measured using ICMP (Internet Control Message Protocol) traffic. In this way, could ICMP packets be sent from the terminal to all Edge sites to check the latency? Answer is NO as this type of traffic is often banned from operator networks due to its flooding capacity and so to avoid DoS attacks.

Anyhow, there are alternative methods to measure latency without using ICMP, such as using TCP traffic. While it is possible to flood using this protocol, it is easier to prevent such attacks as firewalls typically block ports from which a large amount of TCP traffic is being sent.

The concept is as follows: The network will utilise the UNI (User to Network Interface), as defined by the GSMA [6], to communicate with the terminal and request the use of TCP traffic to measure latencies between various IP:Ports (corresponding to the different Edge sites). After that the device would use the same interface to communicate the Network the results and the network would decide the best site for the user to connect.

The problem behind this is that it is not advisable to leave such key information in the hands of the terminal as it can be a conditioning factor in directing the user to other Edge sites than the one that offers the lowest latency, for example the one that meets the KPIs defined by the developer that is the least loaded. Also the functional solution may be complex to be developed as well as the variability in the devices' market leads into difficulties.

In conclusion, using this alternative as an Edge selection mechanism is not recommended due to security concerns related to the risk of flooding, the operator's inability to fully determine the optimal Edge site when the device is interfering and technical concerns regarding device centric solutions.

4.2.2. Mechanisms based in geographic location

The operator network can infer which gNodeB a particular user is connected to at any time. In the 5G standard architectures the Access and Mobility Management function (AMF) is the element in charge of managing the UE mobility within the different gNodeBs of the operator. This information of the radio cell a user is connected to, is sent to the UDM throughout the N8 interface as outlined in the 3GPP standard [13]. However, this information is also needed by the SMF to determine the best UPF the user can connect to, this is done through the N11 interface.

The concept behind this Edge selection approach is to direct the user to the nearest Edge site based on their geographic location obtained for instance at the device using GPS positioning. However, the physically closest Edge site may not necessarily have the lowest latency, as previously mentioned. This is because the SMF, which is responsible for assigning the most suitable UPF for a certain connection, does not follow an intuitive criterion. The SMF typically assigns the UPF to a terminal based on load levels among the different UPFs that can serve this user. These SMFs are typically distributed across geographic zones (e.g. East, North, West, and South) and each SMF serves a number of UPFs. Any UE within the zone ruled by the SMF can connect to any UPF independently of their geographic position. Therefore, a user may be connected to a radio cell in Valencia but access the internet through the UPF in Barcelona instead of the physically closest UPF in Valencia, as shown in the figure 4.1.

In regard to the figure 4.1 and as a conclusion to this section, It is evident that the most optimal Edge site to which UE-1 should connect is the Barcelona Edge site, as it is the closest to the UPF through which it would be accessing to the internet and therefore the one that will offer

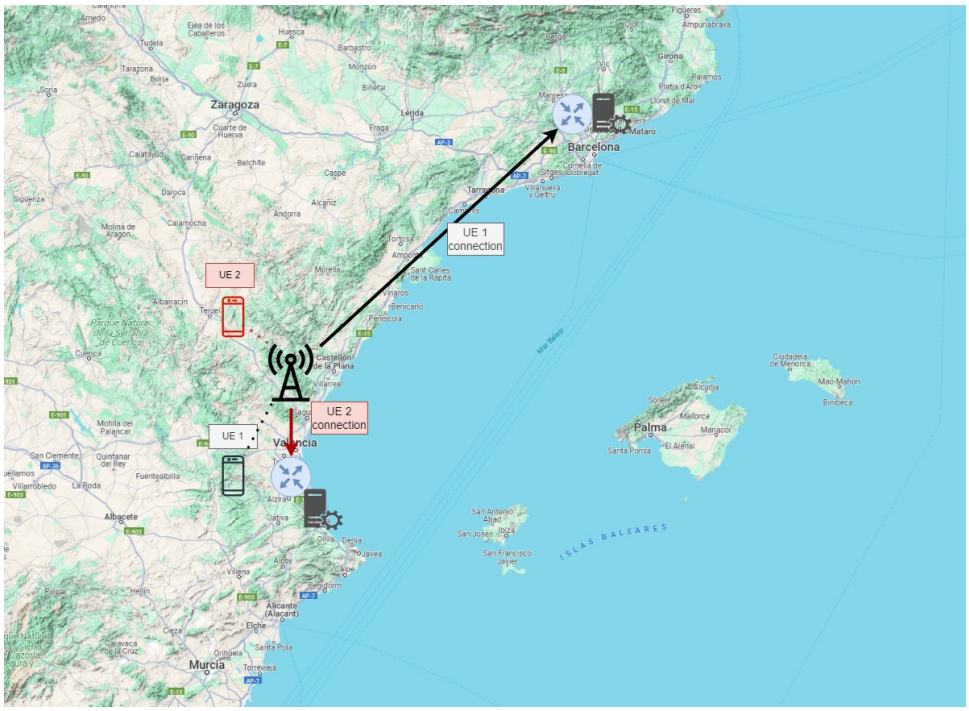

Figure 4.1: Shortest path to the UPF, comparison

the lowest latencies. On the other hand, UE-2 should connect to the Valencia site, as its UPF is located there. The problem with geo-based location mechanisms is that they tend to ignore the SMF tables and make connections based on the Edge site closest to the radio cell to which the user is connected. Therefore, in the above example, if such an Edge selection mechanism were followed, both terminals would connect to the Valencia Edge site, which is not the optimal for both.

4.2.3. Mechanisms based in network topology

Regarding the previous chapter, it is evident that the physically closest Edge site does not always offer the lowest latency. The approximation presented in this chapter suggests using the network topology, which is known by the operator, to infer the most optimal Edge site.

As previously defined, the Edge site is usually located near the User Plane Function (UPF), which acts as the user's gateway to the internet. By identifying the UPF to which a particular user is connected, it is possible to determine the optimal Edge site for connection in terms of latency. In standard architectures, it is possible to query the UPF to which a certain user is connected by requesting the Session Management Function.

Considering that Edge sites may not be deployed in all UPFs, especially in the early stages of the technology, some UPFs may not have an Edge site as close as others. The same Edge cluster may need to provide services to multiple UPFs. Therefore, it is essential to take this into account when implementing the API. Thus, it is proposed that during the deployment of the Edge sites, the design of which UPFs will be served by these new Edge sites and which UPFs should no longer be served by others should be drawn up. For this task simple measurements of latency between Edge sites and UPFs can be performed in other to check the closest one, but Edge deploying is not part of the scope of this project.

The conclusion of this section is that the optimal mechanism to be developed is the Edge selection mechanism based on network topology. Other mechanisms may lead to issues such as security problems or non-optimal Edge selection.

4.3. Edge Selection implementation options

There are a few options for deploying the Edge Selection API. Some operators have developed an API Gateway to expose these services, in line with the new paradigm stated by the Open Gateway initiative, where operator networks are supposed to expose their capabilities through APIs. Alternatively, the standard 5G architectures propose a new element called the Network Exposure Function (NEF), which has similar capabilities in unified interfaces.

4.3.1. NEF implementation

As expressed in the standard [14], NEF supports the exposure of RESTful APIs to Application Functions regarding the various elements of the operator architecture. Application Functions are simply defined as the consumers of this APIs.

In this sense, and referring to section 4.2 it is needed to expose the UPF a certain user is connected to through a NEF API. This can be done through the Traffic Influence API defined in the standards [14] [15]. For this particular API exposure, the SMF connects to NEF through N29 Southbound NEF interface [16]. Figure 4.2 displays the complete working flow of the Traffic Influence API. Interaction with the PCF (Policy Control Function) is shown, but it is not necessary for the acknowledgement of the UPF serving the user. It is used in another use case of this particular Traffic Influence API.

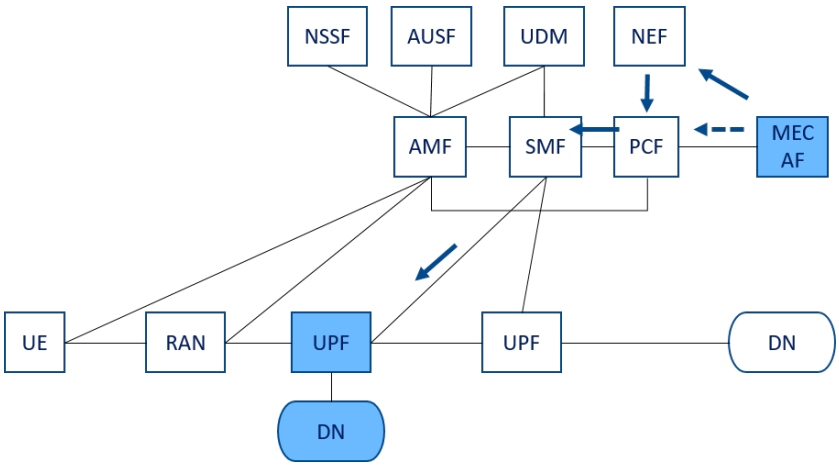

Figure 4.2: NEF Traffic Influence API architecture [1]

Another important feature that could be implemented thank to NEF is the stablishment of notification channels that could help to acknowledge the developer that the user has changed their UPF and so needs to update the serving Edge site. This can be performed thank to the NEF API Traffic Correlation as defined in the standards [14] [15]. Not only this but also the full framework of the Edge site discovery architecture that will be related in this project can be implemented using NEF.

The NEF APIs would be ideal for such projects as it efficiently exposes the required information. However, the industry is not yet ready for this implementation due to the majority of operator architectures being multivendor-based, which sometimes causes compatibility issues with interface and parameter support. Additionally, support for the NEF APIs is currently limited.

4.3.2. Operator API Gateway implementation

The Open Gateway initiative has prompted many operators to create API exposure layers that consume information from various network systems, including Core and commercial services, as well as value-added databases. Most of these systems are capable of exposing their information through REST interfaces, although some may have limitations. API exposure layers can use these interfaces to consolidate information into new interfaces, making it available to third parties.

For the specific case of the Edge Selection API and in accordance with what was explained in Section 4.2, what would be done would be to access the operator's session manager (AAA server or a similar) element) where the information about which UPF is providing service to a given user resides, in order to map the UPF to the Edge site and expose the most optimal one.

The main difference between the NEF approach and this design model is that the latter requires the creation of a greater number of logics, services, and even systems to access the information. On the other hand, the NEF consolidates the information through standard interfaces that are consulted directly from the network elements. As previously stated, the NEF is yet unavailable. Therefore, in this case, the design must proceed with the deployment of the API logic in the operator API Gateway.

4.4. Edge Cloud Service architecture

As indicated in the analysis stages, three main features are essential for the commissioning and operation of Telco Edge services. A developer may have some demands regarding latency or budget. It is possible that the service has flexible latency requirements, enabling deployment on a limited number of sites. Or they may want to cover a specific geographical area. Alternatively, they may need to provide a minimum service that covers the entire national area, necessitating deployment of the service across all operator sites. In any event, either the developer or the user should have the ability to discover the most suitable site for connecting to and accessing the deployed service. Finally, the developer must have the capability to conclude or terminate the service.

The entire API design hinges upon the Edge Selection mechanisms. In terms of efficiency and developer convenience, the chosen Edge selection mechanism relies on network topology. Given that network topology remains relatively stable over time, the concept is to allocate an Edge site to each User Plane Function (UPF) belonging to the operator. This allocation will be based on the sites that are topologically closest, which may not always equate to geographical proximity.

Figure 4.3 shows the expected working architecture for the solution described and the different elements depicted in the figure 4.3 and their role in the Edge Cloud API implementation are described below:

- **API gateway**: Generally, it is situated in the operator's micro-services layer and encompasses all the necessary logic for constructing this API and any value-adding software services that the carrier intends to put in place. It functions as a network exposure layer and is a crucial component in 5G architectures built around Network Exposure.

  There is a standard element defined by 3GPP which is the Network Exposure Function (NEF) [3] In further chapters it will be defined how the implementation of this API could be done following the standard procedures defined in 3GPP.

- **Authorization Server**: It is usually located in the operator's domain and its main purpose is to provide server-to-server authentication without the need to pass sensitive

client data. It is a fundamental part of the OAuth 2.0 authentication model, which is used by the APIs defined in CAMARA.

- **Edge site**: The Edge site is the Core element of the architecture. It is typically an advanced computing cluster located very close to both the end user and the operator's Internet gateway, minimising latency and increasing high computing capacity at the user's endpoint.

- **Developers and market places**: It is the agent which consumes the API. In general, the developer may access the API through an aggregator, or even through an Operator Platform as defined by the GSMA [6], anyhow the developer could also access the API directly though the API gateway but this is not desirable in security terms.

- **Session Manager Function (SMF)**: The SMF as defined by 3GPP [3] covers several functions. These functions involve managing sessions, including their establishment, modification, and termination. Moreover, the SMF has the responsibility of overseeing IP address allocations to User Equipment (UE) and authorizing them when necessary.

  Moreover, this element is in charge of defining to which UPF a given user connects based on Round Trip-Time (RTT) and load measurements and according to the network topology, trying not only to give a best-effort in terms of user connectivity but also to ensure network stability by balancing the network load.

  Hence, the SMF plays a vital role in the architecture of the Telco Edge. It furnishes data about the most suitable connection and routing path of the user within the operator network. Subsequently, this information can locate the least delay route that the UE needs to adopt.

- **UPF to Edge site mapping database**:This element is an ad hoc part of the Telco Edge solution. In order to achieve optimal selection of the Edge site by the UE, it is crucial to plan the mapping between UPF and Edge site to ensure that the Edge site provides service to the UPFs that are topologically closest to it. This way, the latency between UE and Telco Edge Cloud Service will be minimized.

  The way in which this element would be filled is during the deployment process of the Telco Edge sites: Once a site is deployed, it would be convenient to check which UPF can prove better Key Performance Indicators (KPIs) than the current ones and modify the mapping according to this analysis.

  This component is therefore significant in the architecture and could be characterised as either a database or a query file due to the small number of sites, probably less than a hundred.

- **User Equipment (UE)**: In wireless communications, refers to any equipment utilized by end-users to access mobile communication networks and services. Including smartphones, tablets, USB modems, sensors, cameras, tracking devices, connected vehicles, and other IoT gadgets.

  The User Equipment serves as the user's access point to mobile communication networks. The devices are fitted with wireless communication modules, enabling them to establish connections to cellular networks, access a broad range of services including calls, text messaging, internet access, applications and other mobile network-based services.

- **User Plane Function (UPF)**: As defined in the 3GPP standard [3], the primary function of the UPF is to serve as a gateway for access to both the network and the internet and other services (eg. other operators network) for a given UE or group of UEs. In 4G technologies this element corresponds to the junction of the Packet Gateway (PGW) and the Serving Gateway (SGW).

The UPF is a specific element of the operator's architecture and is typically deployed at geostrategic locations for reasons of network demand and availability. This is why during the time the Telco Edge Cloud platform is being deployed, Edge sites may serve one or more UPFs, although ideally a 1:1 ratio between UPFs and Edge sites would be desirable.

- **Virtual directory**: This element proves valuable (although not essential) for Network Exposure-based architectures since it allows for the gathering of all information pertaining to a user into a virtual profile. This simplifies the process of exposing and accessing user data within the network through built-in interfaces (such as REST and SOAP). For the case of Telefónica Spain architecture, this element is also known as Datagrid.

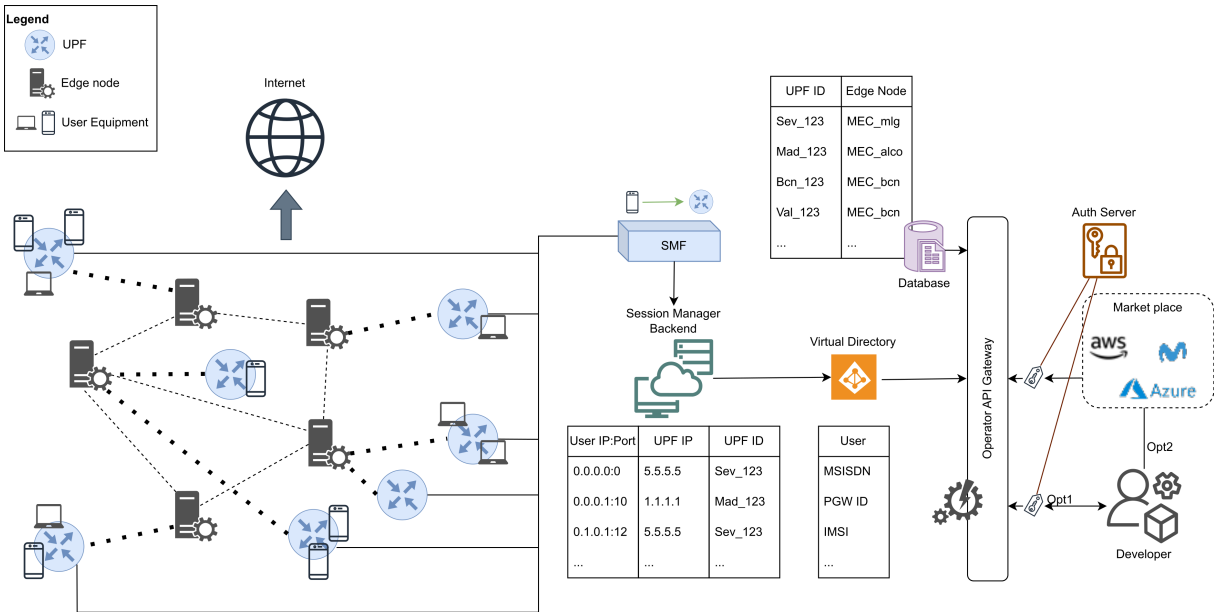

Figure 4.3: Telco Edge API working architecture

In terms of API consuming, the developer is the agent interested in knowing the optimal node of the User Equipment in order to provide the best service in their application. If the developer has the user's IP or MSISDN, they can submit the request to the Edge Selection API directly. However, when it comes to operator networks, developers typically do not have access to either of these parameters. Therefore, it is the responsibility of the front-end of the application application to request this information from the user's device. Once the response is received, it should be returned to the developer who, in collaboration with the operator's Edge platform, can generate the URL that the application front-end should consume to connect to the most optimal Edge site where the application is instantiated.

Concerning the API logic, it runs in the API Gateway and directly consumes the Virtual Directory APIs to obtain the alphanumeric parameter NAS-ID, which the operator uses to identify their UPFs. The API then uses that parameter to query the static database that relates the UPFs with the Edge sites of the operator.

Answering the question of how does the Virtual Directory obtains the UPF the user is connected to. The SMF has a back-end which performs a carbon copy of the contain of the SMF every time there is an update in this network function. This way the Virtual Directory just needs to query this database to obtain the NAS-ID parameter included in the user profile that this element handles.

4.5. Edge Cloud Service Lifecycle Management API definition

Throughout this thesis, the state of the art in terms of Edge Cloud APIs has been presented. It has been pointed out that the Edge Selection API was already defined in the CAMARA standards [17], so it was necessary to develop and implement this API. However, the Edge Cloud service cannot be considered complete without defining the application lifecycle management in CAMARA. Therefore, it is necessary to create the API definition before proceeding with development. This section aims to present the definition and justify the design of the Edge Cloud Lifecycle Management API.

From this first contribution it was expected to achieve a minimum viable product, therefore it was decided to define an API as simplified as possible yet complex enough to cover all the cases and therefore use only POST, DELETE and GET methods.

The design aimed to separate the app manifest and the app instance for two reasons. Firstly, it offers more versatility to the developer (agent interested in consuming the API) when instantiating their applications. Secondly, in the more consolidated cloud industry companies it is more common to operate in this way, so a developer is more likely to be used to the two entities working separately. The app Manifest comprises all the relevant information about the application to be deployed in the Edge sites, including the necessary technical and functional specifications such as the operating system, architecture, images, and repository where they are available. The term 'app instance' refers to the location within the Telco Edge architecture where the application (application manifest) is to be deployed or is deployed.

The expected pattern of operation is as follows:

1. The developer must know where they want to deploy their application since it is possible that they only need to deploy in a single Edge site, in a specific geographical area or in all the available sites, to address this issue, the developer can make use of the method *GET /Edge-cloud-sites* which will return a list with the available sites and their status.

2. The developer makes a request to acknowledge the Edge platform about their application files (images and VMs) and requirements (hardware and software needed), this is the mentioned application manifest and the request is done by *POST /app*. Once an application manifest request (POST) is done correctly, it is returned an *appId* that is needed as an input parameter for the rest of the methods (DELETE and GET) regarding application management.

3. The app must be instantiated and the developer would use the *appId* from the previous request as input for the *POST /appInstance* method to indicate which application needs to be instantiated and where (in a single site, in a geographic zone defined by the operator or in all the sites). As in the previous request, *appInstanceId* will be returned and will be needed for the rest of the methods (DELETE and GET). However in this case the *GET /appInstance* method can be also performed using the *AppID*.

Explaining the fit between this API and the Edge Selection API presented in the architecture of the previous section and as already mentioned, the developer must know the most optimal Edge site among those deployed by his application in order to provide the user with the most optimal service possible or at least meet the requirements of the application. Once this parameter is obtained, he will generate the URL that his application should consume. This way, after the above pattern of operation have been completed, the specific methods for the Edge Selection API (GET) would be performed.

However, it is central to note that for this specific implementation is not contemplated the mobility of the user, so based on network measurements or performance indicators of the

application, the calls to the Edge Selection API would need to be performed several times in the meanwhile the application is running. Notifications regarding the changes in the UPF that the user is attached to would pose an important improvement that could be implemented using NEF as explained in the section 4.3.1.

Annex A provides further details on each of the parameters indicated in the data model in the figure 4.4 which shows the dependencies and the different parameters encountered in this API. This information can also be founded in the Yaml file itself.

The API definition in Yaml format can be found in the following GitHub repository and the changes report and evolution in the following Pull Request:

- GitHub-camaraproject/EdgeCloud: code/API-definitions/EdgeCloud-LcM.Yaml [18]
- GitHub-camaraproject/EdgeCloud: PR-154. EdgeCloud LifecycleManagement API proposal. [19]

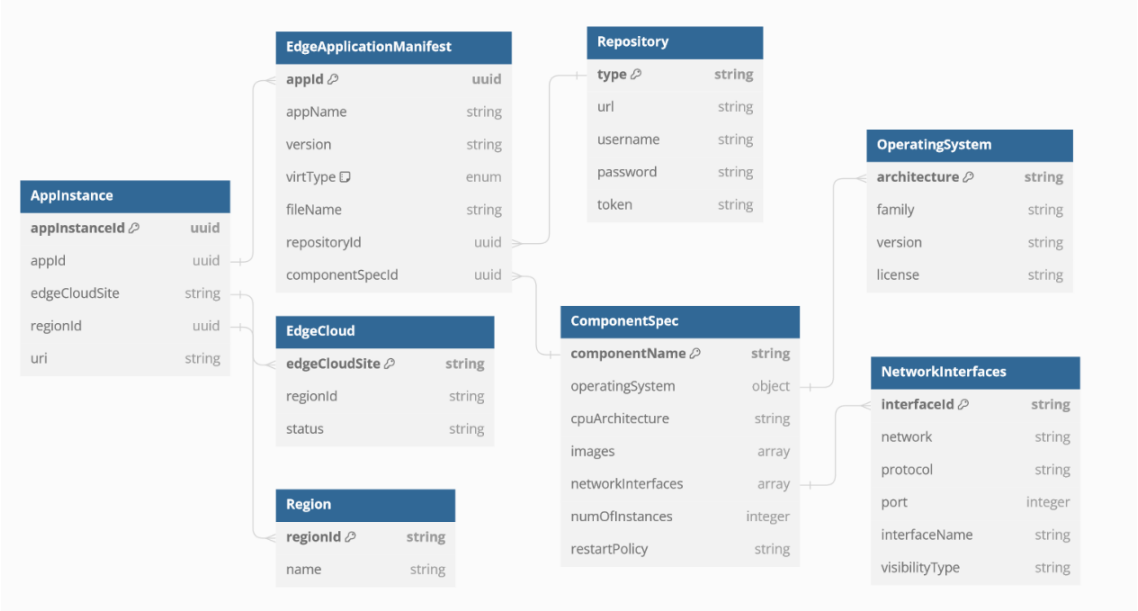

Figure 4.4: Edge Cloud lifecycle management data model

4.6. Authorization and authentication procedures

Authorization

As per CAMARA guidelines [20], the API authorization procedure must be based on OAuth. Section 3.2.1 introduced the possibility of using the Entitlement Server for user authorization. However, the existing laboratory architecture has limitations, requiring the creation of a specific program to mock an OAuth server.

The Auth Server must be able to verify the user's identity and manage access to the different URLs offered by an application or service while keeping the user's credentials isolated from the service itself. In these terms, a Python code must be developed to comply with the authorization requirements based on OAuth. The expected operating model is as follows.

1. The user will be provided with credentials that will be used to request a single-use token from the Auth Server. This emitted Token will be stored in the Server token collection.

This collection will be used to verify the validity of the token, operation that can only be performed once since the token will be erased from the collection after verification.

2. The server would maintain a list of URLs that the user is authorised to access, establishing a relationship between the token and the accessible URLs.

3. An application or REST service requests the user's token and uses it to request authentication from the server. The server checks if the requested URL is allowed and if the token is valid. Therefore, the Auth Server will return an error in two cases:

   - The token is no valid as it is not stored in the Server token collection. This could have happened because the Token has already been used (and therefore deleted) or because it was never been created.
   - The token exists in the Server token collection but the user is not allowed to access the requested URL

All this process is illustrated in figure 4.5

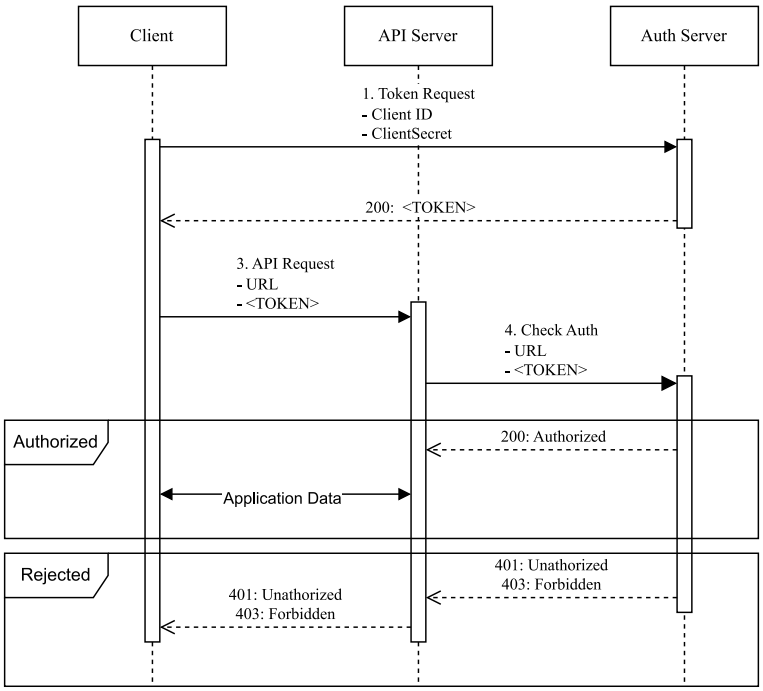

Figure 4.5: Authentication flow

Authentication

In terms of authentication, it is crucial that an internet connection meets specific requirements regarding security. The TLS (Transfer Layer Security) protocol, fulfils the aforementioned security requirements. It authenticates the identity of the parties involved (client and server), ensuring that each entity is who they claim to be. Additionally, it guarantees the integrity of the messages sent, preventing any tampering or alteration during transmission. TLS establishes a secure communication environment by employing encryption to safeguard the confidentiality of the information exchanged. This generates a protected and reliable channel for the transmission of any kind of data on the internet [21].

The TLS protocol utilises a handshake process [2] to establish a secure connection between a client and a server. This process involves mutual authentication, negotiation of security parameters, and the exchange of cryptographic keys for secure communication. SSL certificates, now

commonly known as TLS certificates, play a crucial role in this process. These digital certificates are issued by certification authorities and are used to authenticate the identity of a server. The TLS handshake is a process where the client and server exchange SSL/TLS certificates to verify the server's identity and generate secure session keys for encrypting communication. The use of SSL/TLS certificates is essential for authenticating and establishing a secure connection that ensures the privacy and integrity of information transmitted over the Internet. Figure 4.6 shows this process.

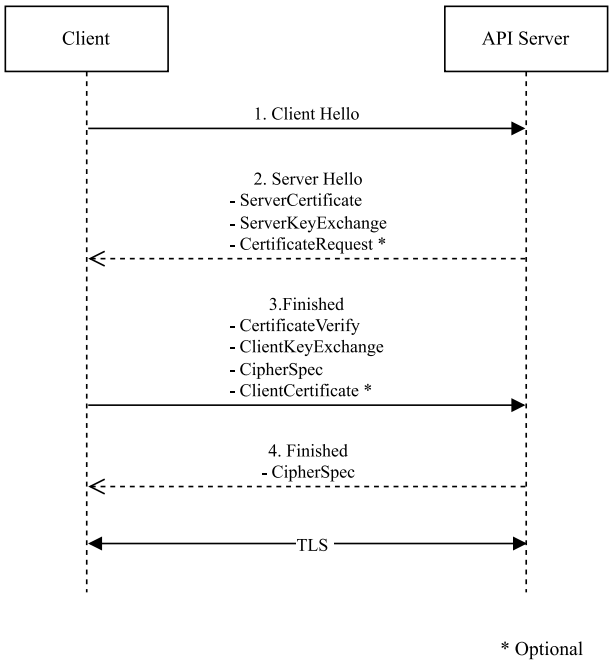

Figure 4.6: TLS Handshake [2]

Therefore, three key elements are required to establish a secure connection based on TLS:

1. The server's certificate. This is usually a file that contains information about the server's identity, its public key, and is signed by a trusted Certificate Authority (CA).

2. The public and private key pair. The public key is usually part of the server's certificate and is used in the asymmetric encryption process to establish a secure connection. Clients use this key to encrypt data that only the server can decrypt with its corresponding private key.

3. The Certificate Authority (CA) certificate. This certificate is the root or intermediate certificate that verifies the authenticity of the server's certificate. Browsers and systems rely on these issuing entities to validate the legitimacy of server certificates.

In relation to Certificate Authorities (CA), obtaining a certificate is typically a straightforward process. The CA verifies the domain's identity and issues a certificate that is only valid for that specific domain and for a limited time. However, in the context of this project, all development is being carried out on a private laboratory network that will only be accessible to a limited number of users (via VPN). Therefore, obtaining such a certificate is impractical due to the restricted environment. The most efficient option is to use self-signed certificates created with tools such as OpenSSL [11]. Then, configure client devices to recognize and accept these self-signed certificates as valid. Similar methods will be used to generate both server certificates and encryption keys.

4.7. Conclusions

In conclusion, this chapter has satisfactorily defined the different design models for implementing Edge APIs. Regarding selection mechanisms, while it is useful to know all available methods, the Edge selection method based on network topology is the most optimal. This method allows for greater control over a key decision and is functionally superior.

As far as the implementation of APIs is concerned, although the NEF offers a very good solution through the Traffic Influence API to essentially develop the Edge Selection API, it is unavailable and therefore it is necessary to develop an abstraction layer on top of the operator architecture to consume the information from the different systems of the operator, an API gateway. The proposed architecture have been explained above.

In relation to the application lifecycle management API, a satisfactory definition has been achieved for this class of API in CAMARA. The design has focused on a simplified yet comprehensive approach, which separates the app manifest from the app instance to offer developers flexibility and align with industry practices. The outlined operational pattern involves specifying deployment locations, creating an application manifest, and instantiating the app, utilizing GET, POST and DELETE methods.

Adhering to CAMARA guidelines it is needed to support bioth authorization and authentication methods. Implementing an OAuth-based API authorization method will be part of this project. As the lab setup poses limitations, developing a mock OAuth server will be necessary. This server will be governed by Python code, following OAuth standards and managing user access to URLs through single-use tokens temporarily stored for validation. Requests against these tokens and URLs will be authenticated by the server, which will identify and report errors for invalid tokens or unauthorized URL access.

TLS protocols will ensure secure internet connections by providing mutual authentication and encrypted communication via SSL/TLS certificates. Establishing a secure TLS connection will require the server's certificate, a public key for encryption, and a Certificate Authority certificate for authentication. Given the restricted lab environment accessible solely via VPN, using self-signed certificates created using OpenSSL will be practical for ensuring secure connections and authentication.

5 Implementation

5.1. Introduction

This section will describe in detail the technical contribution of this project to the telecommunications field. In this sense, a detailed description of the Edge Selection service architecture and the elements that interact in the implementation of the service will be provided. As previously stated, this API development will concentrate on selecting Edge sites based on the network topology.

The contribution consists of a Python code that implements the Edge Selection API according to its standardisation in CAMARA. The code's scope encompasses the development of an identity manager that adheres to OAuth 2.0 authorization requirements. Additionally, a security layer based on SSL/TLS is included to comply with the aforementioned authentication requirements.The architecture of Telefónica Spain plays a significant role in the deployment of this API. The API will be containerized using Docker and deployed in a Telefónica laboratory very similar to the production environment.

The complete implementation is available in the following GitHub repository:

- GitHub-RicardoSerr/EdgeSelection [22]

It should be noted that this work has also made a slightly more theoretical contribution in the context of telecommunications. As described in chapter 4.5, a YAML following the CAMARA guidelines has been released. This definition aims to relate all interfaces and operations of a complete Telco Edge service.

5.2. Deployment Architecture

API consumption involves a client making a Curl request to the developed API, which runs in a Docker containerized instance. The OAuth Server, which has also been developed in Python and runs on a separate instance and server than the Edge Selection API, will return a Token to the user after they have used their credentials to access it. The client will then use the Token to access the API resources and the API instance will then make a new request to the OAuth Server with the Token to verify whether the user is allowed to access to the requested resources.

Both Python instances (OAuth Server and Edge Selection API) will use JSON files to build the final response. In terms of scalability, JSON files are sufficient for storing information about Edge sites against UPFs for the Edge Selection API implementation, as the number of Edge sites and UPFs is not large. However, for the OAuth server, a more complex service may be required depending on the number of clients consuming this API.

Section 4.3.2 suggests that the most appropriate deployment option is to deploy the API in the Operator API Gateway. This server, located in the operator context, is connected through its South Bound interface to various elements in the operator network plane, including Core, operator databases, and other services. It exposes this information using a North Bound Interface, typically based on REST and/or SOAP.

The API Gateway developed by Telefónica is named SDF, which stands for Service Delivery Framework. The API Gateway shares the same interfaces as the production SDF for all network elements, but only the Virtual Directory interface is required for this API. It was related in the section 4.4 the value of this particular element, the Virtual Directory. The system allows for the aggregation of various network element data and the establishment of a virtual profile for each user, enabling seamless access to user profiles. The NAS-ID field is one of the parameters included, which identifies the PGW/UPF to which a specific user is connected. These elements must be accessed via a Virtual Private Network (VPN).

Access to the Virtual Directory from SDF can be achieved through a REST interface. As both elements are on the same network (VPN), the request is simplified. In turn, Virtual Directory connects to the SMF backend to query the NAS ID field. The SMF backend is a database that stores all operator sessions. In the case of Telefónica, this database is accessed by the Virtual Directory using a Lightweight Directory Access Protocol (LDAP) interface.

Figure 5.1 illustrates the deployment environment and the flow of API consumption, encompassing all the elements mentioned in this section.

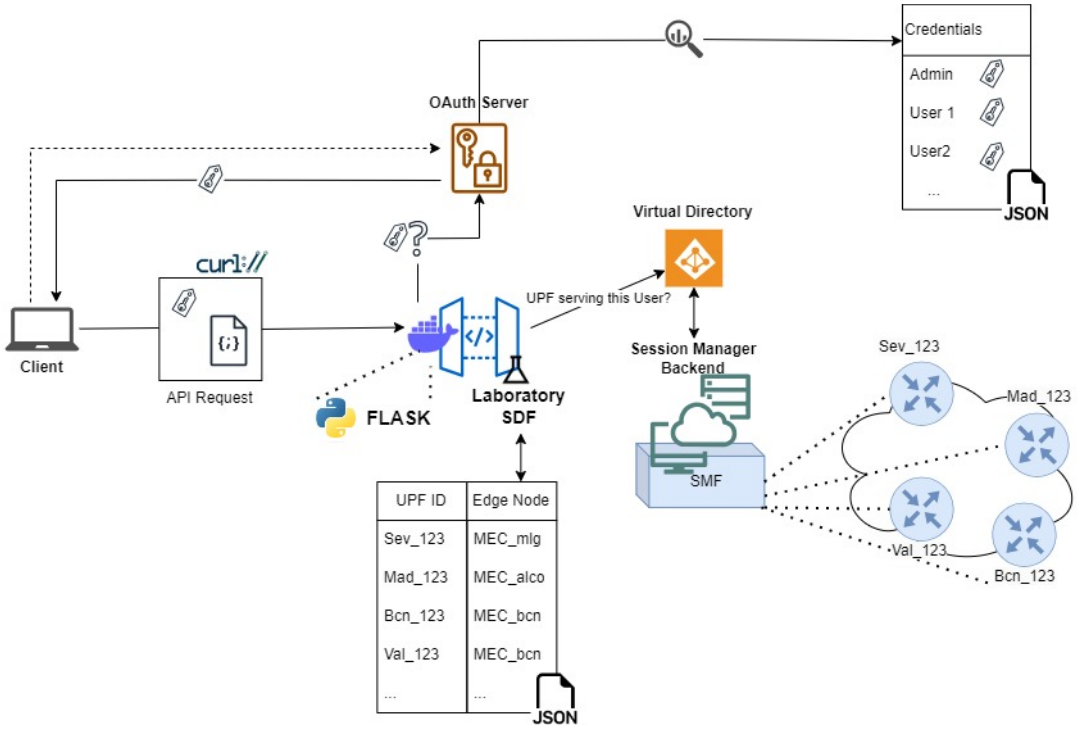

Figure 5.1: Edge Selection API architecture

5.3. Authorization: OAuth Server

The CAMARA guidelines [20] require the handling of API authorization procedures using the OAuth2.0 implementation. While a dedicated IdP server within the operator architecture could have been used for this purpose, the design chapter explains that it is too complex for the scope of this project. This way, a prototype is developed to fulfill with this requirements as the main idea behind these is that the API can handle credentials and authorization tokens to limit the access to its resources. This functionalities are encapsulated in a Python application, **oauthServer.py**, which can be found in the */code* folder of the GitHub repository [22].

The OAuth server prototype has been implemented in Python using the Flask framework for exposing the capabilities through the network. The program generates tokens that are stored in a JSON file (**tokens.json**) for later validation. Each generated token can only be used once providing an additional layer of security for user authentication. However, authorization is also a core functionality of the program, the tokens are generated using user credentials linked to RegEx patterns that indicate the API endpoint accessible to the user. Storing these credentials and endpoints in separate files and protect them with a password is a good practice, although they are currently written directly in the code as server is for demonstration purposes only.

The Flask application includes two essential routes: */checkToken* and */generateToken*. The */checkToken* endpoints is designed to handle POST requests for validating the token's authenticity and determining whether the associated user has the necessary authorization to access a requested URL. This process involves checking if the access token exists, confirming the user's authorization based on their allowed URL patterns, and subsequently granting or denying access.

On the other hand, the */generateToken* route facilitates the generation of new access tokens when presented with valid client credentials, specifically the client ID and client secret. This route serves as the entry point for clients to obtain the necessary credentials for accessing protected resources. If the provided client credentials match any user in the predefined list, the script proceeds to generate a new access token using a secure method (*secrets.token_hex(16)*) and associates it with the client ID. The generated token is then stored, overwriting any existing tokens associated with the same user.

Although this implementation may be secure, it requires additional security implementations. It has only been implemented to simulate the environment in which the Edge Selection API may run. For production environments, it is recommended to use more robust authentication servers such as the operator IdP.

5.4. Authentication: SSL/TLS Certificates

As stated in the Design chapter, security is a critical consideration. In this sense, users must be confident that they are accessing the correct service, and the server on which it is deployed must acknowledge them. SSL/TLS certificates are useful in addressing this issue because they not only enable the server to authenticate itself but also enhance the connection between the client and server by providing encrypted communication. In section 4.6, the necessary elements for implementing an SSL/TLS layer to the developed service were outlined. The server will utilise these certificates, resulting in the use of HyperText Transfer Protocol Secure (HTTPS) instead of HTTP for the server's URL. These considerations are also defined as required by CAMARA guidelines [20] requesting support for HTTPS.

OpenSSl is an opensource implementation of the SSL and TLS protocols and enhances the user to create SSL/TLS certificates in nearly every Linux system [11]. In this implementation OpenSSL is used to generate the server certificate, public and private key pair and the CA

certificate.

5.4.1. Generate the CA Certificate

First thing to be done is generate the CA certificate, the idea behind this approximation is that the server running the API service (SDF) will be its own Certification Authority for the reasons outlined above. The following command is used to generate the private key that will be signing the self-generated CA Certificate:

```
openssl genrsa -aes256 -out ca-key.pem 4096
```

The command is disclosured as follows:

- *genrsa*: This sub-command is used for generating an RSA private key. RSA, or Rivest-Shamir-Adleman, is used as the encryption system as it is one of the most widely used mechanisms [23].
- *-aes256*: This option specifies the cipher to be used for encrypting the private key. In this case, AES-256 encryption is used to protect the private key. This adds an extra layer of security by encrypting the private key with a passphrase. It will be prompted to enter a passphrase during the key generation process. It is crucial not to share this private key, as anyone with it will be able to generate signed certificates by the CA that this server represents. The passphrase aims to avoid this anyhow.
- *-out ca-key.pem*: This option specifies the file where the generated private key will be saved. In this case, the private key will be saved in a file named **ca-key.pem**.
- *4096*: This parameter specifies the number of bits in the RSA key. In this example, a 4096-bit key is generated. Larger key sizes generally provide stronger security, but they may also have an impact on performance.

The following command is using the private key generated previously (**ca-key.pem**) to sign a generated the CA Certificate. The previous used passphrase will be required for the signing process as well as some details regarding the server (Country Name, Organization Name, Common Name etc.) that will be the used by the browsers and clients to validate the Certification Authority:

```
openssl req -new -x509 -sha256 -days 365 -key ca-key.pem -out
   ca.pem
```

The command is disclosured as follows:

- *req*: This sub command is used for certificate requests and related functions.
- *-new*: This option specifies that a new certificate request should be created.
- *-x509*: This option indicates that a self-signed certificate should be created, rather than generating a certificate signing request (CSR) to be sent to a certificate authority.
- *-sha256*: This option specifies the hash function to be used, in this case, SHA-256, for creating the digital signature of the certificate.
- *-days 365*: This parameter sets the validity period of the certificate in days. In this example, the certificate will be valid for 365 days.

- *-key ca-key.pem*: This option specifies the private key to be used for signing the certificate. In this case, the private key from the file **ca-key.pem** is used.
- *-out ca.pem*: This option specifies the file where the generated self-signed certificate will be saved. In this case, the certificate will be saved in a file named **ca.pem**.

5.4.2. Generate the Server Certificate

A new RSA key is generated for the Server Certificate without the need for a passphrase, as the signing process is less sensitive than the previous one. The following command can be used, it is very similar to the previous one but the AES encryption is not done:

```
openssl genrsa -out cert-key.pem 4096
```

The following command generates a certificate signing request (CSR) for a certificate with the subject's Common Name set to "EdgeSelection." The private key from **cert-key.pem** is used to associate the CSR, and the resulting CSR is saved in a file named **cert.csr**. The CSR can then be sent to a certificate authority (CA) for signing, or it can be used with a self-signed certificate as in this case.

```
openssl req -new -sha256 -subj "/CN=EdgeSelection" -key
   cert-key.pem -out cert.csr
```

It is important to indicate in the certificate which IP or DNS domains it is intended for. In this case, the certificate will be used for the IP 10.0.0.1, which is the SDF IP and no DNS domain will be included. This information can be included in a configuration file to be used later in the certificate signing process. The following command can do this:

```
echo "subjectAltName=IP:10.0.0.1" >> extfile.cnf
```

Finally, the following command takes the CSR (**cert.csr**), signs it using the CA's private key, and produces a signed certificate (**cert.pem**). The additional configurations specified in **extfile.cnf**, such as subject alternative names, are taken into account during the signing process. A serial number file (**ca.srl**) is created or incremented to keep track of issued certificates.

```
openssl x509 -req -sha256 -days 365 -in cert.csr -CA ca.pem
   -CAkey ca-key.pem -out cert.pem -extfile extfile.cnf
   -CAcreateserial
```

With that, the certificates chain involving the Server Certificate (**cert.pem**) and the Server Private Key (**cert-key.pem**) is created. But the configuration is not finished. The client needs to trust the CA that the **ca.pem** represents as the **ca.pem** file was self-signed and the client will not have that CA as trusted in their certificates repository.

5.4.3. Client configuration

To establish trust in the **ca.pem** file, the client must include it in their trusted CAs repository. In this case (Ubuntu 20.04 distribution), the CA certificate (**ca.pem**) should be moved to */usr/local/share/ca-certificates/ca.crt*. This involves copying the content of **ca.pem** to the

ca.crt file in the repository [24]. After that, the following command can be used to update the configuration:

```
sudo update-ca-certificates
```

The SSL/TLS configuration is now complete. To enable full TLS communication, the client must include the CA Certificate in the Curl request to the Flask server developed for the API.

5.5. Edge Selection API

As far as the Edge Selection API is concerned, Flask, a micro-framework for Python, is an excellent tool for building web applications and APIs. In this context, a Flask-based application was created within the python file **edgeSelectionAPI.py** to act as a gateway to retrieve the most optimal Edge site for a given user, depending on the UPF they are connecting to. The application is designed to verify OAuth tokens, handle various error scenarios, and interact with external services to provide detailed information about Edge sites as well as administrator management. This file is also available in the */code* folder of the GitHub repository [22].

5.5.1. OAuth Token Verification

The ability to verify OAuth tokens is crucial for the application to encure the security and integrity of requests. The *verify_ token* function acts as the gatekeeper, sending a request to the developed OAuth server endpoint (http://192.168.1.65:2000/checkToken). This function checks the validity of the token and the user's access rights to the requested URL. Upon successful verification, the function returns a boolean indicating the token's validity and, if invalid, provides error details such as HTTP status codes and error messages.

5.5.2. Response Structure and Error Handling

The response should follow the Simple Edge Discovery CAMARA definition [17], including only the name of the most optimal Edge and, optionally, the provider of that Edge site (in this case, Telefónica). Some discussions and intents within the Edge Cloud CAMARA group suggest retrieving a list of the Edge Sites offered by the Edge provider, this has been implemented and will be further extended in the upcoming sections.

Regarding error handling, an *ERROR_ MESSAGES* dictionary has been created to map the most common HTTP status codes required in the CAMARA definition of the API to meaningful error messages. This way, the program gracefully handles a variety of errors, such as invalid headers, unauthorized access, forbidden requests, and internal server errors. The entries of this dictionary are widely used throughout the rest of the code.

5.5.3. GET Method: Retrieve optimal Edge Sites

As with the other methods of the API endpoint, this method first checks whether the Auth Token is present in the request and, if validated by the OAuth server, the program will continue executing. Otherwise, it will return the error code and message retrieved from the OAuth server, if available. If not, the generic error 401 from the *ERROR_ MESSAGES* dictionary will be the response.

After checking the Token, the program verifies which attributes were parsed in the request. One of the attributes checked is MSISDN, which stands for Mobile Station Integrated Services Digital Network, but is commonly known as 'phone number'. The other attribute is the IP address of the device. If neither of these attributes are parsed in the request, a list of all Edge sites is returned. Otherwise, the program uses these attributes to make an HTTP REST request to the Virtual Directory.

Handling the Virtual Directory responses is crucial in order to provide the accurate information in the final return of the developed API. Table 5.1 shows Virtual Directory operation codes and their meanings.

Operation Code	**Meaning**
00	The requested MSISDN or IP have been found and the operation have been completed successfully.
02	The request is badly constructed eg. No MSISDN nor IP is parsed.
03	The MSISDN or IP requested could not be found. This may be due to the user being disconnected from the network or not being registered in Telefónica's user databases.

Table 5.1: Virtual Directory Operation Codes

By capturing these errors, HTTP responses can be constructed more accurately. If the operation code of the request to the Virtual Directory is successful, it is checked in the **dataseed.json** file to determine the Edge site that corresponds with the obtained NAS-ID. The response is then built. If the operation code indicates that the request was not successful, an internal error in the Virtual Directory is returned, indicating that the user was not found. If any other error aside the program occurs, it is returned.

5.5.4. GET Method: Retrieve all Edge Sites Details

As will be explained in later sections, the proposed deployment of the API using Docker containers makes managing and administering the service a challenging task. Any changes require creating a complete Docker image and deploying it in the API gateway for consumption. It is proposed to create this GET method to provide the administrator with the current mapping that the API back-end is doing between Edge sites and UPFs.

In these terms, the Edge Selection API also supports retrieving information about all available Edge sites. This functionality is implemented through an additional API endpoint, /adminEdgeSites, designed for administrator operations. The program retrieves the complete **dataseed.json** file after verifying the validity of the parsed token.

5.5.5. PUT Method: Updating mapping file

After the administrator has checked the mappings using the previous API call, they can use the same endpoint with the PUT method to update the **dataseed.json** file. The Token parsed must be checked using the already mentioned *verify_token* function, which calls the OAuth server endpoint.

5.5.6. SSL/TLS Configuration

The application is configured to run over HTTPS, utilizing SSL/TLS. The ssl module in Python is employed to load a certificate chain, a private key, and CA certificates that were

previously generated. This ensures secure communication between clients and the server. As explained previously in section 4.6 and 5.4.

5.6. Service Deployment

The most straightforward and logical way to deploy the developed API service is to simply run the Python instance in the background and store the logs in a separate file. To achieve this, the following bash script can be used. In this script, the symbol **&** runs the process in the background, and **2>&1** redirects the standard output to the **output.log** file. The PID (Process ID) can be used with the command **kill** to finish the process:

```
#!/bin/bash
python_app="edgeSelectionAPI.py"
log_file="output.log"

# Run Python app in background and redirect output to log file
python "$python_app" > "$log_file" 2>&1 &

# Get the PID of the last background process
pid=$!

# Display the PID
echo "PID:$pid"
```

However in terms of scalability and portability of the service this deployment alternative is not efficient. The program needs to be portable as it may need to be deployed in several systems. Scalability is also crucial as the systems in which the program instances will be running will be loaded with other program instances and software running. In this terms containers proves value, Docker provides a lightweight and efficient way to encapsulate an application along with its dependencies, libraries, and runtime environment into a portable container. This containerization ensures consistency across different environments, making the application easily transferable between development, testing, and production environments. Docker's isolation capabilities enhance security by compartmentalizing the application, reducing the risk of conflicts between dependencies. Additionally, Docker simplifies scalability as multiple containers can run simultaneously on a host system, optimizing resource utilization.

The creation of a Docker image involves specifying the necessary Python libraries and files to be included in the image. This ensures that the image is isolated and that the program instance always uses the same version of the libraries, regardless of the environment or system on which it is running. However, this also makes it difficult to modify the files with which the image was created. For instance, modifying the **dataseed.json** file requires creating a new Docker image with the modified file, which may not be desirable. This is why GET and PUT methods were included in the API implementation and described in section 5.5.4 and section 5.5.5 respectively but are only accessible to administrators.

5.6.1. Containerized API Deployment: Creating a Docker image

After installing the Docker framework, it is necessary to create a **Dockerfile**. This file should include all the required elements and files for creating the Docker image, as well as information about the framework used (Python 3.11), execution commands to be used inside the image, and the port where it should be launched. The used **Dockerfile** is as follows:

```
# Use a base Python image
FROM python:3.11

# Set the working directory inside the container
WORKDIR /app

# Copy the necessary files into the container
COPY edgeSelectionAPI_v2.py /app/edgeSelectionAPI.py
COPY dataseed.json /app/dataseed.json
COPY requirements.txt /app/requirements.txt
# Copy SSL certificate files into the container
COPY cert.pem /app/cert.pem
COPY cert-key.pem /app/cert-key.pem
COPY ca.pem /app/ca.pem

# Install the application dependencies
RUN pip install -r requirements.txt

# Expose the port on which the Flask application runs
EXPOSE 2023

# Command to run the Flask application
CMD ["python", "edgeSelectionAPI.py"]
```

The files involved in the image creation are the Python program itself **edgeSelectionAPI_v2.py**), the json file that matches the UPF list with the Edge sites (**dataseed.json**), the SSL/TLS certificates required in the Flask server that were self-signed and generated in section 5.4 and the file named after **requirements.txt** that includes all the needed libraries for the images and their version, for instance *Flask==2.2.2* and *pyOpenSSL==23.0.0*.

For creating the image the following command can be used:

```
docker build -t <image_name>:<tag> ./files_path
```

The following command can be used to check the existing images:

```
docker images
```

Once the image is created the following command launches the Docker instance in the background (argument **-d**). Port 2023 of the container maps to port 2023 on the local system (argument **-p**). :

```
docker run -d -p 2023:2023 <image_name>:<tag>
```

The following command can be used to check the running containers:

```
docker ps
```

The following command can be used to check the logs of a running container:

```
docker logs <container_ID>
```

The following command can be used to kill a running container:

```
docker stop <container_ID>
```

5.7. Conclusions

One of the main goals of this project was to validate the CAMARA definition of the Simple Edge Discovery API by implementing the API in a realistic environment. This chapter fulfils this challenge by discussing the deployment process. To be considered valid, the deployment needed to support security features for authorization and authentication.

In relation to authentication,due to the complexity of the IdP elements offered by the operator, a Python-based prototype Auth server was developed using Flask. This allowed the API to make the same request as it would to a real IdP.

Authorization is handled by creating self-signed certificates to ensure that the user is connecting to the correct service and to enhance the connection with TLS encryption. Self-signed certificates offer good versatility in testing environments and let the developed API to simulate the handling of certificates to be used in production environments.

With all that, the Python-based Edge Selection API that have been developed is designed to work with all these security elements as well as interacting with the operator elements. These operator elements are based in the real infrastructure yet in a pre-production environment.

In summary, the main challenges for achieving the proposed objectives in the project have been accomplished. These include authentication, authorization, and API development and deployment in the operator architecture.

6 Testing

6.1. Introduction

To validate the work done in the previous chapter, it is important to test all the developed features. This chapter will focus on validating the implementation and the necessary unit testing required to build the solution.

Firstly, the testing architecture will be discussed, and may differ slightly from what has been presented in the implementation and design chapters, in order to cover all corner cases. Subsequently, each element involved in the implementation will be tested to justify the work methodology employed in the Edge Selection API implementation process. Finally, an end-to-end test will be conducted to validate the complete implementation.

For the testing and validation, some BASH scripts were developed and are available in the same GitHub Repository as the rest of the code:

- GitHub-RicardoSerr/EdgeSelection [22]

This chapter will also demonstrate the importance of using encrypted communication for these kind of services. Specifically, packet analysis tools such as Wireshark will be used to assess the risks of sharing application data without encryption.

6.2. Testing architecture and tools

The API operates by receiving HTTP requests from a device, typically a mobile device. A Linux computer is set up for testing purposes to simulate the request that would be made by the mobile device.

The Linux computer will use CURL to perform HTTP requests to the various elements involved in the API's operation, as illustrated in the figure 6.1. These include a request to the Virtual Directory to understand the type of response this element receives, the Edge Selection API to ensure the service is functioning correctly, and the OAuth server used to test performance with and without TLS/SSL as well as the expected behaviour in therms of authorization.

Wireshark will be used to capture the traffic between the Linux computer and the OAuth server both with and without using TLS/SSL to evaluate the advantages of this sort of secure connections. The OAuth server will also be tested to determine if certain users' credentials are limited in terms of accessibility to specific resources of the API.

The Edge Selection API will undergo testing to verify the accuracy of its responses, as well as the functionality of the implemented HTTP methods and the overall service workflow.

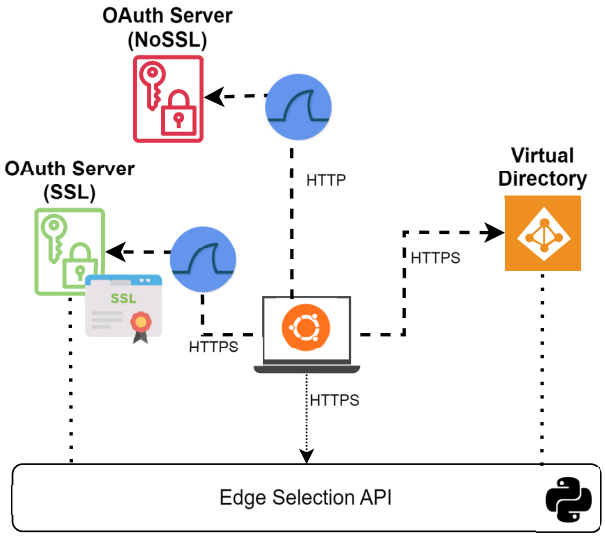

Figure 6.1: Testing architecture

6.3. Unitary testing

This section aims to justify the decisions made during the implementation phase as well as the design itself. It is necessary to understand the way in which the responses from the operator architecture (Virtual Directory) are made, as well as the gaps in the given information that need to be fulfilled.

In the same way that requests can be made to the different systems within the implemented REST program, the same requests can be made externally from any other system (as long as it is connected to the same VPN as it is a pre-production environment). This is exactly what is going to be done in this section.

6.3.1. OAuth Server and SSL/TLS connection

This test will involve two HTTP requests. The first request will use the user credentials stored in the OAuth server code. These credentials are not overly complex as the main purpose is to understand the Auth flow for this mock server (*client_id, client_secret*). The HTTP request will initially use the POST method to generate a token for the provided credentials. The second request will then use this token to verify access to a specific URL. For this specific test, SSL/TLS implementation will not be used. Three test cases are being implemented in these terms:

- Valid Request: First, the POST request will be made to the *generate_token* instance of the OAuth server. Then, the previously generated Token will be used to make a POST request to the *check_token* instance.

- Reusing Token: This test case showcases that the same Token can not be used twice

- Forbiden URL: This test case demonstrates that the OAuth server can restrict access to specific resources for certain users.

The BASH script to perform this test (**OauthTest.sh**) is located in the same GitHub repository as the rest of the project files [22] specifically in the folder named */Testing*. The results of the performed tests are as follows:

```
==============================
TEST CASE: CORRECT REQUEST
Generating token...
Response:
{"access_token":"0e67270825f81ba8d2d4a4a694c956ad"} - 200
Checking token...
Using Token: 0e67270825f81ba8d2d4a4a694c956ad
Response:
{"message":"Access␣granted␣to␣
   https://192.168.1.92:2023/SimpleEdge"} - 200
==============================
TEST CASE: REUSING TOKEN
Checking token...
Using Token: 0e67270825f81ba8d2d4a4a694c956ad
Response:
{"error":"Invalid␣token"} - 401
==============================
TEST CASE: Forbidden URL
Generating token...
Response:
{"access_token":"02255853da4be15fdde0c30947dbebea"} - 200
Checking token...
Using Token: 02255853da4be15fdde0c30947dbebea
Response:
{"error":"Access␣denied.␣Insufficient␣priviledges␣or␣invalid␣URL␣
   parameters."} - 403
```

The aim of this test was to illustrate the shortness of responses given by the developed OAuth server and its different instances. However, as this initial version did not use SSL/TLS, the credentials used could be stolen if the traffic is captured and inspected, as it is not encrypted. The following test will demonstrate this vulnerability and suggest a possible solution.

Traffic Capture: No SSL/TLS

By utilising traffic capturing tools like Wireshark and applying its filters to focus solely on traffic to and from the OAuth server IP adress, as well as filtering by HTTP traffic, one can inspect the content of the HTTP messages. This is illustrated in the figure 6.2 where a Wireshark capture was taken during the execution of the **OauthTest.sh** script. It is shown that the credentials and tokens could be stolen, which is highly undesirable.

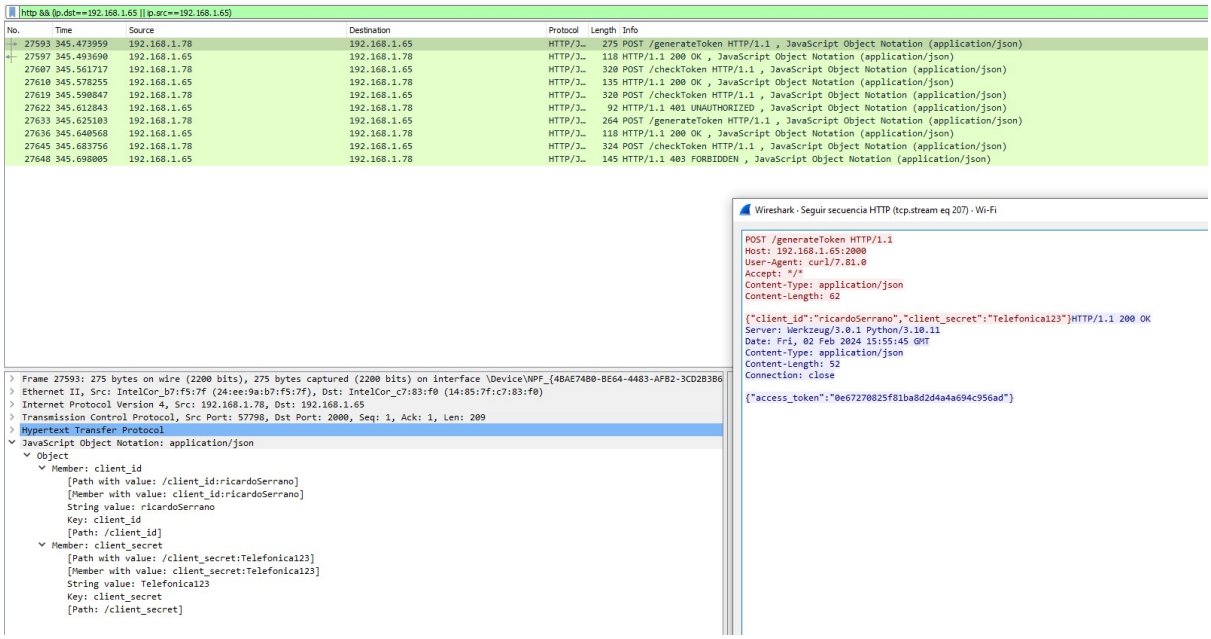

Figure 6.2: Wireshark capture regarding OAuth test. No encryption used

Traffic Capture: SSL/TLS

To solve this issue, encrypting the traffic is recommended. It is advisable to use different certificates for this service and server than those used for the Edge Selection API service. However, the same process described in section 5.4 can be followed to generate the private keys and the server certificates. The CA certificate is not required in this case as the server is always trusted. Nonetheless, an extra authentication layer is always beneficial. Instead of using a CA certificate, the parameter **-k** can be used to force Curl trusting the server and perform the request.

This can be done by incorporating the following lines to the *main* body of the Flask application:

```
context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)
context.load_cert_chain('cert.pem','cert-key.pem')
app.run(host='192.168.1.65', port=2000, ssl_context=context)
```

After generating and incorporating the private and public key pair into the Flask OAuth server, it is evident that the traffic between the client and the OAuth server is no longer HTTP but TLS. The TLS handshake, as described in the design chapter and shown in figure 4.6, can be observed, as well as the application data that corresponds to the encrypted HTTP requests. This is illustrated in the Wireshark capture from figure 6.3

6.3.2. Virtual Directory

To comprehend the necessary requests that need to be made to the Virtual Directory and the shortage of responses received by this system, this test proves itself valuable not only in this stage of the project but also in the implementation of the end to end service. In these terms, the script **VirtualDirectory.sh** have been developed and is available in the */Testing* folder of GitHub repository created for this project [22].

The Virtual Directory, is able to make a response for the imput parameters described in table 6.1. As previously mentioned, only the IP address (in particular the mobile IP address) and the

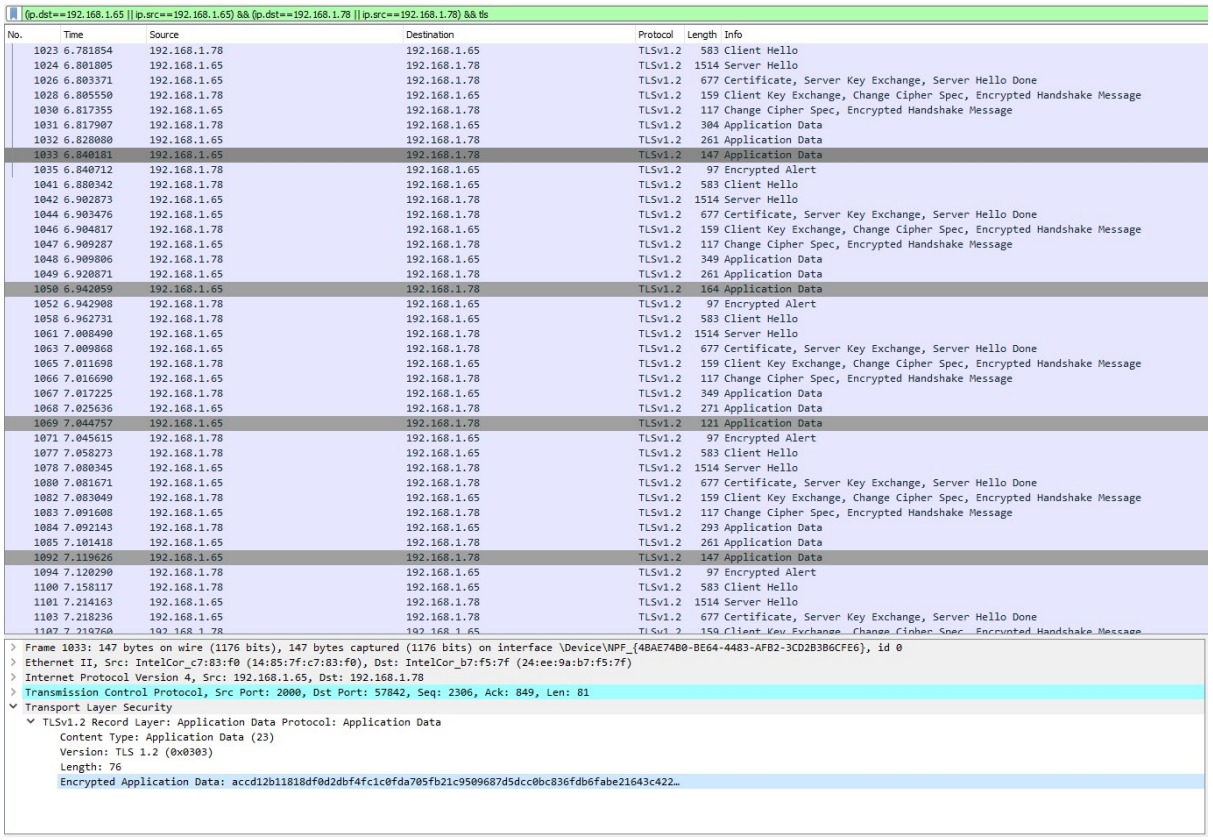

Figure 6.3: Wireshark capture regarding OAuth test. TLS protocol used

MSISDN are currently required as input parameters. The list of attributes that can be obtained from the Virtual Directory cannot be disclosed due to confidentiality concerns. However, it is necessary to verify the UPF that the user is connected to. This is stored in NAS-ID attribute from the Virtual Directory user profile.

Parameter	**Format**	**Type**	**Description**
MSISDN	String	Optional	User MSISDN
IpAddr	String	Optional	User fixed IP, v4, v6 or IP:PORT
IpBAM	String	Optional	User mobile IP, v4, v6 or IP:PORT
IMSI	String	Optional	User IMSI attribute
IMEI	String	Optional	User IMEI attribute
Attributes	String	Optional	List of cero or more Virtual Directory elements to be queried

Table 6.1: Virtual Directory allowed input parameters

Executing the **VirtualDirectory.sh** script, the responses are as follows. It can be seen that there are three parameters (*operationResultCode*, *nodeCode* and *errorCode*). The operation and the response is correct always that the *operationResultCode* is 00. If the *operationResultCode* is 02 it means that there is something wrong regarding the parsed parameters (if *errorCode* is 120 there was no parameters in the request and if 121 the requested parameters are unavailable). Finally, if the *operationResultCode* is 03 this means that the requested attributed are not available in the Virtual Directory. The *nodeCode* does not imply error in this case:

```
TEST CASE: No Parameters
Response:
```

```
{"Attributes": [], "operationResultCode": "02", "nodeCode": "01",
   "errorCode": "120"}
==============================
TEST CASE: Request NAS-ID using MSISDN
Response:
{"Attributes": [{"NAS-id": "DCBESP2_vUGW01"}],
   "operationResultCode": "00"}
==============================
TEST CASE: Request NAS-ID using IP
Response:
{"Attributes": [{"NAS-id": "DCBESP2_vUGW01"}],
   "operationResultCode": "00"}
==============================
TEST CASE: Request NAS-ID using wrong MSISDN
Response:
{"Attributes": [], "operationResultCode": "02", "nodeCode": "01",
   "errorCode": "121"}
==============================
TEST CASE: Unavailable attributes
Response:
{"Attributes": [], "operationResultCode": "03", "nodeCode": "01",
   "errorCode": "132"}
```

6.4. End to end Testing

This section demonstrates the final API operation by performing an end-to-end test. The test involves making multiple requests to the OAuth Server to obtain authorization tokens, which are then used to make various requests to the API to showcase its behaviour.

The tests were conducted using a BASH script called **e2eTest.sh**, which can be found in the */testing* folder of the GitHub repository used for this project [22]. This follows the same procedure as the other tests in this section. This script, first retrieves all possible EdgeSites using a token from the user named "ricardoSerrano." Subsequently, it demonstrates accessing the *SimpleEdge* endpoint with a specified IP address and MSISDN using tokens from different users ("user1" and "user2"). The script attempts forbidden access to the *adminEdgeSites* endpoint, simulating a scenario where regular users should not have access. It also tests a bad request by providing an invalid MSISDN. Finally, it showcases an example PUT request for an admin user ("ricardoSerrano") to update data on the *adminEdgeSites* endpoint and a subsequent GET request to retrieve the updated information.

```
For ricardoSerrano making API request with token::
   5a10e6c319bc78f07b586cadd391c8a3
API Response:
{
  "Telefonica": [
    "TE_alco_231",
    "TE_bilb_452",
    "TE_val_565",
    "TE_bcn_129",
    "TE_sev_675"
```

```
  ]
}
==============================
For user1 Making API request (IP ADDR) with token::
   b4aa40dd605bff19bc19c955e6669567
API Response:
{
  "Ern": "TE_sev_675",
  "Provider": "Telefonica"
}
==============================
For user2 making API request (MSISDN) with token::
   0c081e81c494c194fd034e0166a769db
API Response:
{
  "Ern": "TE_sev_675",
  "Provider": "Telefonica"
}
==============================
For user2 making a forbiden request with token::
   54d9879ba06802f6b8b0849a6c1d5a62
API Response:
{
  "403": {
    "code": 403,
    "error": "Access␣denied.␣Insufficient␣privileges␣or␣invalid␣
       URL␣parameters."
  }
}
==============================
For user2 making a bad request with token::
   e226dfd1cc4a89ae271c27d341a946da
API Response:
{
  "404": {
    "code": "404",
    "error": "DATAGRID␣INTERNAL␣ERROR:␣Device␣not␣found"
  }
}
==============================
For ricardoSerrano making API request with token::
   716486aa6cb4c4792cf03f331369d0ac
PUT Admin API Response:
{
  "message": "DataSeed␣updated␣successfully."
}
==============================
For ricardoSerrano making API request with token::
   aea54ca98b06055771c8d2a02bcbed44
GET Admin API Response:
{
  "data": [
```

```
    {
      "EdgeNode": "TE_sev_675",
      "EdgeNode␣IP": "23.127.0.4",
      "NAS-id": "DCBESP2_vUGW01"
    },
    {
      "EdgeNode": "TE_alco_231",
      "EdgeNode␣IP": "23.127.0.1",
      "NAS-id": "DCBTYB2_vUGW01"
    },
    {
      "EdgeNode": "TE_bcn_129",
      "EdgeNode␣IP": "23.127.0.3",
      "NAS-id": "DCVCAR2_vUGW01"
    }
  ]
}
```

This way, the **e2eTest.sh** script prints the API responses for each scenario, providing a comprehensive overview of the API's behavior under various conditions. It emphasizes authentication, successful requests, forbidden access, and error handling to ensure a thorough assessment of the API's functionality.

6.5. Conclusions

The main purpose of this chapter was to validate the work done in the previous sections, particularly the Edge Selection API implementation. This was achieved throughout the different sections, but especially in section 6.4: End to end testing. Additionally, demonstrating how the different working systems were supposed to interact with the main API function was important for justifying the implementation decisions made in earlier chapters.

The BASH scripts were developed to showcase the user stories of this API, including how information is consumed and the expected requests and responses from the various systems in the architecture.

The importance of using encryption protocols to protect sensitive information, such as tokens or user credentials, was also demonstrated. As it was shown, if a request to the authorization system is made without TLS/SSL protection, an attacker only needs to be connected to the same network as the client to steal the user's credentials using traffic analysis tools such as Wireshark. The attacker can then authenticate to the API using the stolen credentials.

Regarding the functioning of the Edge Selection API, no showstoppers, bugs, or errors were detected. It is especially important to test the APIs in at least two operator architectures in CAMARA. In conclusion, not only this chapter but also the entire project has demonstrated the feasibility of implementing the Edge Selection API in Telefónica Spain's architecture.

7

Conclusions and future work

7.1. Conclusions

The development in the field of 5th generation of mobile networks will allow high computing capabilities to be brought closer and closer to end users. The Telco Edge Cloud appears as a new paradigm within these 5G networks with the aim of enabling the use of the Cloud computing at an application level as never known before. Not only does the Telco Edge bring the Cloud closer to the user in a literal way, allowing applications that have traditionally run inside terminals to now do so in a much more advanced computing nodes, but it also allows latency to be reduced to levels never before seen in this type of technology, which gives more weight to this type of use cases, for example.

Having an application fully deployed in the Cloud (cloud-native) and their front-end or app client running on the terminal performing the most computationally intensive operations in the Cloud allows the device's processor load to be increasingly lower, which translates into battery savings on the part of the device. Thanks to the confluence of low latency and high performance computational processing offered by the Telco Edge Cloud, this and many other interesting use cases for the technological development of the telecommunications industry are encouraged.

Another interesting use case that arises is 3D processing. VR and XR applications require a high degree of processing as they mostly use a large amount of 3D images that are difficult to process efficiently on a mobile device, and if these images have to be processed in real time, the difficulty is increased. The Telco Edge offers a very interesting solution to deal with this issue.

In line with the new 5G philosophy where telecommunications networks need to be able to expose capabilities and information through standardised interfaces harmonised across industry forums, the Telco Edge is no exception. To ensure minimum latency, it is key that the device is able to know which is the most optimal edge site to connect to. This capability has to be offered by the Telco Edge provider through this standard interfaces.

As far as the Telco Edge is concerned, it is also necessary that the lifecycle management of applications is subject to standard interfaces. That is, a developer must be able to use these interfaces to manage the different images or instances that make up the application being deployed on the Telco Edge.

Throughout this project, these gaps in the state of the art and the need to fill them have been presented, which was also the main functional objective of this project as far as the industry was

concerned. The Edge Selection API needed to be implemented in an operator and the Lifecycle Management API of the applications needed to be defined in the standardisation forums.

The implementation of the Edge Selection API has been a challenge in terms of analysing the functional architecture of the operator as well as understanding the location of the data necessary for the operation of this API. Not only has the functional component of the API been a challenge, but it has also been challenging to develop all the accompanying securitisation elements, which implies an understanding of all these systems in order to be able to develop them. In any case, it is clear that the objectives and goals proposed in the introductory chapters have been met with regard to this implementation, as the understanding of the elements of the architecture that make up this solution as well as the security protocols expected of a development such as the one proposed are demonstrated throughout this document.

With regard to the lifecycle management of applications. It was necessary to understand and define the different methods necessary for the deployment and management of applications by a developer and to generate the relevant definition in order to fill the gap that existed in the standards. Therefore, the objectives proposed for this part of the project are also understood and accomplished.

In summary, the rapid evolution of the telecommunications landscape, driven by the advent of 5G networks and the emergence of Telco Edge Cloud, necessitates the development of robust APIs to cater to the dynamic demands of cutting-edge applications such as XR and cloud-native solutions. The paradigm shift towards applications that rely on high-performance edge computing calls for sophisticated decision-making in real-time scenarios. To address this need, the implementation of APIs becomes imperative. The Edge Selection API plays a pivotal role in deciding optimal deployment sites for application workloads, bringing computational capabilities closer to end-users. Simultaneously, the Lifecycle Management API facilitates the seamless deployment and management of applications, ensuring adaptability to the dynamic nature of modern network architectures. The significance of these APIs cannot be overstated, as they enable the realization of cloud-native applications, reduce device processor loads, and contribute to enhanced user experiences. As telecommunications networks continue to evolve, these APIs stand at the forefront, providing the foundation for efficient, real-time decision-making and deployment in the era of Telco Edge Cloud and 5G networks.

7.2. Future work

In general, it can be referred to the introductory chapter where the objectives and expected deliverables were proposed, as all these objectives are considered as fulfilled and thus this project is an interesting contribution to the telecommunications industry. However, there is still a long way to go before the Telco Edge is fully adopted and even defined, so in terms of future work it is proposed the following activities:

Implementation of Lifecycle Management API and other Telco Edge APIs

Once the Lifecycle Management API of the applications has been defined, as it is the case, it would be necessary to validate this definition by implementing this API in a real architecture. This would be a big step towards putting the Telco Edge technology into production if the full implementation of the Edge Cloud APIs is done. The proposal would be to first implement the Edge Selection and LcM APIs so that they work in confluence, as they are the least complex at an operational level, and then, as a last implementation, also include the Traffic Influence APIs, as they are the most complex at a technical level. In this way, there could be advances in this paradigm to begin with a minimum viable product that would later evolve.

Improvement of the Edge Selection API

Currently, the Edge Selection API (or Simple Edge Discovery by its name in CAMARA) simply returns the name in alphanumeric characters of the most optimal Edge Site to which a user or device should connect, once the user have that name it can be applied to the Cloud Edge platform to make the connection to the Edge Site that corresponds to it. The proposal is that the Edge Selection API directly returns the URL of the application to which the user must connect, which is hosted in the most optimal Edge Site.

NEF integration with Telco Edge

As previously mentioned, the NEF is a significant advancement in showcasing the telecommunications network's capabilities. The integration of Telco Edge mechanisms in synergy with the NEF would enable telecom operators to fully deploy the Telco Edge in current and future generations of mobile networks. Additionally, NEF could enhance the implementation of Traffic Influence APIs, as discussed in previous chapters. Furthermore, the enhancements of network events and notifications to be exposed by NEF prove to be a significant improvement in terms of the deployment of Telco Edge architectures and solutions, especially in terms of user mobility. The implementation of Traffic Correlations APIs is also an interesting piece of future work to be carried out in regard to NEF.

Glossary

- **AMF**: Access and Mobility Management function
- **API**: Application Programming Interface
- **CA**: Certification Authority
- **CSR**: Certificate Signing Request
- **DoS**: Deny of Service
- **GSMA**: Global System for Mobile Communications Association
- **IdP**: Identity Provider
- **IMS**: IP Multimedia Subsystem
- **HTTP(S)**: HyperText Transfer Protocol (Secure)
- **KPI**: Key Performance Indicator
- **NEF**: Network Exposure Function
- **MSISDN**: Mobile Station Integrated Services Digital Network
- **PGW**: Packet Gateway
- **RTT**: Round Trip Time
- **SDF**: Service Delivery Framework
- **SGW**: Serving Gateway
- **SMF**: Session Management Function
- **SSL**: Security Socket Layer
- **TLS**: Transport Layer Security
- **UE**: User Equipment
- **UNI**: User to Network Interface
- **UPF**: User Plane Function
- **VM**: Virtual Machine
- **VR**: Virtual Reality
- **XR**: eXtended Reality
- **3GPP**: 3rd Generation Partnership Project

Bibliography

[1] Steven Chiu. Mec in 5g networks. https://stevenchiu30801.github.io/2020/10/15/MEC-in-5G-Networks/.
Accessed: 2024-January.

[2] T. Dierks; E. Rescorla. Rfc 5246. the transport layer security (tls) protocol version 1.2. *Internet Engineering Task Force (IETF)*, 2008 (updated 2021).

[3] 3GPP TS 23.501. Technical specification group services and system aspects; system architecture for the 5g system (5gs). version18.3.0., 2023-September.

[4] F. Mazzenga F. Vatalaro G. Ciccarella, R. Giuliano and A. Vizzarri. Edge cloud computing in telecommunications: Case studies on performance improvement and tco saving. *Fourth International Conference on Fog and Mobile Edge Computing (FMEC)*, 2019-June.

[5] GSMA. Telco edge cloud value and achievements. 2022-March.

[6] GSMA. Operator platform telco edge requirements version 2.0. 2022-April.

[7] GitHub. CAMARA-QualityOnDemand. Qualityondemand. https://github.com/camaraproject/QualityOnDemand.
Accessed: 2024-January.

[8] GitHub. CAMARA-EdgeCloud. Trafficinfluence api. https://github.com/camaraproject/EdgeCloud/tree/main/code/API_definitions/Traffic%20Influence.
Accessed: 2024-January.

[9] OAuth2.0 Working Group. https://oauth.net/2/.
Accessed: 2023-November.

[10] GSMA TS.43. Service entitlement configuration. version 8.0., 2022-January.

[11] OpenSSL. https://www.openssl.org/.
Accessed: 2024-January.

[12] P. Martínez-Julia J. Baliosian, L. M. Contreras and J. Serrat. An efficient algorithm for fast service edge selection in cloud-based telco networks. *IEEE Communications Magazine*, vol. 59, no. 10, pp. 34-40, 2021-October.

[13] 3GPP TS 29.503. 5g system; unified data management services; stage 3. version18.3.0., 2023-September.

[14] ETSI TS 129 522. 5g system; network exposure function northbound apis; stage 3, 2019-April.

[15] 3GPP TS 29.522. Technical specification group core network and terminals; 5g system; network exposure function northbound apis; stage 3 (release 18), 2023-December.

[16] 3GPP TS 29.541. 5g system; network exposure (ne) function services for non-ip data delivery (nidd) and short message services (sms); stage 3 (release 18), 2023-July.

[17] GitHub. CAMARA-EdgeCloud. Simpleedgediscovery api. https://github.com/camaraproject/EdgeCloud/blob/main/code/API_definitions/Discovery/simple_edge_discovery.yaml.
Accessed: 2024-January.

[18] GitHub. CAMARA-EdgeCloud. Edgecloudlcm api. https://github.com/camaraproject/EdgeCloud/blob/main/code/API_definitions/EdgeCloud_LcM.yaml.
Accessed: 2024-January.

[19] GitHub. CAMARA-EdgeCloud. Pr-154. https://github.com/camaraproject/EdgeCloud/pull/154.
Accessed: 2024-January.

[20] GitHub. CAMARA-Guidelines. https://github.com/camaraproject/Commonalities/blob/main/documentation/API-design-guidelines.md.
Accessed: 2023-November.

[21] Learn.Microsoft. https://learn.microsoft.com/en-us/windows/win32/secauthn/transport-layer-security-protocol.
Accessed: 2024-January.

[22] GitHub. RicardoSerr. Edge selection. https://github.com/RicardoSerr/EdgeSelection.
Accessed: 2024-January.

[23] R.L. Rivest A. Shamir and L. Adleman. A method for obtaining digitalsignatures and public-key cryptosystems. *Communications of the ACM*, 1978-February.

[24] WikiDebian. https://wiki.debian.org/Self-Signed_Certificate.
Accessed: 2024-January.

A

Edge Cloud data model relevant definitions

1) **AppId**

- Description: A globally unique identifier associated with the application. Generated by the Operator Platform (OP) when the application is submitted over the Network-based Interface (NBI).
- Type: String
- Format: UUID

2) **AppInstanceId**

- Description: A globally unique identifier associated with a running instance of an application. Generated by the Operator Platform (OP).
- Type: String
- Format: UUID

3) **AppInstanceInfo**

- Description: Information about the application instance.
- Properties:
  - appInstanceId: Reference to the AppInstanceId.
  - edgeCloudsiteName: Name of the Edge Cloud site where the application instance is running.
  - uri: Uniform Resource Identifier (URI) identifying the endpoint within an Edge Cloud site where the user equipment may connect to the selected application instance.
  - status: Status of the application instance (ready, instantiating, failed, terminating, unknown).

4) **AppInstantiation**

- Description: Attributes containing information about where to instantiate a given application.
- Properties:
  - appId: Reference to the AppId.

- edgeCloudsiteName: Name of the Edge Cloud site where the application should be instantiated.
- regionId: Human-readable name of the geographical zone of the Edge Cloud site. Defined by the OP.

5) AppManifest

- Description: Contains information about the application to be instantiated.
- Properties:
  - name: Name of the application.
  - version: Application version information.
  - virtType: Virtualization type (VM, CONTAINER, HELM).
  - fileName: Name of the file with the extension (e.g., zip, tar.gz, etc.).
  - repository: Details about the artifact repository.
    - type: Type of repository (PUBLICREPO, PRIVATEREPO).
    - url: URL of the repository.
    - userName: Username to access the artifact repository.
    - password: Password to access the artifact repository.
    - token: Authorization token to access the artifact repository.
  - componentSpec: Details about compute, networking, and storage requirements for each component of the application.

6) ComponentSpec

- Description: Details about compute, networking, and storage requirements for each component of the application.
- Properties:
  - componentName: Name of the component; must be unique within an application.
  - operatingSystem: Reference to the OperatingSystem.
  - cpuArchitecture: CPU Instruction Set Architecture (ISA) E.g., Intel, Arm, etc.
  - networkInterfaces: Array of networking interface details.
  - numOfInstances: Number of component instances to be launched.
  - restartPolicy: How the platform shall handle component failure (RESTART-POLICY-ALWAYS, RESTART-POLICY-NEVER).

7) CpuArchType

- Description: CPU Instruction Set Architecture (ISA), e.g., Intel, Arm, etc.
- Type: String
- Enum: x8664, arm64

8) EdgeCloudsiteName

- Description: Identifier for an edge cloud site in the operator domain.
- Type: String

9) EdgeCloudsiteDetails

- Description: Details about an Edge Cloud site.
- Properties:
  - edgeCloudsiteName: Reference to the EdgeCloudsiteName.

- status: Status of the Edge Cloud site (active, inactive, unknown).
- region: Reference to the RegionId.

10) ErrorInfo

- Description: Information about an error response.
- Properties:
  - status: HTTP status code returned with the error response.
  - code: Code given to this error.
  - message: Detailed error description.

11) OperatingSystem

- Description: Details about an operating system.
- Properties:
  - architecture: CPU architecture (x86-64, x86).
  - family: Operating system family (RHEL, UBUNTU, COREOS, WINDOWS, OTHER).
  - version: Operating system version.
  - license: Type of operating system license (OS-LICENSE-TYPE-FREE, OS-LICENSE-TYPE-ON-DEMAND, OTHER).

12) RegionId

- Description: Human-readable name of the geographical zone of the Edge Cloud site. Defined by the Operator Platform.
- Type: String

13) Status

- Description: Status of an entity (default is 'unknown').
- Type: String
- Enum: active, inactive, unknown

Madrid, February 2024
Master Thesis

Ricardo Serrano Gutiérrez
Graduate in Telecommunications Technology and Services Engineering