eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

MASTER'S DEGREE IN TELECOMMUNICATION ENGINEERING

MASTER'S THESIS

A NOVEL EDGE COMPUTING FRAMEWORK FOR AUTOMOTIVE DATA PROCESSING

Student: Serón Esnal, Mikel
Director: Astorga Burgo, Jasone
Co-director: Huarte Arrayago, Maider
Director's Department : Communications Engineering

Bilbao, September 2022

Abstract Laburpena Resumen

In recent decades, the mobile cellular communications market has seen much activity. Telecommunications operators are undergoing a digital transition to give users a real-time, on-demand experience. The fifth generation of mobile networks, or 5G, marks the start of the digitization of society's interconnection. The telecommunications sector sees the next generation of mobile networks, as a critical growth factor for innovation and automation, resulting in increased commercial and economic competitiveness.

5G systems are planned to handle a wide range of application scenarios and use cases (for example, automotive, public services, smart cities, and medical care), each with its own needs. In this context, edge computing is considered one of the essential features in next-generation networks since it will allow for a flood of latency, throughput, and data-sensitive edge-native applications. Edge application developers need edge infrastructure to host the application workload and network connectivity protocols to link application users to the nearest edge where the application workload is located.

At the same time, in the vehicular sector, the growing amount of car-generated data is opening new possibilities for the automotive sector. The use of automotive data is strategic for industry players. New paradigms in car messaging and data gathering for establishing new opportunities are vital for expanding the ecosystem of connected and autonomous mobility services.

In this context, this project aims to study the current state-of-the-art in edge computing, analyze the existing edge computing frameworks oriented to connected vehicle use case, and design a platform that will fit all the requirements of car-generated data management. The captured data will be processed on consumer demand in an edge server and delivered to the users through a cloud platform, being possible to filter the data between geographical positions.

Keywords: 5G, Automotive data, CCAM, Edge Computing, Kubernetes, MEC, V2X, VNF.

Azken hamarkadetan, komunikazio zelularren merkatuak mugimendu handia bizi izan du. Telekomunikazio operadoreak trantsizio digital batean murgiltzen ari dira erabiltzaileei denbora errealean lan egiten dituzten esperientziak eskaini ahal izateko. Sare mugikorren bosgarren belaunaldiak, edo 5G-k, gizartearen interkonexioaren digitalizazioaren hasiera adierazten du. Telekomunikazioen sektoreak sare mugikorren hurrengo belaunaldia ikusten du berrikuntzaren eta automatizazioaren aldeko faktore kritiko gisa, lehiakortasun komertzial eta ekonomikoa handituko duelarik.

Bosgarren belaunaldiko sistemak aplikazio eta erabilera kasu ugari (automobilgintza, zerbitzu publikoak, hiri adimentsuak eta osasun arreta zerbitzuak esaterako) kudeatzeko diseinatuak izan dira. Testuinguru honetan, muturreko konputazioa edo *edge computing*-a funtsezko ezaugarri bat da, izan ere latentzia gutxiko, errendimendu handiko edo denbora errealeko datuen inguruko aplikazioak ahalbidetzen dituelako. Edge computing aplikazioen garatzaileek edge azpiegiturak behar dituzte aplikazioen lan-kargak eta sare protokoloak ezartzeko, erabiltzaileak hurbilen dagoen edge-era konektatzeko, non bertan aplikazioa kokatuta egongo den.

Aldi berean, automobilgintza sektorean, autoek sortzen dituzten datuak gero eta anitzagoak izateak, aukera berriak ireki dizkio automozioaren sektoreari. Kotxean sortutako informazioaren erabilera estrategikoa da industriako enpresentzat. Auto mezularitzan eta datuen bilketan paradigma berriak izatea ezinbestekoa da aukera berriak ezarri eta mugikortasun-zerbitzuen ekosistema zabaldu ahal izateko.

Testuinguru honetan , proiektu honek *Edge computing*-eko artearen egoera aztertzea du helburu, ibilgailuei zuzendutako *edge computing* egitura edo *framework*-ak aztertuz, eta autoek sortutako datuak kudeatzeko sistema bat diseinatzea. Lortutako datuak kontsumitzaileen eskaripean tratatuko dira *edge* zerbitzari batean eta erabiltzaileei eskainiko zaizkie *cloud* plataforma baten bidez, datuak horiek kokapen geografikoaren arabera iragazi ahalko direlarik.

Gako-hitzak: 5G, Autoen datuak, CCAM, Ertz konputazioa, Kubernetes, MEC, V2X. VNF.

En las últimas décadas, el mercado de las comunicaciones móviles celulares ha experimentado una gran actividad. Los operadores de telecomunicaciones están llevando a cabo una transición digital para ofrecer a los usuarios una experiencia en tiempo real. La quinta generación de redes móviles, o 5G, marca el inicio de la digitalización de la interconexión de la sociedad. El sector de las telecomunicaciones considera que la próxima generación de redes móviles es un factor de crecimiento crítico para la innovación y la automatización, lo que se traduce en una mayor competitividad comercial y económica.

Los sistemas 5G están planificados para manejar una amplia gama de escenarios de aplicación y casos de uso (por ejemplo, automoción, servicios públicos, ciudades inteligentes y atención médica), cada uno con sus propias necesidades. En este contexto, la computación de borde o *edge computing* se considera una de las características esenciales en las redes de próxima generación, ya que permitirá una avalancha de aplicaciones *edge* nativas aportando una latencia mínima y una gran afluencia de datos. Los desarrolladores de aplicaciones de borde necesitan una infraestructura edge para alojar la carga de trabajo de la aplicación y los protocolos de conectividad de red para enlazar a los usuarios de la aplicación con el edge más cercano donde se aloja la aplicación.

Por otra parte, la creciente cantidad de datos generados por los automóviles está abriendo nuevas posibilidades para el sector de la automoción. El uso de los datos del automóvil es estratégico para los agentes del sector. Los nuevos paradigmas en la mensajería dirigida al automóvil y la recopilación de datos son vitales para establecer nuevas oportunidades y ampliar el ecosistema de servicios de movilidad conectados y autónomos.

En este contexto, este proyecto tiene como objetivo estudiar el estado actual de la técnica en *edge computing*, analizar los marcos o frameworks existentes de *edge computing* orientados al caso de uso del vehículo conectado, y diseñar una plataforma que se ajuste a todos los requisitos de la gestión de datos generados por el automóvil. Los datos capturados serán procesados a demanda del consumidor en un servidor *edge* y entregados a los usuarios a través de una plataforma en la nube, siendo posible filtrar los datos entre posiciones geográficas.

Palabras Clave: 5G, CCAM, Computación en el borde, Datos de automóviles, Kubernetes, MEC, V2X, VNF.

Contents

List of Figures

List of Tables

Listings

Terms and Abbreviations

Terms

Terms that will be common and repeatedly employed during the work

Edge Computing The delivery of computing capabilities to the logical extremes of a network in order to improve the performance, operating cost and reliability of applications and services. By shortening the distance between devices and the cloud resources that serve them, and also reducing network hops, edge computing mitigates the latency and bandwidth constraints of today's Internet, ushering in new classes of applications. In practical terms, this means distributing new resources and software stacks along the path between today's centralized data centers and the increasingly large number of devices in the field, concentrated, in particular, but not exclusively, in close proximity to the last mile network, on both the infrastructure and device sides.

Dataflow Unique flow/stream of produced data that is shared by Sensors and Devices (S&D) through a Pipeline in a MEC.

Data-type Type of data that S&D shares and that a Pipeline in a MEC accepts and handles.

Pipeline Group of Modules (containers) running in a MEC to handle a specific data-type. Each module can process the data received by subscribing to a queue/topic and generate output data in a new queue/topic into the MEC's message broker.

Module Container running in a MEC that handles a specific data-type or group of data-type (e.g. C-ITS messages).

Third-party application Application/service lying outside the platform, which acts as a third-party, consuming the available data, and optionally pushing events to the platform (e.g., a CCAM application).

Abbreviations

3GPP 3rd Generation Partnership Project

5GPPP 5G Infrastructure Public Private Partnership

AAA Authentication, Authorization and Accounting

AI Artificial Intelligence

AMQP Advanced Message Queuing Protocol

AP Access Point

API Application Programming Interface

AR Augmented Reality

AWS Amazon Web Services

BSS Business Support System

CAM Connected and Automated Mobility

CAPEX Capital Expenditures

CCAM Cooperative, Connected and Automated Mobility

C-ITS Cooperative Intelligent Transport System

CNCF Cloud Native Computing Foundation

CNI Container Network Interface

COTS Commercial-Off-The-Shelf

CPU Central Processing Unit

C-V2X Cellular Vehicle-to-everything

DCAE Data Collection, Analytics and Events

eMBB Enhanced Mobile Broadband

EU European Union

ELK Elasticsearch, Logstash and Kibana

ETSI European Telecommunications Standards Institute

gNB gNodeB

GPU Graphics Processing Unit

GUI Graphical User Interface

HTTP Hypertext Transfer Protocol

IaaS Infrastructure-as-a-Service

IoE Internet of Everything

IoT Internet of Things

ISG Industry Specification Group

ISP Internet Service Provider

ITU International Telecommunication Union

KNF Kubernetes-based Network Function

KNFD Kubernetes Network Function Descriptor

KPI Key Performance Indicator

LAN Local Area Network

LTE Long Term Evolution

LF Linux Foundation

LIDAR Laser Imaging Detection and Ranging

MANO Management and Orchestration

MEC Multi-access Edge Computing

MQTT Message Queuing Telemetry Transport

mTMC Massive Machine-Type Communication

NF Network Function

NFV Network Functions Virtualization

NFVI Network Functions Virtualization Infrastructure

NFVO Network Function Virtualization Orchestrator

NR New Radio

NS Network Service

NSD Network Service Descriptor

OAS OpenAPI Specification

OBU On-Board-Unit

OEM Original Equipment Manufacturer

ONAP Open Network Automation Platform

ONOS Open Network Operating System

OPEX Operating Expenditures

OS Operating System

OSI Open Systems Interconnection

OSM Open Source MANO

OSS Operations Support System

PaaS Platform-as-a-Service

QoS Quality of Service

RAM Random Access Memory

RAN Radio Access Network

ROI Region Of Interest

RSI Road Side Infraestructure

RSU Road Site Unit

SaaS Software-as-a-Service

SDN Software Defined Networking

SLA Service Level Agreement

SME Small and Medium-sized Enterprise

S&D Sensors and Devices

TCP Transmission Control Protocol

UE User Equipment

URLLC Ultra-Reliable Low-Latency Communication

V2I Vehicle-to-Infrastructure

V2N Vehicle-to-Network

V2P Vehicle-to-Pedestrians

V2V Vehicle-to-Vehicle

V2X Vehicle-to-Everything

VM Virtual Machine

VNF Virtual Network Function

VNFM Virtual Network Function Manager

VR Virtual Reality

WAN Wide Area Network

WLAN Wireless Local Area Network

WP Work Package

1. Introduction

Radio technology has enabled wireless communications over longer distances and ever-higher capabilities for more than a century. Guillermo Marconi made the first radio transmission at the beginning of the 20th century, capable of sending Morse code through radio waves up to a half-mile, laying the groundwork for wireless communication [1]. Wireless mobile technologies were explored and tested before the turn of the century, but it was not until the early 1970s that cellular networks and mobile phones were mass-produced. Since then, as technology has advanced, the options have expanded, as have user requirements and expectations. We can now connect with anybody, wherever globally, thanks to more than seven billion mobile smartphones [2] [3].

With the introduction of 5G, the telecommunications sector is undergoing a revolution compared to previous generations of mobile communications. The networks will not only provide connection but will also give end-users a differentiated quality of experience. 5G offers 1000 times more bandwidth, 50-100 times reduced latency, and 10-20 times faster download speeds [4].

Moreover, the number of devices connected to the internet, as well as the volume of data created by those devices and consumed by enterprises, is outpacing traditional data center infrastructures. According to Gartner, [5], 75 percent of enterprise-generated data will be created outside of centralized data centers by 2025. The thought of transmitting so much data in situations where time or interruption is critical places an enormous demand on the global internet, which is already prone to congestion and disruption.

As a result, the attention has been switched away from the central data center toward the logical edge of the infrastructure, relocating storage and computing resources from the data center to the point where data is generated. The idea is simple: if it is not possible to bring the data closer to the data center, move the data center closer to the data, making possible to address the limitations of centralized systems [6].

Edge computing, an emerging distributed computing paradigm, is all a matter of location. It may operate many devices across a much smaller and more efficient Local Area Network (LAN) by placing servers and storage where the data is generated, resulting in nearly non-existent latency and congestion. Local storage captures and secures raw data, while local servers can execute critical edge analytics and pre-process and reduce the data in real-time to make decisions before transferring results or just forward essential data to the cloud or central data center [7].

5G networks complemented by edge computing are enabling low latency communication and related use cases [8]. They are two significantly linked technologies, and both are designed to boost application performance and enable massive volumes of data to be processed in real-time.

In the same way, the telecommunication industry has been leveraging edge comput-

ing to solve critical network challenges, and the automotive industry is also looking to edge computing as a framework for ensuring connectivity. The sector of vehicle manufacturing is on a path where vehicles are continuously becoming more aware of their environment due to the addition of various types of integrated sensors. They are continuously collecting data and performing a variety of functions, i.e., monitoring road conditions and engaging the brakes when a hazard is detected. These kinds of functions produce such high data volumes that must be analyzed in real-time, and they must occur at the edge rather than in a central site. Connecting autonomous vehicles to the edge can increase safety, efficiency, reduce accidents, and cut traffic congestion [9].

The marriage of 5G and edge computing relieves the load on mobile networks as a growing number of vehicles become linked and generates an increasing amount of data. More connected vehicles and the data they exchange can be enabled by this distributed approach to data processing at the mobile network's edge, allowing more vehicles and data to be processed on existing networks.

Edge computing is quickly being adopted by the automobile industry, and the intelligent vehicles, in particular autonomous vehicles, will disrupt numerous economic sectors and significantly affect people's lives. Typical use cases that will benefit automotive Original Equipment Manufacturers (OEMs) from edge computing and 5G capabilities could include, for example, collision risk warnings (in particular at high speeds and following unexpected actions from other traffic participants), warnings of nearby vulnerable road users such as pedestrians or cyclists, cooperative lane changes, real-time traffic alerts, real-time telemetry, high-definition maps for real-time navigation, and new, innovative passenger experiences. Edge computing solutions can contribute to the effective realization of many of those use cases as they offer [10].

The growing amount of car data available opens up new possibilities for the automotive sector. The use of automotive data is strategic for industry players to generate money, lower costs, and improve safety and security. The automobile data and device ecosystem has been targeted by high-tech Internet companies which use digital platforms with data-driven business models.

Several research and surveys predict that car-related earnings will fall in the long run, with data-driven services compensating for the decline after 2050 [11]. Over-the-air upgrades and maintenance, sharing models, the penetration of car fleets with autonomous vehicles, and smart mobility services contribute to the growth of this data-based business industry. New paradigms in-car messaging and data gathering and the establishment of new opportunities and markets on top of car data processing and analysis are key for expanding the ecosystem of connected and autonomous mobility services.

5G technology and edge computing are crucial for the vehicular industry as it is undergoing a shift quicker than ever before with the rapid evolution of the technology. The increasing number of sensors in connected vehicles and roads creates a large data processing and storage issue. This requires new flexible platforms with substantial processing, reliable storage, and real-time communication for pipelining car-captured and generated data to traditional and new automotive industry players.

This work presents a platform designed as an open architecture that will allow access to in-vehicle data and resources enabling the different actors in the sector to consume it. This architecture aims to ease the development of different solutions when accessing in-vehicle data as well as the interoperability, portability, and the use of open standards to avoid vendor lock-in. For that, the captured vehicular data will be processed on consumer

demand through pipelines deployed in the Multi-access Edge Computing (MEC) that process the data samples, applying privacy, interoperability, computing, and security functions. Service Level Agreement (SLA) policies and geographic filtering will be applied.

2. Background

As stated in previous sections, 5G technology is expected to be a key enabler for reliable vehicle communication, managing their safety and automation. It will also allow for real-time data processing, allowing applications and devices to react to data instantaneously. 5G features, like Cellular Vehicle-to-Everything (C-V2X) communications or edge computing, will play a significant role in changing the automotive industry. An increase in the number of sensors on board vehicles and an increase in the number of vehicles will require a network with a high capacity for efficient functioning. 5G will also speed up the communication processing time for devices, thanks to extremely low latency.

This chapter will discuss the context in which this project is framed, as 5G technology, Software Defined Networking (SDN) and Network Functions Virtualization (NFV) paradigms; and making special mention to Cooperative, Connected and Automated Mobility (CCAM) and Edge computing technologies, the principal key enablers of this Master's Thesis. This thesis has been developed within the framework of Horizon 2020 project 5GMETA, funded by the European Union (EU) [12].

2.1 5G network and technology enablers

5G represents a fundamental redesigning of the access network that takes advantage of several technological advances and sets it on a path to enable much greater innovation. Similar to how 3G defined the transition from voice to broadband, 5G's promise is primarily focused on the transition from a single access service (broadband connectivity) to a more robust assortment of edge services and devices.

5G networks will allow many additional use-cases based on these upgraded qualities, which may be divided into three categories according to the International Telecommunication Union (ITU): Massive Machine-Type Communication (mTMC), Enhanced Mobile Broadband (eMBB), and Ultra-Reliable Low-Latency Communication (URLLC).

These characteristics will allow the start-up of products and services in which high speed is required, such as multimedia or augmented reality applications, as well as the definitive takeoff of the Internet of Things (IoT), due to the possibility of simultaneously having a huge volume of connected devices. On the other hand, it will allow applications that require responses in real time to become a reality, such as those of the connected industry, remote assisted surgery or connected vehicles, and will enable the expansion of services based on automated decisions, often using artificial intelligence. To carry out the deployment of 5G networks, network architecture and network functions are going to undergo major changes with the introduction of technologies such as virtualization, edge computing and network slicing. In Figure 1 5G use cases and their mapping to the 5G services are represented:

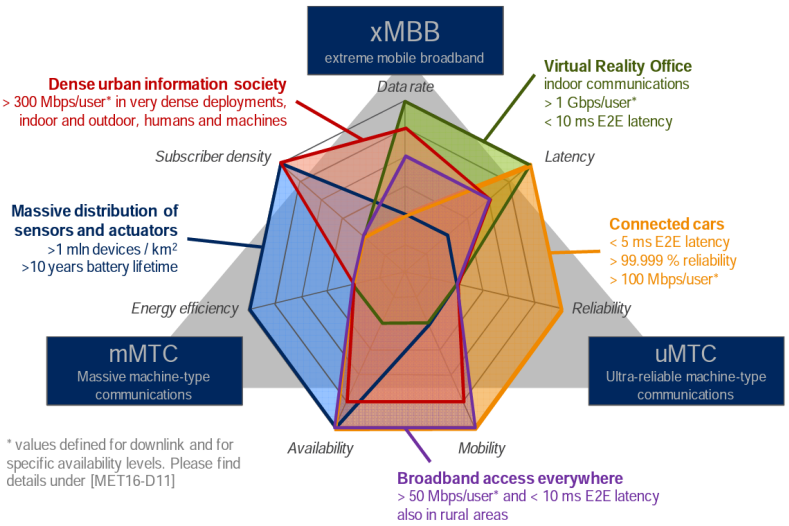


Figure 1: 5G use cases and their mapping to the 5G services

Source: [13]

- **Enhanced Mobile Broadband**: 5G is designed to deliver peak data rates up to 20 Gbps. The large bandwidth of 5G enables a huge amount of data transfer and traffic. This is particularly relevant for complex visual solutions such as augmented and virtual reality. For example, 4K videos can be loaded on a smartphone in seconds instead of minutes, enabling many new real-time applications. It is vital to note that all users on a cell tower share bandwidth. With the previous generation, if a few people were streaming a movie at the airport or watching replays of a touchdown at a stadium, the download would most likely be choppy with much buffering, and the experience would be less than pleasant.
- **Ultra-Reliable Low-Latency Communication**: Autonomous automobiles and mission-critical industrial gadgets will be controlled in real-time using 5G networks. These use cases necessitate high availability and dependability at all times. End-to-end latency — the time it takes for data to go across the network — must be exceedingly low to happen safely. 5G networks will have a latency of less than 1ms.
- **Massive Machine-Type Communication**: According to Ericsson's recent mobility report [8], 52 percent of all cellular IoT connections are expected to be Massive IoT connections by 2025. Massive IoT primarily consists of wide-area use cases, connecting vast numbers of low-complexity, low-cost devices with long battery life and relatively low throughput speeds. From environmental sensors for agricultural applications installed in remote areas that might send a few bits of data every few days or weeks to extremely high-precision, low-latency devices in nanotechnology, autonomous cars, and critical industrial environments that rely on real-time communication for potentially lifesaving functions, these devices will have widely varying network requirements.

5G networks introduce several new technologies across different layers, including radio, core, operation support system (OSS), and business support system (BSS). On other hand, new technologies that have emerged in recent years, such as Software Defined Networking [14] and Network Functions Virtualization [15], are also essential enablers of 5G networks. SDN and NFV follow the principle of decoupling software functions from general-purpose hardware, usually referred Commercial-Off-The-Shelf (COTS), where they are meant to be run.

SDN solutions focus on forwarding capabilities, i.e., layer 2 (L2) and layer 3 (L3) of the Open Systems Interconnection (OSI) model, providing a centralized control to monitor and operate distributed network routers and switches [16]. NFV technologies manage higher layers (L4-7) of the OSI model, virtualizing Random Access Memory (RAM) and Central Processing Unit (CPU) resources and simplifying the lifecycle management of software instances of network functions, referred to as Virtual Network Function (VNF), or a combination of them in a complete infrastructure, referred to Network Service (NS), on top of them [15].

Consequently, SDN and NFV are deeply changing the telecommunication infrastructures and pushing the research on network deployment, monitoring, and management. The combination of SDN and NFV capabilities will create virtualized networks embedding heterogeneous software functions and running on top of programmable network devices [17].

However, the paradigm shift introduced by SDN and NFV brings significant challenges. The most important one is their interoperability inside the network architecture. While their complementarity is evident, their integration into a common network infrastructure still presents significant issues to overcome. It means that SDN and NFV solutions will need further steps to be integrated and achieve a fully programmable virtualized network.

2.1.1 NFV

The NFV concept introduces a new way to abstract and virtualize network functions, allowing them to be created, operated, distributed, and controlled by software running on standard servers.

NFV Management and Orchestration (NFV-MANO), presented in 2, is the architecture proposed by the European Telecommunications Standards Institute (ETSI) to cope with the need to effectively deploy and manage NSs, VNFs, and the underlying infrastructure. The NFV-MANO includes three main components: the Virtual Infrastructure Manager (VIM), the Virtual Network Function Manager (VNFM), and the Network Function Virtualization Orchestrator (NFVO).

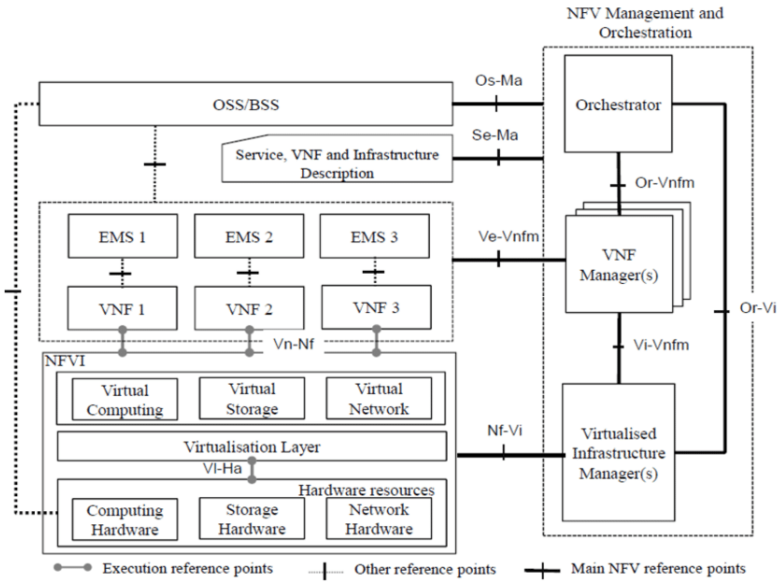


Figure 2: ETSI NFV architectural framework

Source: [18]

Each of these components provides Application Programming Interfaces (APIs) according to ETSI specifications [19] in order to intercommunicate with each other. The VIM controls and manages the NFV Infrastructure (NFVI) by virtualizing the psychical resources, including computation, storage, and network ones. It creates and assigns the virtual resources needed by each VNF. The VNFM oversees the lifecycle of VNFs deployed on top of the NFVI, including the configuration and deployment, the scaling operations, and the termination. Finally, the NFVO is responsible for the NSs and VNFs by validating them before the deployment. It is also in charge of the lifecycle management of NSs, meaning the orchestration of the different VNFs included in the NS according to programmed networking policies [20].

When considering VIM solutions, common public cloud platforms, such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform, already provide APIs to be used as VIMs. OpenVIM [21], hosted by ETSI, and OpenStack [22], supported by Open Infrastructure Foundation [23], represent the reference solutions and the most used ones. Both of them are open source and have proved to be valid for managing private data centers [24].

Regarding VNFM and NFVO, it is not easy to separate the solutions between those that provide VNFM and those that provide NFVO, as in many cases, they are together in the same suite. Focusing on open source software, the most relevant VNFM/NFVO implementations are Open Source MANO (OSM) [25], hosted by ETSI, and Open Network Automation Platform (ONAP) [26], supported by Linux Foundation.

The introduction of NFV is essential to increase flexibility when provisioning network resources. It enables fitting the Quality of Service (QoS) requirements of applications in a network while reducing the costs for its Capital Expenditure (CAPEX) and Operational Expenditure (OPEX) [27]. Operators and network service providers have been pushed to embrace NFV in order to deliver network services faster and minimize the need to manually set up specific hardware devices to establish service chains [28].

The integration of SDN and NFV enables the operation and management of VNF

instances on top of virtualized resources available at NFV-enabled data centers. The instances are interconnected through the forwarding rules of the SDN resources, such as switches and routers, established by an SDN controller.

2.1.2 SDN

To adequately define SDN technology, it is necessary to explain how a typical network operates and what symptoms led to the development of the SDN paradigm. The data plane and the control plane are the two main planes that constitute a network in general. The first, also known as the user plane or forwarding plane, displays messages generated by network users that must be transmitted according to rules. The network must undertake many actions in order to transfer these messages, including discovering the overall network topology, determining the optimum path, and deciding where the traffic should be sent, among other things. The control plane is represented by the exchanged requests to complete these operations. SDN paradigm makes management and evolution challenging [14] [29]. The SDN architecture, principles, and building blocks are depicted in 3.

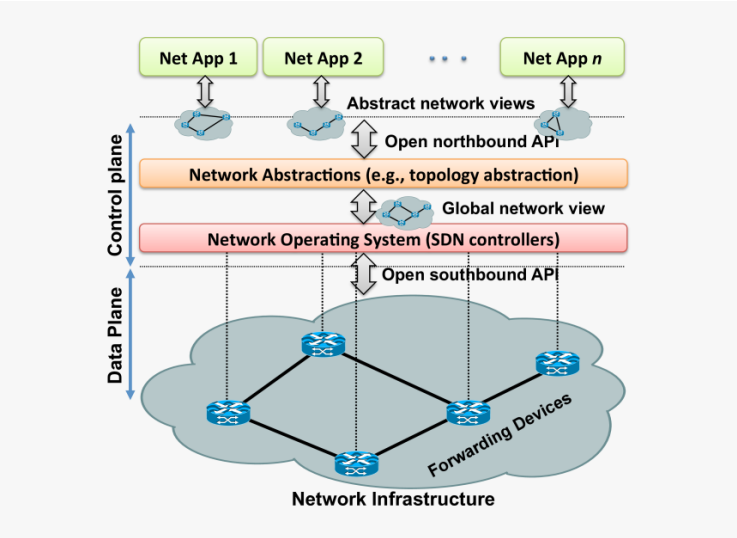


Figure 3: SDN architecture and its fundamental abstractions

Source: [20]

The main feature of the SDN paradigm is the separation of the control and data planes. The SDN controller provides a Northbound API to enable communication between the network administrator application and the control plane. The Southbound API is responsible for communication between the SDN controller and network devices. Different from the Northbound API, where each SDN controller has its API implementation, the Southbound API is usually based on standard, and widely employed protocols, such as OpenFlow [30] and Netconf [31] to bridge universal interoperability.

Concerning the control plane, lots of SDN controller implementations are available, where the most employed for research purpose are Open Network Operating System (ONOS) [32], OpenDaylight [33], RYU [34] and FloodLight [35]. A comparison of common SDN controllers is presented in [36], where several features, such as programming language, Graphical User Interface (GUI) and APIs, platform support, and internal architecture (modularity, distributed/centralized), are considered.

When the capabilities and benefits of SDN are considered, it can be concluded that its operations can be used to leverage NFV by acting as the network orchestrator. By changing flow rules at forwarding nodes and provisioning network connectivity, SDN can automate the service chaining. This complementarity between these two technologies also applies in the opposite direction. The SDN controller can benefit from NFV in reliability and elasticity because it is deployed and delivered as a service.

Network services and functions can be created as software and deployed at the network's edge as virtualized implementations thanks to the seamless integration of NFV and SDN. As a result of this collaboration, service orchestration becomes more effective by allowing the implementation of a network function at any location when required.

An SDN controller can reside with the virtualized infrastructure manager, is considered part of the NFV infrastructure, virtualized as a VNF, or merged with the OSS/BSS system. As a result, SDN controllers are installed in edge servers to provide on-demand edge services by connecting VNFs and managing infrastructure resources dynamically (i.e., computing, storage, and networking). Both together provide higher levels of automation necessary to implement edge computing, promoting the adoption into cellular and non-cellular networks. Edge computing enables service providers to launch innovative services that require cloud computing capabilities at the network's edge [29].

Multi-Access Edge Computing (MEC) is an important element of the 5G architecture. MEC represents an evolution in cloud-based computing that moves applications from centralized data centers to the network edge and therefore closer to end users and their devices. This implies a shortcut in content delivery between the user and the host, and the long network path that previously separated them.

This technology is not exclusive to 5G technology, but it is certainly essential to its effectiveness. Features of the MEC include low latency, high bandwidth, and real-time access to Radio Access Network (RAN) network information that distinguish the 5G architecture from its predecessors. This convergence of RANs and core networks will require operators to take advantage of new approaches to network testing and validation.

2.2 Edge computing

Over the years the computing paradigms have evolved from distributed, parallel, and grid to cloud computing. The inherent benefits of cloud computing include scalability, on-demand resource allocation, decreased administrative effort, and easy application and service provisioning. Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS) are the three main service models included in it. IaaS provides virtualized resources, such as compute, storage, and networking. PaaS offers software platforms for the creation, deployment, and administration of applications. SaaS is a service that offers end users and other applications software applications and composite services [37].

Cloud computing is widely used today, but it has some restrictions, though. The connectivity between the cloud and the end devices is the main limitation. Since communication is established over the Internet, it is unsuitable for many cloud-based applications, such as those that depend on latency, for example, connected vehicles or content delivery applications which generate a massive amount of data at a very high speed. The rapid development of the Internet of Everything (IoE), and the growing

amount of IoT devices or sensors also do not help at all.

The conventional cloud-only model may shortfall due to centralized computation, storage, and networking in a small number of data centers. Moreover, the relatively long distance between the edge devices and the remote data centers is vulnerable to real-time application. Therefore, there has been a need for looking 'beyond the clouds' towards the edge of the network, and edge computing technologies have emerged [38]. Bringing the computing capabilities closer features the following key values [10]:

- **Speed and Latency**: The more time it takes to process data, the less useful it becomes. Time is of the essence in the case of the autonomous car because most of the data it collects and requires becomes obsolete after a few seconds. Without a round-trip to the cloud, data latency decreases, reducing the time it takes automobiles and other connected equipment to complete control loops.
- **Optimized Data**: Edge computing will bring data processing closer to automobiles and other connected devices, lowering the quantity of data that must be sent back and forth between the cloud and the network's edge, as the data is pre-processed before, and not all the raw data is sent.
- **Security**: Edge computing aids in the security concerns for connected vehicles and devices. End-to-end security (from a vehicle or device to the cloud) is enabled by managing and deploying localized security policies, which provide real-time threat identification at the network's edge.
- **Scalability**: In most circumstances, data must initially be routed to a centralized data center, even in cloud computing infrastructures. Expansion or even simple modifications to dedicated data centers are costly. Furthermore, rather than relying on the coordination of efforts from employees at numerous locations, IoT devices can be installed along with their processing and data management capabilities at the edge in single implantation.
- **Regional Processing** : Data compliance and deployment in a particular region is possible by regional processing (in regulated countries). Using edge computing, sensitive, personally identifiable information will be locally processed.

2.2.1 Edge computing landscape

Over the last years, the concept of edge computing has evolved from many initial concepts. Under the umbrella of the edge computing paradigm, various edge computing approaches have been proposed in the literature. Terms such as Cloudlet, Fog and MEC are all umbrella concepts of Edge computing. However, each proposal's implementation specification differs in several ways.

One of the first concepts of edge computing was highlighted by Satyanaraynan et al. in 2009 [39], where authors introduced a term called "Cloudlets". The authors discussed the constraints of IoT devices which have limited computing and storage capabilities and a latency-constrained central cloud in order to create new use cases that take into account the data stream. By introducing "Cloudlets," a Virtual Machine (VM) based platform that provides compute, storage, and networking services to the devices and applications more nearby to the user, the authors suggested a need to move the cloud closer to the user/device. The concept claims that it could handle a wide range of use

cases where devices needed to boost their computing power or offload large amounts of data to the cloud in a low-latency and high-bandwidth manner[40].

Cisco [41] introduced a similar concept in 2011 called Fog computing [26] and elaborated further in 2012 by Bonomi et. al [42] by describing the characteristics of fog computing and proposing a virtualized platform to deliver the fog computing capability. Fog computing enables a better management of the Clouds by making possible services, applications, and content storage close to mobile end users. Data processing takes place locally rather than being transferred to cloud servers. Offloading, caching, location awareness, and mobility data are supported by fog computing. Applications that are time-sensitive can benefit greatly from it [37].

Both cloudlets and fog computing's primary goal is to make it possible for resource-constrained mobile or IoT devices to run and process intensive applications close to the User Equipment (UE) or edge. Low latency and low data transport costs, which enable quicker local task execution, are advantages of hosting these applications as cloudlets/-fog nodes/edge clouds rather than at a central cloud. Fog computing positioned itself as a component of the central cloud computing, and core network and smaller data centers hosted close to the users hence named as fog. Cloudlets were referred to as small data centers close to users without any network connection [43].

The term Multi-access Edge Computing (MEC), formerly Mobile Edge Computing, was created by the ETSI with the intention of leveraging software virtualization at the radio edge and pushing computational power into RANs. According to ETSI, MEC can shorten latency and give mobile users location awareness. Future mobile networks, such as 5G and beyond, should meet requirements like bandwidth (higher), latency (lower), and mobility. Therefore, both the RAN and core network should be optimized to serve billions of devices in order to satisfy such requirements. Additionally, Edge servers deal with the core network congestion problem as most of the traffic is processed locally rather than sent to the backbone networks [44].

To reduce latency and energy consumption, MEC offers access within RAN rather than core Wide Area Network (WAN). Cloudlets and Fog computing provides the services to offload the subscriber's tasks. According to the MEC requirements, the base station for the cellular network and MEC servers should be positioned together. On the other hand, Fog servers are generally provided by private environments like shopping centers, coffee shops, etc. However, they can be used in the infrastructure of Internet Service Providers (ISPs) as routers and gateways. Distributed deployment of Cloudlet is possible. However, no exact location or vendor is designated for the deployment of Cloudlets. Third generation/Long Term Evolution (3G/LTE) base station is used to access MEC server. However, a wireless Access Point (AP) with a much smaller coverage area than 3G/LTE is required to access Fog servers and Cloudlets. Wi-Fi is primarily emphasized as an access technique in the Cloudlet study. Cloudlet, however, is not constrained and can also be used with other wireless technologies.

The operators put a greater emphasis on MEC technology due to cellular networks' technological advancement. In other words, MEC is more frequently used to refer to future cellular networks than any other Edge Computing concept. Hence, for cellular networks, MEC is the de-facto Edge computing technology. The issue is that Cloudlet and Fog computing cannot function at the MEC level since they only have short-range communication capabilities, like Bluetooth and Wi-Fi.

The number of users and devices varies across each proposal. IoT use cases and vehicle-to-vehicle (V2V) connectivity are addressed by fog computing. Fog computing is

therefore anticipated to have more users and devices than Cloudlet. Cloudlet, however, excludes V2V communication from its coverage of IoT devices. The users of MEC is much smaller because MEC only focuses on subscribers and providers in cellular environment. The server density of MEC servers is also restricted to the base stations. While the Cloudlet can be set up anywhere that is open to the public. Wireless Local Area Network (WLAN) is the primary access method used by cloudlets. As a result, Cloudlet has a significantly higher density than comparable edge computing solutions. Similar to Cloudlet, Fog server(s) deployment is average and cannot be deployed everywhere [44].

2.2.1.1 MEC

The concept of edge computing is gaining more and more interest in different fields, from IoT to the distribution of multimedia content. For this reason, there are different concepts of edge. This section will concentrate on what has been defined by ETSI in the framework of the MEC group starting from late 2014 [45].

According to the ETSI, the MEC is defined as following: "Multi-access Edge Computing (MEC) offers application developers and content providers cloud-computing capabilities and an IT service environment at the edge of the network. This environment is characterized by ultra-low latency and high bandwidth as well as real-time access to radio network information that can be leveraged by applications. [46]"

MEC is a new paradigm that provides computing, storage, and in general, networking resources within the edge of the network. While at the beginning, the group was focused on cellular networks, today, it has broadened its interest to every network type, e.g., Wi-Fi or cabled. MEC servers are deployed on generic computing hardware, allowing delay-sensitive, highly scalable, and context-aware applications to be executed close to the end users. An application can also be a network function chained with others to realize a more complex end-to-end service.

The two main benefits of MEC systems are the reduction of latency and the possibility to improve localized user experience thanks to the proximity to the users, even though the processing power and storage of a MEC system is typically limited with respect to that of the centralized cloud. It is worth mentioning that MEC paradigm is not opposed to the Cloud principle, in fact, the two concepts complement each other.

The position of a MEC server inside the network operator depends on the use case requirements but also on the CAPEX and OPEX expenditure. In general, the MEC server could be placed very close to the gNodeB (gNB) where the best performance in terms of latency and scalability can be obtained, but with a high cost of deployment and maintenance (MEC server is needed for each base station). Moving the MEC server towards the operator's core means a slight increase in the latency and more devices to be managed, and lower maintenance costs. It is worth mentioning too that the MEC can be seen as an opportunity to find new revenues for the network operators. MEC servers can host third-party services with access to data and information not available in the cloud and so with a potential big value to build novel applications.

The MEC ETSI Industry Specification Group (ISG) has defined a reference architecture with different APIs to be used among the different components of the MEC architecture. These APIs are related to the basic functionality of the application development platform and to the management and orchestration layer. The general framework is shown in Figure 4:

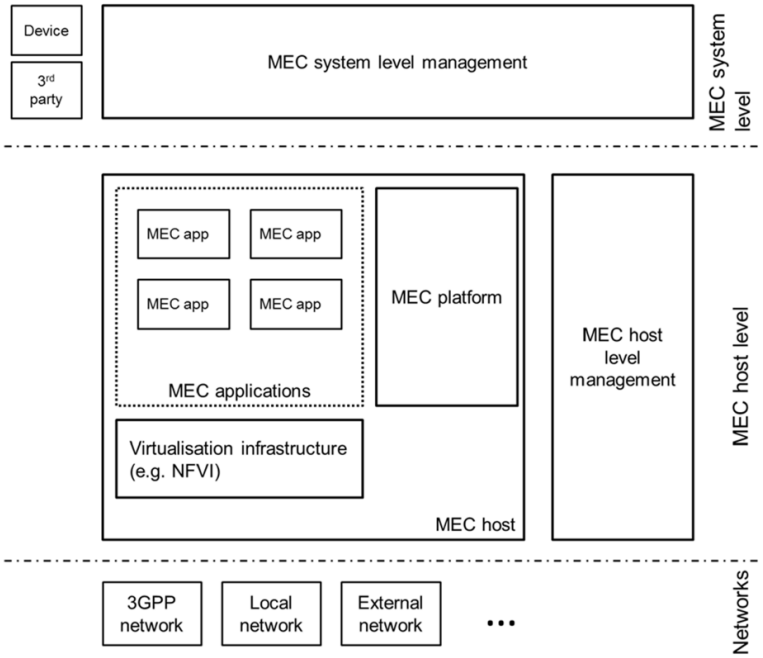


Figure 4: Multi-access Edge Computing architecture

Source: [47]

The MEC host level is where the MEC applications are hosted typically on top of a virtualized or containerized infrastructure that provides computing, storage, and network resources. This level also includes a management subsystem that is the interface with the orchestration in the MEC system level. The management level has the global view of the system and some of its responsibilities are: coordination and control of instantiation, resolving resource conflicts, scaling up and down resources, etc. The management block is similar to the same level in NFVs, and the two concepts are strictly related (e.g., MEC apps can be implemented as VNFs). The MEC orchestrator is also responsible for checking that the application's status complies with the target requirements (e.g., throughput, latency, etc.). The orchestrator can decide to allocate additional resources or relocate applications in order to fulfil their needs.

Moreover, ETSI ISGs have worked on the integration between MEC and NFV MANO and they have already achieved a good degree of integration. Other milestones in MEC standards are the use of container technology, in addition to the virtualization one, the multi-tenancy and the network slicing at the edge.

MEC standardization activities are held in the corresponding ETSI ISG group as explained previously. Several standards have been released, the main one is: MEC 001 [48] which provides a glossary of terms relating to the conceptual, architectural, and functional elements within the scope of Multi-access Edge Computing. On the other hand, MEC 002 [49] specifies the requirements for Multi-access Edge Computing with the aim of promoting interoperability and deployments. The framework and reference architecture for MEC is described in MEC 003 [47] where a MEC system that enables MEC applications to run efficiently and seamlessly in a multi-access network is depicted.

Moving from the architecture to the APIs, it is worth mentioning MEC 009 [50], which defines the design principles for RESTful MEC service APIs, provides guidelines and templates for the documentation of these, and defines patterns of how MEC service

APIs use RESTful principles. The lifecycle of the applications together with rules and requirements management are highlighted in MEC 010 [51]. Other relevant APIs are presented in MEC 013 [52] that describe the Location APIs, useful to access the UE position as known by the network.

MEC groups are also working on the CCAM field: MEC 022 [53] is a Study on MEC Support for V2X Use Cases, while MEC 030 [54] specifies the API to facilitate V2X interoperability in a multi-vendor, multi-network, and multi-access environment.

2.2.2 Uses-cases of edge computing

We live in a technologically advanced, intelligent world with smart technologies. As a result, many people are unaware that edge computing is a part of their daily life. Edge technology makes possible anything from remote office work to remote surgery, smartphones to smart cities, self-driving cars and voice-controlled products.

Edge computing is crucial for enterprises to function with the greatest operational efficiency, higher safety, and better performance at an enterprise and industrial level. It opens the door for better and more innovative ideas. Every industry vertical can benefit from edge computing, including banking, healthcare, retail, and mining.

While there may be dozens of examples and use cases for edge computing, here we focus on some of the more important ones [55]:

- **Automotive:** Latency is a significant barrier to the widespread adoption of connected and, subsequently, self-driving automobiles. Autonomous automobiles will need to engage in a significant volume of real-time data transmission in order to operate efficiently and securely. Even with the advent of 5G connection, the ecosystem needs for such exchanges may not be met by the speed and dependability of those exchanges. This problem is partially resolved by the newly developed C-V2X communication protocol. However, edge computing has the potential to accelerate its uptake and expand the range of V2X services for connected and autonomous vehicles. Delays in information in this regard could be all the difference between endangering a life and saving one.
- **Manufacturing:** Industrial leaders are starting to encounter speed restrictions in data transmissions as Industry 4.0 initiatives transition from the pilot stages to mass deployments. In fact, more resilience, speed, and bandwidth are needed to support more complex networks of Industrial IoT devices and, potentially, completely automated factories. Once more, edge architecture might accomplish that.
- **Augmented Reality (AR) and Virtual Reality (VR):** Due to the growing demand for AR/VR experiences, edge computing applications have also increased recently. However, the present barrier to widespread adoption is the high cost of the equipment. Users must invest in pricey head-mounted displays, consoles, and computer setups since VR programs demand a lot of processing and storage space for graphics. Edge computing has the potential to significantly lower the cost of VR and accelerate its adoption in other sectors like education, hospitality, or travel.
- **Healthcare:** The healthcare industry, particularly the manufacture of medical monitoring equipment, has a lot of potential and opportunity thanks to edge computing. It could change how inpatient and outpatient records are handled. In conjunction with automation and machine learning, Edge computing could quickly

identify patients exhibiting troubling symptoms and take action to help them in real-time when devices like health tools measuring heart rate, temperature, glucose monitors, sensors, and other medical equipment record data.

- **Smart cities:** Massive volumes of data are necessary for smart cities to function. Edge computing can fuel every component of a smart city, including autonomous vehicles, intelligent street lights, intelligent manufacturing, intelligent power grids, and public transportation that can be monitored for increased efficiency.
- **Video streaming:** The methods of accessing content have significantly evolved over time, from cable to streaming. Customers need a pleasant streaming experience even if High Definition (HD) video streaming uses a lot of bandwidth. Moving the load close by and edge caching content can dramatically enhance content delivery.

A growing number of new edge computing use cases will probably emerge as the edge computing business grows.

2.3 C-V2X and CCAM

Autonomous driving capabilities in vehicles have been a growing reality on our roadways in recent years. Huge research and development efforts are being made to incorporate connectivity features into these vehicles to enable information exchange between them and their surroundings, also known as Vehicle-to-Everything (V2X) communications, in order to significantly improve road safety and to further this revolution.

Furthermore, the automotive industry is moving toward a future where vehicles are increasingly wirelessly connected, automated, and cooperative for safer and more effective driving. Today, on-board sensors, which are restricted to visual line-of-sight, support the majority of automobile safety and efficiency functions. By increasing vision and detection ranges even when visual line-of-sight is impossible, connectivity is a useful complement to the on-board sensors. Additionally, connectivity is essential for cooperative activities [56].

Cooperative Intelligent Transport Systems (C-ITS), will enable road users and traffic controllers to share information and use it to coordinate their actions. By assisting the driver in making the best decisions and adjusting to the traffic situation, this cooperative element—made possible by digital connectivity between vehicles, and between vehicles and transportation infrastructure—is anticipated to significantly increase road safety, traffic efficiency, and driving comfort.

In order to boost the safety of future automated vehicles and ensure their full integration into the larger transportation systems, communication between vehicles, infrastructure, and other road users is essential. To further support EU countries and the European automotive industry in their transition to connected and automated driving, the European Commission started the Cooperative, Connected and Automated Mobility (CCAM) initiative [57], for cooperation with European standardisation bodies to ensure interoperability of new technologies being developed.

The introduction of C-ITS (The main standard for CCAM [58]) as well as new capabilities for CCAM are backed by V2X technologies, which are viewed as crucial technologies for

enabling vehicles to communicate with one another and everything around them. C-V2X (Cellular Vehicle-to-Everything) is a new concept that refers to cellular technologies adapted for connected vehicle communication. C –cellular- refers to 4G LTE and 5G New Radio (NR) 3rd Generation Partnership Project (3GPP) technologies. C-V2X term covers communication between vehicles, and other road users or entities by using the cellular network (so-called, Vehicle-to-Network –V2N- communication) but also direct communication (either Vehicle-to-Vehicle -V2V-, Vehicle-to-Infrastructure –V2I- or Vehicle-to-Pedestrians -V2P).

C-V2X was firstly published by 3GPP in Release 14. The direct communication procedures used by C-V2X are based on Device-to-Device (D2D) communications, firstly introduced in Release 12. Release 15 deals with 5G, specifying the support for data connectivity over NR and LTE access technologies. The enhanced V2X services foreseen are advanced driving, extended sensors, remote driving, and platooning. Release 16 provides full LTE and NR communication support, including the 5G core, interworking between both technologies, and an extensive list of enhancements.

3. Objectives and scope

The main goal of this work is to propose and implement a novel edge-computing platform that fits to the requirements of the use cases of connected vehicles.

The platform will capture vehicular data and instantiate on consumer demand pipelines that process the data samples according to Service Level Agreement (SLA) policies and filter it from specific geographic areas. The deployed pipelines in the Multi-access Edge Computing (MEC) platform will aid in delivering real-time through data privacy, interoperability, computing, and security functions embodied in Virtual Network Functions (VNFs) deployed in a Network Functions Virtualization (NFV) enabled architecture.

With that aim, the specific objectives can be stated as follows:

1. Study the current context [Section 2] and state-of-the-art in edge computing [Section 5.1].
2. Analyse the current existing edge computing frameworks for connected vehicles [Section 5.2].
3. Analyze the requirements of the use cases in connected vehicles (input from the 5GMETA European project) and understand their implications [Section 6]. Analyze the different solution alternatives for fitting the requirements [Section 7].
4. Design, define and implement a novel proposal for an edge-computing platform that fits the aforementioned analysis and requirements [Section 9].
5. Validate the proposed framework [Section 10].

Thanks to the proposed framework, a flexible and modular 5G edge approach will be provided to catalyze innovative data-based services for connected and autonomous mobility applications, delivering data pipelines and sending relevant data to data-based services for the benefit of new stakeholders.

4. Outcomes

The following section outlines the main technological, economic and social outcomes of the work carried out.

4.1 Technical outcomes

This work proposes an architectural approach designed and implemented to facilitate the delivery of a vast volume of data coming from the vehicles and their surroundings and to filter this data from specific geographical areas. Furthermore, some novel features provided by the proposed architecture, when compared to market solutions, are the separation of data producers and consumers through dedicated IoT topics where pipelines read raw samples and write processed data, the allocation of computing resources adapted to the data consumption demand and the hosting of services which require low latency in MEC infrastructures applying the container-based life cycle management for pipelines to third-party services. The presented architecture also embeds many components to ensure security and privacy by design and by default, offering an efficient and secure platform to process the data.

The design of this architecture proposes a technological advance in the processing of automotive data while it will bring better use of the resources of MEC systems, as well as an innovative way to filter the different data that the MEC processes.

4.2 Economic outcomes

Taking advantage of the large amount of data generated by the different in-vehicle sensors, the use of real-time data processing is expected to have an overall positive impact on our economy. The availability of automotive data can support the continued growth of the Small and Medium-sized Enterprise (SME) market and the creation of new employment opportunities, thanks to the opportunities for new business concepts based on new technologies as the ones proposed by this work.

The stakeholders can utilize collected data in the automotive ecosystem to bring new data-driven services, innovative products, and new business models into the market. These data can be vital in the advancement of not just the automotive industry but also for new players that could enter the market.

4.3 Social outcomes

Even though the project will focus on technological outcomes, exploring a novel edge computing platform for connected cars could also bring some social outcomes. Thanks to the data collected and delivered by the platform, new indirect benefits are expected to be generated from increased road safety and traffic efficiency, greener traffic, as well as a reduction in time spent on the road and a reduction in accident rates.

5. Edge Computing: State-of-the-art

To optimize the design of a framework at the network edge, an initial analysis of related concepts/technologies and state-of-the-art literature must be performed.

5.1 MEC technologies for CCAM services

The following section presents a study of different research works about MEC implementations in the vehicular context. The application of MEC technology to boost the response of CCAM services or to empower the processing of automotive data is a novel research field with some works studying edge computing in vehicular communications.

A MEC architecture and some APIs for C-V2X systems are proposed in [59], checking the feasibility of hosting CCAM use cases in the MEC. The work presented in [60] goes a step further in describing a Docker-based ETSI-compliant MEC platform, taking benefit of the microservice paradigm. Thanks to the modularity of the microservices, a service migration procedure is proposed.

In regard to mobility inherent to users of cellular networks, the handovers between different MEC servers have gained a lot of attention. The problem of maintaining service continuity and synchronization of relevant data among multiple MEC servers to support vehicular applications is studied in [61] and an architecture is proposed for this end.

Two different frameworks for integrating big data analytics with vehicular edge computing are proposed in [62] and [63]. These frameworks select the offloading infrastructures from the available ones satisfying heterogeneous requirements. Another approach involves a hierarchical model for resource management at MECs targeting latency and energy efficiency requirements [64].

Two topics are paramount when studying the fields related to vehicular data sharing using edge computing platforms and cellular networks: privacy and efficiency. Regarding privacy concerns, secure peer-to-peer data sharing systems are necessary to prevent second-hand data sharing without authorization, as the one based on blockchain proposed in [65]. Regarding processing and communications efficiency, a hierarchical edge framework performing computation offloading and content caching is proposed [66] to minimize network communication overheads for recalled tasks.

All the presented works ignore critical aspects of vehicle data platforms to connect data sensors with future CCAM applications. They focus on processing all the produced data, lacking a mechanism to trigger data processing offloading in the MEC when the

consumption demand is there. All the data processed and not consumed mean a big overhead and an inefficient approach. Additionally, they put SLA levels aside ignoring business models when allocating computing resources.

5.2 Open-source Edge computing frameworks

This section analyzes some existing open source initiatives that currently revolve around developing solutions for edge computing. The main interest when presenting these solutions is to provide insights about existing developments and baseline solutions that can potentially be used to enrich more complex, business-driven use cases.

KubeEdge [67] is an open source Kubernetes distribution originating from Huawei, currently adopted by the Cloud Native Computing Foundation (CNCF) as an incubating project. It is specifically designed for use in an edge environment, with a control plane in the cloud and worker nodes at the edge. It provides essential infrastructure support for network app deployment and metadata synchronization between cloud and edge. It allows a light installation on devices with limited computing and storage resources, although it requires the existence of a previous K8s cluster. This implies that it must be installed on top of a base installation of K8s. It is an appropriate distribution for IoT environments, which supports device management and communication with them through Message Queuing Telemetry Transport protocol (MQTT).

Akraino [68] is an opensource initiative of Edge Computing platforms from Linux Foundation (LF) Edge. Is a set of open infrastructures and application blueprints for the Edge, spanning various use cases, including 5G, AI, Edge IaaS/PaaS, IoT and Automotive, for both provider and enterprise edge domains. These Blueprints, created by the Akraino community, focus exclusively on edge in different forms. What unites all of these blueprints is that they are community tested and are ready for adoption or used as a starting point for customizing a new edge blueprint. Its mission is to create a unified community for open source edge that fosters cross-industry collaboration across IoT, Telecom, Enterprise, and Cloud ecosystems. Enable organizations to accelerate adoption and the pace of innovation for edge computing and deliver value to end-users by providing a neutral platform to capture and distribute requirements across the umbrella.

EdgeX Foundry [69] is an open-source, vendor-neutral, Edge IoT middleware platform under the Edge umbrella. It collects data from different sensors at the Edge and acts as a dual transformation engine sending and receiving data to and from enterprise, cloud, and on-premise applications. EdgeX enables autonomous operations and Artificial Intelligence (AI) at the edge. EdgeX translates and transforms the information from sensors and devices and delivers it to applications over network-based protocols in formats and structures that meet customers' needs. It also takes data from applications and delivers it to the edge nodes/devices for updates, control, and actuation.

Kube5G [70] is an open source platform that fosters research and development of cellular mobile networks, 5G and Beyond, in a cloud native environment. Services can be deployed on baremetal, virtual machines, and/or any private/public cloud using the Kube5G platform. Kubernetes is used as container orchestration for both applications and infrastructure. Kube5G, which was inherited from K8S, can support both CNF and VNF applications by using docker and kubevirt as runtime environments. Kube5G supports a variety of value-added extensions, including management and orchestration like OSM, monitoring and network intelligence like Kubeflow, and Kube5G-Operator for zero-touch configuration and dynamic service updating/upgrading. It also presents a

revolutionary method for creating and packaging a telco network function that complies with cloud native technology in the form of nested, clearly defined layers.

StarlingX [71] is a complete edge cloud infrastructure software stack used by demanding applications in industrial IoT, telecom, video delivery, and other ultra-low latency use cases. StarlingX provides a container-based infrastructure for edge implementations in scalable solutions with the deterministic low latency required by edge applications and tools that make distributed edge manageable.

Baetyl [72] is an open edge computing framework of LF Edge that seamlessly extends cloud computing, data, and services to edge devices. It can provide temporary offline, low-latency computing services such as device connection, message routing, remote synchronization, function computing, video capture, AI inference, status reporting, and configuration optimization, among other things. It provides a edge cloud integration platform, which adopts cloud management and edge operation solutions, and is divided into an Edge Computing Framework and a Cloud Management Suite. It can manage all cloud resources, such as nodes, applications, configuration, and so on, and can automatically deploy applications to edge nodes to meet a variety of edge computing scenarios.

6. Analysis of requirements

Since this Master Thesis is part of a European project called 5GMETA, this work received as an input the requirements created by the European consortium, with key stakeholders in the connected vehicles sector, and categorizes them in a requirements table that the proposal should fulfill.

The whole platform must address all the requirements stated in the presented table. During the course of this thesis different parts of the platform have been designed and developed. The requirements marked in green are the ones that the developed modules have to comply with and address.

The accomplished requirements will be validated in Section 10 mapping them to the developed modules and explaining why the requirement is validated.

Table 1: Requirements to address

Tier	Requirement Category	Requirement	Req ID	Description
MEC	General / Platform	Use of VNF	REQ01	According to the dataflows subscribed by CCAM applications, it is needed to deploy VNFs in the MEC infrastructure to run microservices that run pipelines to connect data an CCAM service. This means that dataflows not being demanded by CCAM applications are directly dropped at the edge and only the dataflow with CCAM subscribers are processed.
		Datatype Pipeline execution	REQ02	Be able to deploy a datatype specialized VNF at the MEC infrastructure, being downloaded from a Dockerhub repository, managing the VNFs lifecycle.
		Pipeline allocation	REQ03	Accept containers with Third Party signatures and certify the limited interaction with MEC local services.
		Data Processing Scalability	REQ04	Provide hierarchical messaging structures to connect multiple live dataflows to different CCAM services.
		Scalability at MEC level	REQ05	Produce dataflows tag with origin and delivered to CCAM applications.
		Uplink and Downlink dataflows	REQ06	Make possible to send data from sensors to CCAM services . and to push notifications from CCAM services to sensors and devices of an area subscribed to events bus.
	Security / Privacy / Data protection	Intra services flows confidentiality	REQ07	The access to internal dataflows is limited to local systems. Dataflows are regenerated to avoid access to data origins.
	Data , Management	Specialised Processing Capacity	REQ08	MEC infrastructure will provide hardware acceleration resources when available for high demanding VNFs. However, when possible raw processing will be avoided, and sampling rates will come first to avoid oversampling for anonymization or sharing, applying first the SLA.

Tier	Requirement Category	Requirement	Req ID	Description
Cloud	General / Platform	Use of VNF	REQ09	Approach required to create the requested data pipelines in a dynamic way and in general to have a flexible and scalable platform
		Datatype Pipeline execution	REQ10	To scalably create the data pipelines based on developer' requirements.
		Pipeline allocation	REQ11	To on board and instantiate the data pipeline.
		Resources allocation on MEC	REQ12	The platform should allocate resources on the edge servers fulfilling the SLA requested by the developers.
		5G network resources monitoring	REQ13	To use the SDN APIs to monitor that the allocated resources of the 5G network can satisfy the SLAs.
		5G network resources allocation	REQ14	To use the SDN APIs to allocate resources of the 5G network for satisfying the SLAs.
		Data Interoperability	REQ15	To make available to developers data from different sources in a consistent way.
		Data Processing Scalability	REQ16	To automatically scale the processing of large quantities of data.
		Scalability at MEC level	REQ17	To exploit the geographic diversity of MEC server to implement a scalable approach for the collection of data from sensors and devices.
		Uplink and Downlink dataflows	REQ18	The data flows should be managed both in uplink (to collect data) and in downlink (e.g. to give feedback to vehicles).
		Data anonymization	REQ19	The data coming from different sources and devices must be anonymized.
	Security / Privacy / Data protection	Data access	REQ20	To manage the accounting policies.
		User registration	REQ21	To authenticate users accessing the loud Platform.
		Intra services flows confidentiality	REQ22	All the flows among different services in the platform should be protected.
		Data access authorization	REQ23	To authorize the access to data flows.
	Third party APIs	API registration	REQ24	To make data available through Third Party APIs.
		API format	REQ25	To implement a widespread API formatting approach such as the OpenAPI.
		API protection	REQ26	To protect access to the API exploiting the AAA module.
		API discovery	REQ27	Developer's APIs for Sensors and Devices have to connect to data broker according to a protocol to provide dataflow metadata including licensing, sample-rate, resolution and type, push dataflows according to the returned configuration and subscribe to published event topics.
		API data broker	REQ28	Developer's APIs for Sensors and Devices have to connect to data broker according to a protocol to provide dataflow metadata including licensing, sample-rate, resolution and type, push dataflows according to the returned configuration and subscribe to published event topics.
		API data gateway	REQ29	Developer's APIs for CCAM applications have to connect to a single endpoint where all the dataflows are signaled and provided with IoT and standard communication protocols and where notifications to specific sensors or areas can be pushed.
	5G features requirements	NFV and SDN approach	REQ30	To enable a dynamic configuration of the platform components.
		Network slicing	REQ31	To provide adequate resources to fulfil heterogeneous SLAs for each CAM service.
		MEC based approach	REQ32	To distribute the load of some of its functionalities from the cloud platform to the edge servers to ensure scalability.
		MEC based approach	REQ33	To guarantee low latency communications to time sensitive CAM services.

7. Analysis of alternatives

This chapter presents an analysis of different alternatives that have been considered to design and implement the proposed solution, based on the requirements analyzed before. For each alternative solution, several variants will be presented and their strengths and weaknesses will be analyzed. Later, an analysis of the main characteristics of the solution will be made.

To make the best choice of the described alternatives, a grade from 0 to 10 will be assigned in the evaluation criteria.

Concerning the technology stack at the network edge to support an edge platform for vehicular data processing, different options can be used for resource virtualization, orchestration, monitoring or messaging.

7.1 NFV Orchestration Systems

Prior to the introduction of NFV, it was common practice to deploy network applications and services using specialized proprietary hardware and software that could only be used in particular installations, being an unyielding system. NFV overcomes challenges like reducing capital and operating expenses and satisfying the growing demand for mobile services. Due to NFV, software and services may be deployed in any environment, enabling them to be virtualized. NFV Management and Orchestration is a crucial infrastructure for unlocking the full potential of the virtualization of network functions. The most relevant implementations are Open Source MANO (OSM), hosted by ETSI, and Open Network Automation Platform (ONAP), supported by Linux Foundation.

Both OSM and ONAP bring interesting alternatives, but OSM is more widely employed in research projects funded under the umbrella of the Horizon 2020 5G Infrastructure Public Private Partnership (5G PPP) programme as analyzed in [73]. In order to decide which better fits the project requirements, the key features to be considered are:

- Technological, comprising the support of baremetal infrastructures, multiple VIMs, slicing.
- Monitoring, support of performance with third party frameworks.
- Learning curve, documentation and community.

Considering those features, OSM supports Bare Metal or VM hosts. ONAP's base architecture also supports installation in bare metal infrastructure. Concerning network slicing, the basic idea is to create dedicated virtual networks using a common physical

infrastructure for a specific service by using independent logical network functions. This component aims to become complementary to the already used SDN/NFV technologies to improve the existing network infrastructure resources usage and management.

OSM has an integrated slice manager that follows the guidelines provided by ETSI for slicing support since release 5. In the case of ONAP, from Guilin Release version onwards, orchestrates network slices in all three domains – RAN, Transport, and Core, expanding upon the end-to-end network slicing introduced with Frankfurt Release.

Regarding the compatibility with multiple VIMs, OSM supports deployments in Openstack-based cloud environments, VMware, Amazon Web Services (AWS), Whitestack or Wind River for example. ONAP MultiCloud aims to mediate most interactions between ONAP and any underlying VIM or Cloud to enable ONAP to deploy and run also on multiple infrastructure environments such as OpenStack, VMware, Azure, AWS, etc.

Concerning monitoring support, OSM has a monitoring module designed to support a flexible plugin method to integrate with the monitoring tool of choice. The Data Collection, Analytics and Events (DCAE) module of ONAP is responsible for gathering performance, usage and configuration data about the VNFs and their underlying infrastructure. It provides APIs for developers of data analytics applications.

OSM, in general, has an easier learning curve than ONAP. The time and effort required to set up the platform or to attach a VIM are less complicated in the first solution. OSM also provides a very good wiki and has active community support. ONAP is not as well documented and sometimes is difficult to get clear information, but it has huge support from the community and main technology companies.

Table 2: Comparison between OpenStack, OpenShift and Kubernetes

Criteria	**OSM**	**ONAP**
Resource footprint (10%)	Small resource footprint (8)	High resource footprint (5)
Bare metal server installation support (5%)	Yes (10)	Yes (10)
Kubernetes installation support (30%)	Supported (10)	Yes, ONAP-B can have multiple instances with different name-spaces (10)
Performance monitoring (10%)	Open for 3rd party monitoring services (6)	DCAE (Data Collection and Analytics Engine) module of ONAP is responsible for this with much richer APIs for developers of Data Analytics Applications (7)
Multi-VIM support (20%)	OpenStack, VMware, AWS, Microsoft Azure (9)	OpenStack, VMware, AWS, Microsoft Azure (9)
CLI support (5%)	Powerful CLI (8)	Not user friendly (5)
Life-cycle management support (5%)	Subscribe life-cycle management event (8)	Not available (0)
Documentation and community (5%)	Very good wiki and active community support (8)	Not well documented, pretty difficult to get clear information (6)
TOTAL	**7.5**	7.05

Taking these characteristics into account, there is no one solution that prevails over the other. Nevertheless, as OSM is more widely employed in research projects, we will consider it as the orchestration solution for the proposed platform.

7.2 Virtualization technology

With respect to the virtualization technology, en edge platform can be designed in different ways depending on the operations to be made. If infrastructure layer

services must be provided, OpenStack is a reference technology. If the platform handles containers, other technologies such as Kubernetes or OpenShift are more appropriate. Anyway, it is important to consider the following criteria when selecting technologies:

- Flexibility on the heterogeneous OS to virtualize, support of virtualized hardware acceleration, configuration formats, readiness for dynamic changes or access from outside infrastructure.
- TRL including open-source project versus industrial/commercial product.
- Learning curve, documentation and community.

Kubernetes and OpenShift are both open-source software platforms that facilitate application development via container orchestration in a robust and scalable architecture. They make managing and deploying containerized apps easy. Due to the similarities, the decision to choose one of the two platforms can be difficult. In the following an overview of both platforms are provided.

Kubernetes is an open-source Container-as-a-Service (CaaS) platform for managing containerized workloads and services. It handles scheduling onto nodes in a compute cluster and actively manages workloads. Kubernetes is flexible when it comes to running on different operating systems, and it can be run in multiple environments, including on-premises, public, or hybrid cloud infrastructures. Kubernetes lacks a networking solution but lets users employ third-party network plug-ins and provides monitoring capabilities to help check the health of servers and containers. It neither does have built-in authentication or authorization capabilities. Kubernetes enables you to set up your own Docker registry, but no integrated image registry exists. Kubernetes has a complicated web console, which makes it difficult for novices, but it has a large and active online user community and new features get added frequently. The user community also provides technical support that encourages collaborations.

OpenShift is a Platform-as-a-Service (PaaS) that operates independently of cloud resources through containerization. From OpenShift version 3 onwards, Docker is the prime container technology, and Kubernetes is the container orchestration technology. OpenShift can be installed on Red Hat Enterprise Linux (RHEL) and CentOS. Version 4 currently supports AWS and vSphere. OpenShift comes with Prometheus, which is a DevOps database and application monitoring tool. It allows users to visualize the applications in real time, using a Grafana dashboard. Regarding the security policies, OpenShift restricts you from running simple container images and many official images, requiring specific privileges to maintain a minimum-security level. OpenShift has a very user-friendly web console that allows users to perform most tasks directly on it. OpenShift has no vendor lock-in and provides a vendor-agnostic open-source platform, allowing users to migrate their own container processes to other operating systems as required without taking any extra steps. OpenShift has a much smaller support community that is limited primarily to Red Hat developers.

While Kubernetes helps automate application deployment, scaling, and operations, OpenShift is the container platform that works with Kubernetes to help applications run more efficiently.

On the other hand, OpenStack is an Infrastructure-as-a-Service (IaaS) platform that controls large pools of compute, storage, and networking resources throughout a datacentre. It allows to build and manage cloud computing platforms for public and

private clouds. However, OpenStack offers minimal support for containers and instead focuses on bootable virtual machines, meeting the hyper scale-demands of massive network providers. It can even provide Bare Metal as a Service for high-performance applications without the usual management complexity.

The CaaS and IaaS archetypes offered solutions to relatively similar problems but do so on different layers of the stack. Deploying a container platform such as Kubernetes and OpenShift with a cloud infrastructure platform such as OpenStack can offer a very good solution for scalability and automation, allowing faster delivery of infrastructure.

Table 3: Comparison between OpenStack, OpenShift and Kubernetes

Criteria	**OpenStack**	**OpenShift**	**Kubernetes**
Container support (25%)	Minimal, with some plugins, VM focused (5)	Yes (10)	Yes (10)
Monitoring (25%)	Basic monitoring (6)	Yes (9)	Offers an API for that, third-party tool needed (8)
Storage orchestration (10%)	Yes (10)	Yes (10)	Yes, through a plugin (10)
Scaling (10%)	Yes, complicate (6)	Yes (7)	Yes, very powerful (10)
Self-healing (10%)	Partially (6)	Partially (6)	Yes (10)
Load-balancing (10%)	Yes (10)	Yes (10)	Yes (10)
Learning curvem (5%)	Difficult (7)	Medium (8)	Difficult (7)
Documentation and community (5%)	Not many and difficult documentation (5)	Smaller support community (5)	Large and active support (10)
TOTAL	6.55	8.7	**9.35**

Finally, Kubernetes has been chosen because it gives high availability, low latency at the edge, and the ability to iterate and quickly deploy different services. It is very versatile and fast scalable. Because of its automated management features, Kubernetes can react instantly to changes at the edge, being a very fittable solution for a MEC platform.

7.3 Monitoring system

In today's world, with many services fueling hundreds of components, the failure of just one piece can cause a crash for the whole system. The solution is to constantly monitor key characteristics. Thus, the infrastructures need to be monitored, given the importance of preventing errors and system failures. In addition, obtaining metrics and logs of the deployed applications is key to measuring their performance, status, or speed in real-time and automatically reacting to a given event.

Multiple monitoring and visualization tool options are available, such as Prometheus/-Grafana and Elasticsearch, Logstash and Kibana (ELK) stacks. Monitoring the platform and its workloads allows for scheduling the server's resources and reserving a portion of the Central Processing Unit (CPU) and memory resources for use in different services.

Prometheus is an open-source metrics monitoring tool. Metrics are collected and stored as time-series data (metrics that change over time). It is appropriate to monitor metrics from both static container environments and traditional IT infrastructures. Metrics collection, storage, and querying are its main objectives. For its part, Grafana is a popular open-source visualization and analytics tool that focuses on providing rich visualizations of time-series metrics. These two tools used together, become very powerful in providing amazing insight into infrastructure and services performance.

On the other hand, ELK stack is a combination of three open-source tools (Elastic-

search, Logstash and Kibana), that form a log management platform that specializes in searching, analyzing, and visualizing logs generated from different systems. They include a distributed storage system, a search and processing engine, and a display system.

Table 4: Comparison between ELK stack and Prometheus / Grafana

Criteria	**ELK Stack**	**Prometheus / Grafana**
Resource footprint (10%)	High resource footprint (5)	Small resource footprint (10)
Scope (15%)	Log analysis (6)	Metric monitoring (9)
Database (10%)	No SQL (8)	TimeSeries DBMS (8)
Queries (15%)	Domain-specific query language based on JSON, also SQL-like queries (8)	PromQL (8)
Retention (10%)	Long-term data retention (8)	Difficult for longer periods (6)
Data stored (15%)	Numeric, string, boolean, binary data (9)	Numeric examples of named time series (5)
Access method (10%)	RESTful HTTP/JSON API access methods (10)	RESTful HTTP/JSON API access methods (10)
Integration with systems (10%)	High (10)	High (10)
Learning curve (5%)	Difficult (5)	Medium (7)
TOTAL	7.8	**8.05**

The main objective of the monitoring solution is to gather metrics so that the platform can react to a given event. Prometheus/Grafana is a more lightweight solution for the MEC and it has very good compatibility with Kubernetes system, becoming a very good option for the solution proposed.

7.4 Messaging protocol and data broker

Message Queuing Telemetry Transport (MQTT) is a standard messaging protocol for IoT. It is designed as an extremely lightweight publish/subscribe messaging transport protocol based on Transmission Control Protocol (TCP), ideal for connecting remote devices with a small code footprint and minimal network bandwidth while providing multiple QoS options. Nowadays, MQTT is used in a wide variety of industries, such as automotive, manufacturing, telecommunications, oil and gas, etc.

Advanced Message Queuing Protocol (AMQP) is a message-oriented middleware protocol commonly used for IoT, as an alternative to publish/subscribe protocols. It relies on TCP for message delivery and ensures interoperability between vendor implementations by enforcing a common message transfer/interpretation approach. One of its main characteristics is its ability to be extended to support application-specific properties.

Table 5 shows the main differences between the two protocols.

Table 5: Comparison between MQTT and AMQP

Criteria	**MQTT**	**AMQP**
Messaging scenarios (10%)	Basic messaging scenarios only (5)	Rich set of messaging scenarios (8)
Framing (10%)	Stream-oriented approach (7)	Buffer-oriented approach (7)
Connection security (10%)	Limited security support (5)	Comprehensive security support (8)
User security (10%)	User authentication exists but is weak (6)	Comprehensive security support (8)
Metadata (20%)	No metadata support (0)	Metadata enables deeper functionalities for routing, messaging, etc (8)
Reliability (20%)	Reliable with three levels of reliable message delivery (10)	Reliable, with two levels of reliable message delivery (8)
Discovery (10%)	Hierarchical topic space only (6)	Comprehensive mechanism of queues/namespaces (8)
Scalability (10%)	Very scalable (10)	Scalable (9)
TOTAL	5.9	**8**

The main characteristics that tip the balance in favor of a messaging protocol for a MEC platform are to have Universal support from existing Sensors and IoT frameworks, to be a lightweight protocol, and to be scalable for hierarchical data aggregation. It is important to have the possibility to merge and filter the data. AMQP is the protocol that best fits this case.

Besides that, a message broker solution must be used for handling the messaging protocol at the MEC. Apache Kafka, ActiveMQ and RabbitMQ are three broadly used messaging brokers that allow the exchange of messages between applications to be decoupled, making use of some pattern (Pub-Sub or Queues).

Apache Kafka is capable of processing messages and storing them with a publisher-subscriber model with high scalability and performance. It distributes the topics among the nodes through partitions to store the received events or messages. It combines the two messaging patterns described above, message queuing with publisher-subscriber, taking advantage of both. It is also responsible for guaranteeing the order of the messages for each consumer.

RabbitMQ is considered a more traditional messaging broker. It is based on the publisher-subscriber pattern, although it can handle communication synchronously or asynchronously, depending on the configuration. It also guarantees the delivery and order of messages between producers and consumers. ActiveMQ for its part is also a message broker supporting point-to-point and publish-subscribe messaging semantics. Standard messaging constructs like queues and topics are available for various messaging use cases.

The main differences can be found in Table 6

Table 6: Comparison between ActiveMQ, RabbitMQ and Kafka

Criteria	**ActiveMQ**	**RabbitMQ**	**Kafka**
Clients (20%)	Java, C, C++, Python, etc (9)	Java, C, C++, Python, etc (9)	Java, C, C++, Python, etc (9)
Protocols (20%)	AMQP, MQTT, REST, WebSockets (9)	AMQP, MQTT, STOMP, WebSockets (8)	Binary over TCP (9)
Synchronous / Asynchronous (20%)	Both (10)	Both (10)	Asynchronous (8)
Persistence (20%)	Yes (8)	Persist messages until they are dropped on the acknowledgement of receipt (7)	Persists messages with an option to delete after a retention period (10)
Failover / HA (20%)	Yes (10)	Yes (10)	Yes (10)
TOTAL	**9.2**	8.8	**9.2**

Although the three technologies provide equal security, scalability, and transaction capabilities, things get different when we narrow down specific use cases. ActiveMQ and RabbitMQ have been established as enterprise message brokers, providing reliable delivery guarantees across business applications. They are configured to acknowledge each message by default and have built-in recovery mechanisms. Also, they provide interfaces to popular messaging protocols, allowing more integration points with existing enterprise systems.

Kafka is an event streaming platform designed to meet different expectations. Kafka has the edge over former technologies in performance, message retention, and total ordering. It allows low-latency and high throughput write performances at scale. Its partition-based design enables a higher read capacity and a strict ordering of messages. Therefore, Kafka is ideal for building applications that demand more scalability, performance, message ordering, and longer retention periods.

Considering this conclusion, ActiveMQ has been chosen as the message broker for the MEC platform. Nevertheless, Kafka is a very interesting solution for a cloud platform managing dataflows coming from different MECs.

8. Analysis of risks

In this chapter, the possible risks that can influence both the planning and the execution of the project itself will be reviewed. The identification of these risks provides security for their management.

The chapter will be divided into two sections. The first will be based on a description of each risk that may influence the project, and also contingency measures are defined with the purpose of minimizing the impact of these risks. Subsequently, each of the identified risks will be evaluated by employing a risk matrix in which the probability and impact of the same will be related.

8.1 Description of the risks and contingency measures

Four possible risks have been identified that may affect the project's development. Therefore, this risk analysis has been carried out in order to quantify the effect that these risks would have if they were to occur. The identified risks are:

8.1.1 R1: Deviation from schedule

This risk is a management risk and occurs when the deadlines foreseen in the initial planning are exceeded. It is a risk with an unlikely probability of happening since this project includes a learning period of complex softwares, each one with its characteristics and properties. This learning period is a long and complex period. Therefore, slowing down in this process may mean misaligning all the planned planning. In order to avoid a significant delay in the planification, it is considerable to invest time in making a proper work planning.

8.1.2 R2: Variance against budget

This risk is a management risk and occurs when the budget proposed for the project is exceeded. It is a risk with a rare probability of happening since the equipment to be used is already available and the software to be used is mostly open source. Therefore, there are few expenses that can really divert the project from the proposed budget. However, if it does happen, it may have a moderate impact on the project. Depending on the case, it could force the modification of some element of the approach, design or deployment of the project.

8.1.3 R3: Attacks affect the production network

This risk occurs when attacks against the production servers affect other users of the servers. It is considered that it has a relatively unlikely probability of happening, since different security mechanisms are protecting the servers. At the same time, it is considered that, if it occurs, the impact would be extreme, since it could affect other users outside the project. The consequences of this risk could lead to various problems such as the denial of services or loss of information, among others. To avoid this risk, you must be careful with the deployment of the components. Designing them in such a way that they do not pose a real risk. Additionally, an attempt should be made to separate the production and development environments. To carry out this action, there is the possibility of creating independent tenants, which have the same objective.

8.1.4 R4: Hardware or physical infrastructure failure

This risk occurs when a hardware or physical infrastructure failure appears, such as in the electrical network, communications network, or equipment used. These failures can be derived from specific events such as power outages, a saturation of computing capacity, etc. The probability of these types of risks happening is very rare. However, the repercussions of these events could have a major impact. These types of failures are totally unrelated to the project and, therefore, no action can be taken to reduce the probability or prevent them from happening. Nevertheless, you can invest in reducing the high impact they would have. Through backup and redundancy techniques, the impact of these failures can be minimized and, should they occur, normal production can be recovered in a matter of a few moments.

8.2 Risk probability-impact Matrix

In this section, the risks defined in the previous section will be evaluated using the impact probability matrix. In this matrix, both the probability and the impact will be segmented into 5 different classes. In addition, a color code will be used to see in which risk zone each identified risk falls. The probability-impact matrix is shown in Table 7:

Table 7: Risk probability-impact Matrix

		Impact				
		Trivial	Minor	Moderate	Major	Extreme
Probability	Rare			R2	R4	
	Unlikely		R1			R3
	Moderate					
	Likely					
	Very likely					

9. Description of the solution

As already stated in the Background section [2], this platform is being deployed and operated in the Horizon 2020 project 5GMETA. In the next section, the high level of the framework will be presented as well as the general platform operation. In the following sections, the modules developed during the course of this thesis and the contributions to the European project will be presented and how they integrate within the architecture.

9.1 High-level architecture

The high-level reference architecture defined in this section defines the main common layers on which all the framework's building blocks sit, which will leverage the development and deployment of third-party applications. To accomplish the objectives, this architecture needs, through its design principles, to address the requirements stated in the previous chapter [6].

Before introducing the high-level architecture, it is important to define some terms that will be common and repeatedly employed in this chapter:

- **Dataflow:** is a unique flow/stream of produced data that is shared by Sensors and Devices (S&D) through a Pipeline in a MEC.
- **Data-type:** is a type of data that S&D shares and that a Pipeline in a MEC accepts and handles.
- **Pipeline:** is a group of Modules (containers) running in a MEC to handle a specific data-type. Each module can process the data received by subscribing to a queue/topic and generate output data in a new queue/topic into the MEC's message broker.
- **Module:** is a container running in a MEC that handles a specific data-type or group of data-type (e.g. C-ITS messages).
- **Third-party application:** is an application/service lying outside the platform, which acts as a third-party, consuming the available data, and optionally pushing events to the platform (e.g., a CCAM application).

Figure 5 outlines the reference architecture of the framework. It represents on a high-level the different layers through which the collected data will travel, from the Sensors and Devices (bottom layer), where it is generated, to reach the third-party applications and services (upper layer), where it is made available to be used by platform users. The 5G network is the enabler of the general platform. 5G features exploited include network

aspects, such as faster communication between nodes in the network and lower latency, but also a newer software point of view in which the network itself and the network functions are virtualized. The blocks in red are those that have been developed during the thesis. The architecture embodies four layers:

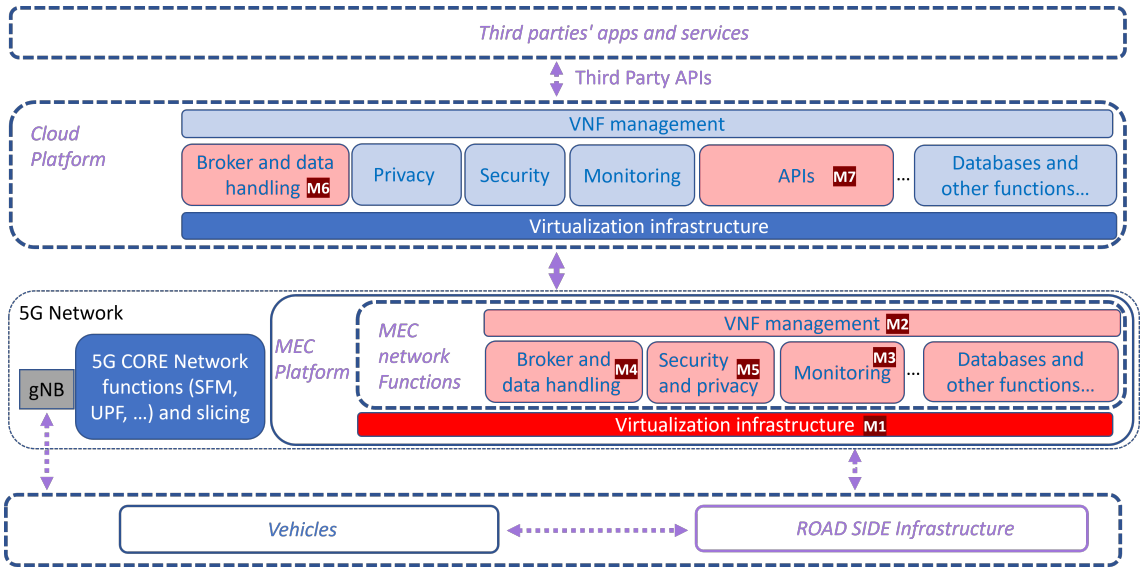


Figure 5: High-level architecture of the platform

- **Sensors and Devices:** This layer revolves around the core idea that at the bottom of the network lie sensors, devices or equipment that produce data, generally mounted on autonomous/connected vehicles or on Road Side Infraestructure (RSI), and which act/mimic the behavior of a regular UE connected to a cellular network. To that end, vehicles embedding such sensors will rely on On-Board-Units (OBUs) computers to exchange the generated data with the surroundings, and which allow short-range communication with other OBUs (within the same vehicle or other intelligent vehicles) or with Road Site Units (RSUs) or with the network [74] [75].

  Moreover, within this context RSUs play a major role thanks to their various capabilities and functionalities, by offering an entry point, or instead by allowing car data offloading [76], whether to overcome limited vehicle storage or to share warning messages such as accidents [77]. Furthermore, in some cases, RSUs are also equipped with sensors (e.g. Laser Imaging Detection and Ranging (LIDAR), camera) that allow not only to multicast data but also to generate it. Therefore, RSUs are also considered as Sensors & Devices.

  All the data generated by them will be forwarded to the MEC infrastructure through a 5G connection.

- **5G Network and MEC infrastructure:** Represents the main 5G core functions, 5G New Radio, and the platform's MEC system. This layer is mainly based on the 5G features that allow data to be routed with minimal latency to the edge platform. It also provides a virtualization Infrastructure connected to the Base Station of the 5G infrastructure. This layer is directly connected to the cloud platform.

  The edge platform plays an intermediary role between the vehicles and the cloud platform. The edge system is in the proximity of the end-users, collecting raw data generated by sensors and including secure and private pipelines. Moreover, it includes some computational capabilities, such as filtering, pre-processing, and

data collection before sending them to the cloud. It brings flexible computation and privacy capabilities closer to the end-users. The system architecture is composed of a virtualized infrastructure, a management level and a host level as detailed after in the chapter.

- **Cloud Platform:** This layer addresses the data and resource management aspects and requirements and, allows users to access the data. It also bundles third-party APIs. The cloud platform provides the necessary computing, network, and especially storage functionalities for the collected data. It has a twofold objective. On the one hand, it is the entry point for users that interact with the platform. On the other end, the platform cooperates with the 5G network and the MEC platform to create the pipelines that will provide the needed data flows with its given Service Level Agreement (SLA).
- **Third-party services:** Where the applications that have authorized access to the data sit.

Each layer of the platform addresses a set of requirements defined in the previous chapter. Later in the chapter, a deep dive is taken into the developed module, analyzing its components and the requirements that the component addresses. The platform is composed mainly of the MEC and Cloud platforms.

The module stack of the MEC Platform to provide a Virtualization Infrastructure connected to the Base Station of the 5G infrastructure is composed by:

- **VNF management** function to manage the different virtual network functions to be deployed. It is in charge of deploying data pipelines embodied in VNFs. The pipelines are VNFs specialized in a specific data-type, i.e., the processing of C-ITS messages. Each of them has different processing, but the objective is the same: privacy, interoperability, computing, and security functions.
- **Broker and data handling** function to receive data and register data flows. It manages the data transition between the S&D and the MEC platform.
- **Security** function to protect the MEC platform and the deployments that occur inside.
- **Privacy** function to anonymize the different data-types.
- **Monitoring** function to gather metrics, analyze and monitor the platform, evaluate the platform and optimize the resources and the applicable SLAs. available.
- **Databases** storing the current data and assets configuration.

The software stack of the cloud platform to host centralized systems, where the Virtualization management will be based on a commercial solution to facilitate the integration from third parties and the visibility/reachability from each MEC infrastructure, is composed by:

- **VNF management** function to orchestrate the VNFs in the different MEC platforms connected to the Cloud.
- **APIs** function, where are stored the different APIs to allow third parties ask for data consumption and to produce events or notifications to S&D connected to a MEC.

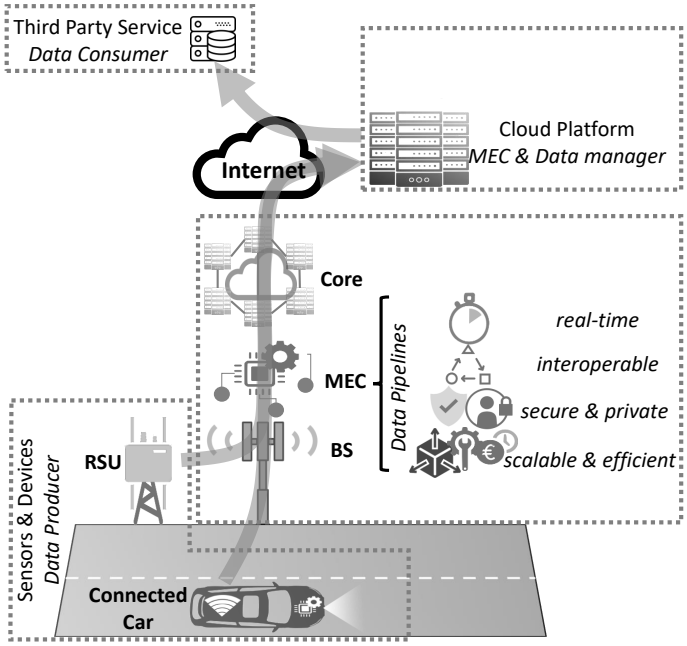


Figure 6: Platform's automotive data workflow

- **Broker and data handling** function to manage the data transition between the MEC and the cloud platforms. Data filtering.
- **Security** function to protect the Cloud platform. Authentication, Authorisation and Accounting (AAA) protocols. API Gateway to control access to third-party APIs. User identity management.
- **Privacy** function to anonymize the data flowing in the cloud.
- **Monitoring** function to gather metrics, analyze and monitor the platform, allowing the evaluation of the platform and optimization of the resources.
- **Databases** storing the current data producers' availability, consumer subscriptions and assets item inventory. Catalogs to allow producers to browse and configure the target data-types and the relevant locations.

9.1.1 General operation of the platform

The proposed approach designs a platform that: i) performs data processing in the MEC based on consumer demand; ii) enables filtering on geographical and data-types criteria to get just relevant data; and iii) applies SLA levels to different asset allocation profiles.

As depicted in Figure 6, the described framework is an open data-centric IoT live messaging platform for CAM services and applications where the security, privacy, scalability, interoperability, and licensing features are provided by the 5G networks functions executed at the edge to gain zero latency, capillarity, and geo-driven networking.

Concerning the data workflows and permitted directions, both upload and download directions are considered.

- **Upload:** This is the main data flow delivering data from sensors, vehicles, and RSUs to CAM services. When a CAM application selects a data-type and a ROI (Region Of Interest), the containers of a pipeline processing the specific data-type are deployed in the selected MECs. The data is only processed in the MEC and forwarded to the Cloud when a CAM service has selected a specific data-type from a serving MEC.
- **Download:** Spontaneous, discrete, and lightweight alarms and notifications sent as broadcast messages from CAM applications to Sensors and Devices subscribed to the notification data-type. This means no need for a specific function or container to deliver download messages.

So, the upload direction provides continuous live data feeds, while the latter is mainly designed to broadcast notifications and updates to all subscribed systems in a MEC.

The access level of the third-party application to the data available within the platform may differ depending on the application requirements.

Applications will mainly access the data through the cloud platform. After allocating the required computing assets for the pipelines to process raw data from sensors and devices at the edge resulting data is forwarded to the cloud. However, if a specific application has strict requirements when it comes to latency, data can be directly reached from the MEC platform in real-time.

For handling ROI and geolocations, Microsoft's Bing Maps Tile System, where each region is represented by a single tile of the same shape and size, and geographical indexing with quadkeys is used [78]. A quadkey number, a one-dimensional array that combines zoom level, column, and row information, uniquely identifies a single tile position. The tiles and quadkey filtering permit data consumers to browse and quickly filter the locations where data is being produced. Every MEC also registers the tile where it is located and serves to data producers so that they can find the serving MEC infrastructure and do a handover across serving MECs as the vehicle moves.

The deployment of pipelines in the MEC is triggered by data consumers (i) interested in processing and consuming a specific data-type from a particular location (ii). When a CAM application selects a data-type and if the SLA of the CAM application supports the required resources, the containers of a pipeline processing the specific data-type are deployed in the corresponding MEC(s) (iii). The data is only processed in the MEC and forwarded to the Cloud when a CAM service has selected a specific data-type from a serving MEC; otherwise, data pushed to the MEC is immediately discarded.

The platform should aid in delivering real-time data pipelines for car-captured and generated data through data privacy, interoperability, computing, and security functions embodied in VNFs deployed in a NFV-enabled architecture.

9.2 Implementation of the modules

In the previous section, the high-level architecture of the platform was presented. In figure 5 we can see that some of the modules have been marked in red and that have an identifier to be able to be identified in a simple way. In this section, we will analyze and explain the implementation and technical deployment of each of the modules marked in red.

9.2.1 M1: MEC Virtualization infrastructure

In the designed architecture, containers, specifically, Docker containers, have a prominent role. Docker runtime technology enables running an application in a lightweight, standalone, executable software package or container. Secondly, Kubernetes will be used as container orchestration framework for managing the platform's workloads and services. It will handle the scheduling of containers in a compute cluster. and thanks to horizontal and vertical auto-scaling, the scale of the resources for data pipelines will be performed, always according to the selected SLA level.

Figure 7 illustrates the MEC platform architecture, where the different industry solutions that build the platform stack are specified.

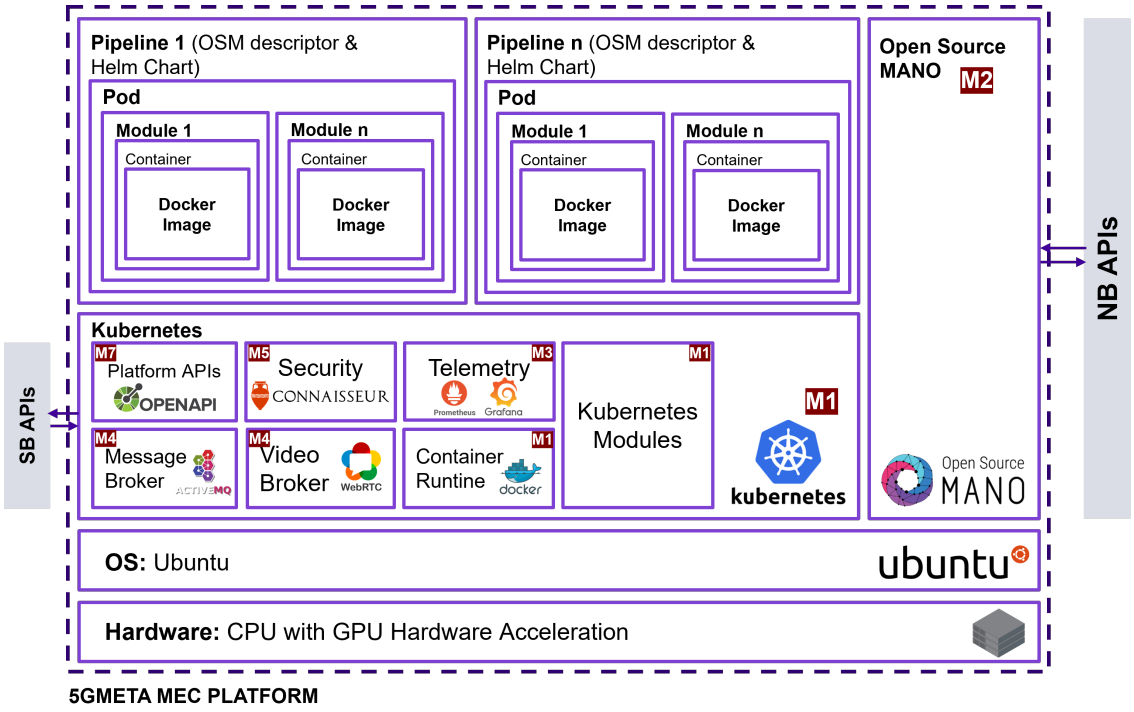


Figure 7: Architecture of the MEC platform

Kubernetes installation lets you make a lot of decisions. The decisions made to best fit with the requirements of the platform are the following:

- Ubuntu 18.04 or 20.04 Operating System (OS). The reason for choosing this SO will be explained in M2 section.
- Docker as the container runtime.
- Flannel for networking. No particular reason, open-source and most widely deployed.
- OpenEBS for storage. No particular reason, open-source and most widely deployed.
- MetalLB for load-balancing. No particular reason, open-source and most widely deployed.
- Single node. For quick deployment and easier management

Ansible will be used for MEC provisioning. Ansible is a software tool that provides simple but powerful automation for application deployment, server provisioning, updates on servers, configuration management and service orchestration. Thanks to Ansible, all the components forming the Edge Stack will be easily deployed. An Ansible playbook has been developed, and through it, the virtualization infrastructure and all the dependencies, components and modules necessary for the functioning of the MEC server will be installed. Furthermore, every time a change is made in the stack, it will be reflected in the playbook and running it again will let to update every MEC without extra effort.

The only requirement to deploy the stack is a clean Ubuntu 18.04 or 20.04 image with Ansible and some collections installed.

Listing 9.1: Python example

```
sudo apt-add-repository ppa:ansible/ansible
sudo apt update
sudo apt-get install curl python3-pip ansible
ansible-galaxy collection install community.general community.docker
    kubernetes.core
```

As for the minimum resources to be able to run the entire stack, 16 GB of RAM, 4 vCPUs and 60 GB of disk is at least necessary. The playbook will deploy the following components:

- Single-node Kubernetes cluster (v1.20.11)
  - Composed by Flannel CNI, OpenEBS Storage and MetalLB Load-balancer
- Docker Engine
- Helm k8s application manager
- Kube-prometheus-stack and Kube-eagle for k8s monitoring
  - Prometheus, Grafana and dashboards
- MySQL cluster for storing the databases
- Open Source MANO (OSM): v11 for Ubuntu 20.04 and v10 for Ubuntu 18.04
- Notary and Connaisseur for managing security in the cluster
- Developed MEC APIs

The ansible playbook is organized into tasks. First, it will install the necessary system dependencies and then, the installation of the virtual infrastructure will be performed. Finally, the rest of the components will be deployed. In the next code block, we can see the tasks for installing the system packages, docker and Kubernetes binaries, the cluster initialization and configuring it as a single node cluster.

Listing 9.2: Python example

```
- name: Install required system packages
become: yes
apt:
  name: "{{ packages }}"
```

```
  state: present
  update_cache: yes
vars:
  packages:
  - <packages_list>

- name: Install docker binaries
become: yes
apt:
  name: "{{ packages }}"
  state: present
  update_cache: yes
vars:
  packages:
  - docker-ce
  - docker-ce-cli
  - containerd.io

- name: Install kubernetes binaries
become: yes
apt:
  name: "{{ packages }}"
  state: present
  update_cache: yes
vars:
  packages:
  - kubelet=1.20.11-00
  - kubeadm=1.20.11-00
  - kubectl=1.20.11-00

- name: Initialize the cluster
become: yes
shell: "{{ item }}"
with_items:
  - 'echo "apiVersion: kubeadm.k8s.io/v1beta2\nkind:
     ClusterConfiguration\nnetworking:\n  podSubnet: 10.244.0.0/16\
     napiServer:\n  extraArgs:\n    service-node-port-range:
     "80-65535"" > {{ ansible_env.HOME }}/5gmeta/tmp/cluster-config.
     yaml'
  - kubeadm init --ignore-preflight-errors=SystemVerification --config
     {{ ansible_env.HOME }}/5gmeta/tmp/cluster-config.yaml > {{
     ansible_env.HOME }}/5gmeta/logs/cluster_init
args:
  creates: /etc/kubernetes/admin.conf

- name: Taint master
shell: "{{ item }}"
with_items:
  - kubectl taint node $(kubectl get nodes | awk '$3~/master/'| awk '{
     print $1}') node-role.kubernetes.io/master:NoSchedule-
  - touch {{ ansible_env.HOME }}/5gmeta/logs/master_tainted
args:
  creates: "{{ ansible_env.HOME }}/5gmeta/logs/master_tainted"
```

The entire playbook can be found in Annex A.1.

Once deployed, all the virtualization infrastructure and the platform modules will be

deployed and accessible for communication with S&D and the cloud platform and its lifecycle management.

9.2.2 M2: MEC VNF Management

OSM, the management and orchestration software stack that follows the standards defined by ETSI for NFV technology, carries out the task of orchestrating the specific data pipelines.

The default installer of OSM deploys a standalone Kubernetes on a single host, and OSM on top of it. As already stated, the MEC platform is based on a single node under Kubernetes technology, and it is not possible to install OSM in an already Kubernetes cluster. Furthermore, The MEC platform should be fully configurable and the Kubernetes scenario that deploys the OSM installer is not configurable. To solve that problem, a custom OSM installer has been deployed, mostly based in the original one. It can be found in Annex A.2.

Once the server is installed, it must be linked to the Kubernetes cluster. Both the installation and the linking to the cluster or other configuration operations are performed by the Ansible playbook presented in M1 [A.1].

After the installation is finished, the different configuration tasks have been completed, the platform is ready to orchestrate real-time data pipelines for car-captured and generated data wrapped in VNFs. As already explained in the section before, the virtualization technology chosen is Kubernetes, for container-based microservices operation. OSM can onboard and instantiate container-based VNFs, also knowns as Kubernetes-based Network Functions (KNFs) into a Kubernetes cluster using Helm Charts or Juju Bundles.

The design and development of a data pipeline have three main steps.

1. **Development of Docker image(s):** For the processing of different data-types, a Docker image must be developed, which contains all the logic for privacy, interoperability and computing functions.
2. **Development of a Helm chart:** A Helm chart is a collection of files that describes the behavior and configuration of a microservice in a Kubernetes cluster. The helm chart will reference the Docker designed in the previous step and will be configured for its deployment in the K8s cluster.
3. **Development of OSM descriptors:** Kubernetes Network Function Descriptor (KNFD) together with a Network Service Descriptor (NSD) must be designed. The name of the descriptors is tied to the data-type to be processed. Through these descriptor packages, the description of the process of deploying, configuring, and managing a KNF instance is fixed. An example of some descriptors for "cits" data-type can be found in Annex A.3.

Once all the steps for pipeline development have been done, each component must be stored in a repository respectively. Also, OSM descriptors must be added to the orchestrator. After that, the MEC is ready for pipeline orchestration. All the orchestration operations will be done through OSM API.

In Fig 8 MEC's OSM interface is shown, which contains different data-type specific pipelines. Secondly, in Listing 9.3, the API call schema for deploying a data pipeline is

outlined. It will be deployed in a unique K8s namespace, belonging to the user making the call.

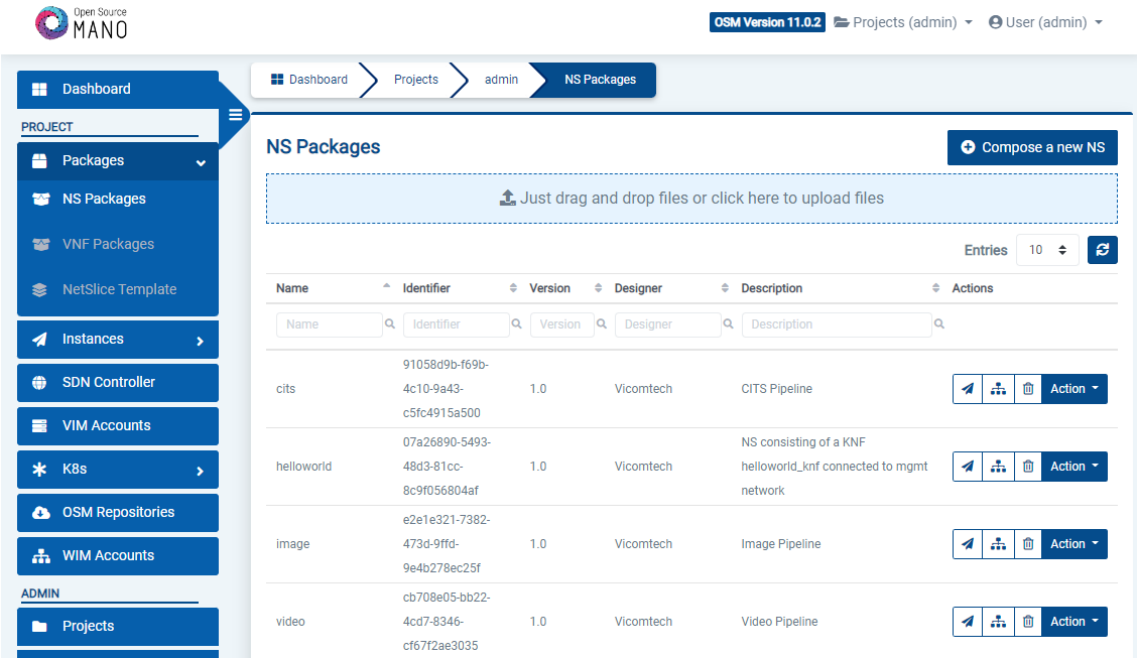


Figure 8: OSM interface

Listing 9.3: OSM API call to deploy a pipeline

```
url = 'https://' + orchestrator_ip + ':9999/osm/nslcm/v1/
    ns_instances_content'
headers = {'Content-Type': 'application/json', 'Authorization': bearer}
data = '{ "nsName": "' + payload["data_type"] + '", "nsdId": "' +
    datatype_response[datatype_index]["_id"] + '", "vimAccountId": "' +
     vim_response[vim_index]["_id"] + '", "additionalParamsForVnf": [ {
     "member-vnf-index": "1", "additionalParamsForKdu": [ { "kdu_name":
     "' + payload["data_type"] + '", "k8s-namespace": "' + payload["
    client_name"] + '", "kdu-deployment-name": "' + payload["
    client_name"] + '-' + payload["data_type"] + '-' + str(uuid.uuid1()
    )[:8] + '" } ] } ] }'
requests.post(url, data=data, headers=headers, verify=False)
```

This API call will come from the cloud platform, when a third-party requests a data-type in a geographical location, where the MEC is serving. Then the deployment of pipelines in the MEC is triggered. In the example depicted in Figure 9, we see a deployed pipeline and the logical container composition of different VNFs providing specialized data functions.

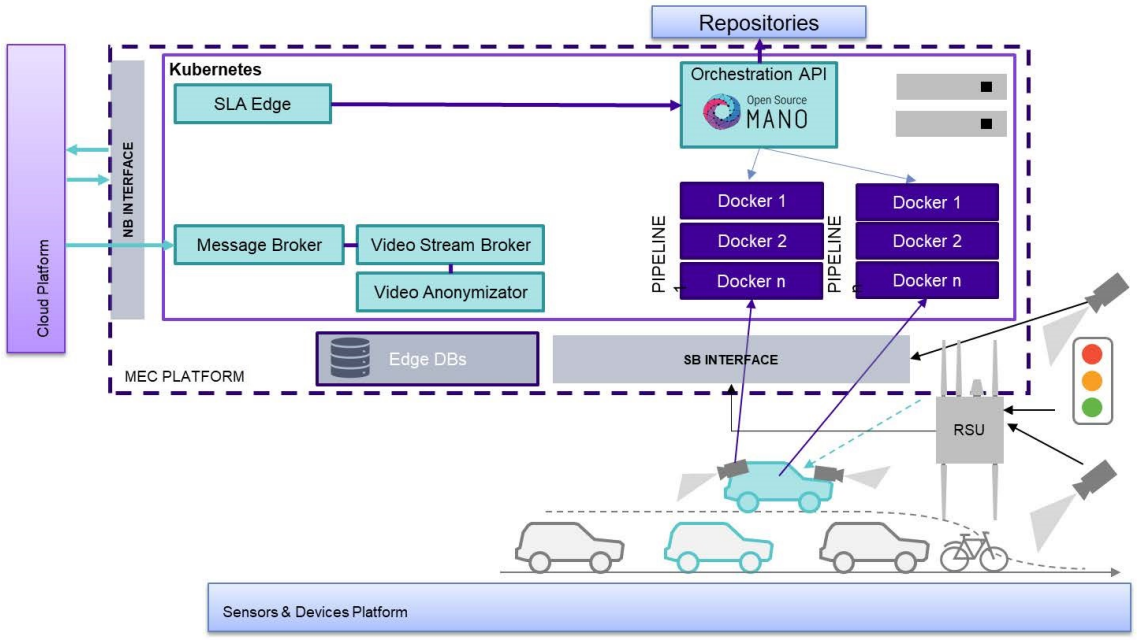


Figure 9: Pipeline deployment at a specific MEC infrastructure

9.2.3 M3: Monitoring

For platform infrastructure and modules monitoring two open-source solutions will be used. Kube Prometheus Stack and Kube Eagle. Kube Prometheus Stack is a pre-configured solution to collect metrics from all Kubernetes components. It includes a collection of Kubernetes manifests, Grafana dashboards, and Prometheus rules combined with scripts to provide easy-to-operate end-to-end Kubernetes cluster monitoring with Prometheus using the Prometheus Operator and Grafana visualization tool.

Additionally, Kube Eagle is a Prometheus exporter that exports various metrics of Kubernetes pod resource requests, limits and its actual usage. It provides a better overview of the Kubernetes cluster resources so that resource allocation can be optimized. It includes a Grafana dashboard for easier visualization.

Both components have been easily installed through helm charts. For example, for deploying Kube Prometheus Stack just the following lines of code are needed. These tasks are also made by the Ansible playbook [A.1].

Listing 9.4: OSM API call to deploy a pipeline

```
helm repo add prometheus-community https://prometheus-community.github.
    io/helm-charts
helm repo update
helm install kube-prometheus-stack prometheus-community/kube-prometheus
    -stack
```

Once deployed you can access the different dashboards through a web interface. Figure 10 and Figure 11 show the dashboards of each component where some cluster metrics are depicted.

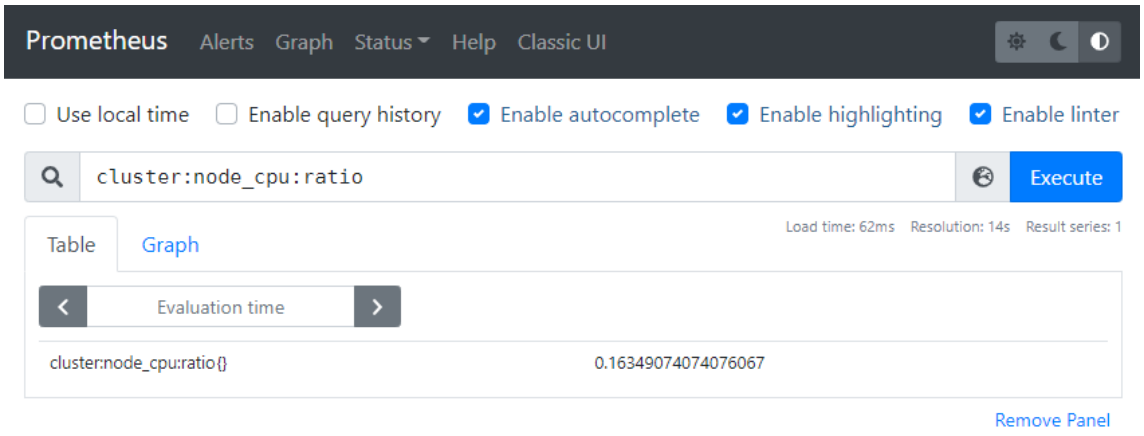


Figure 10: Prometheus interface

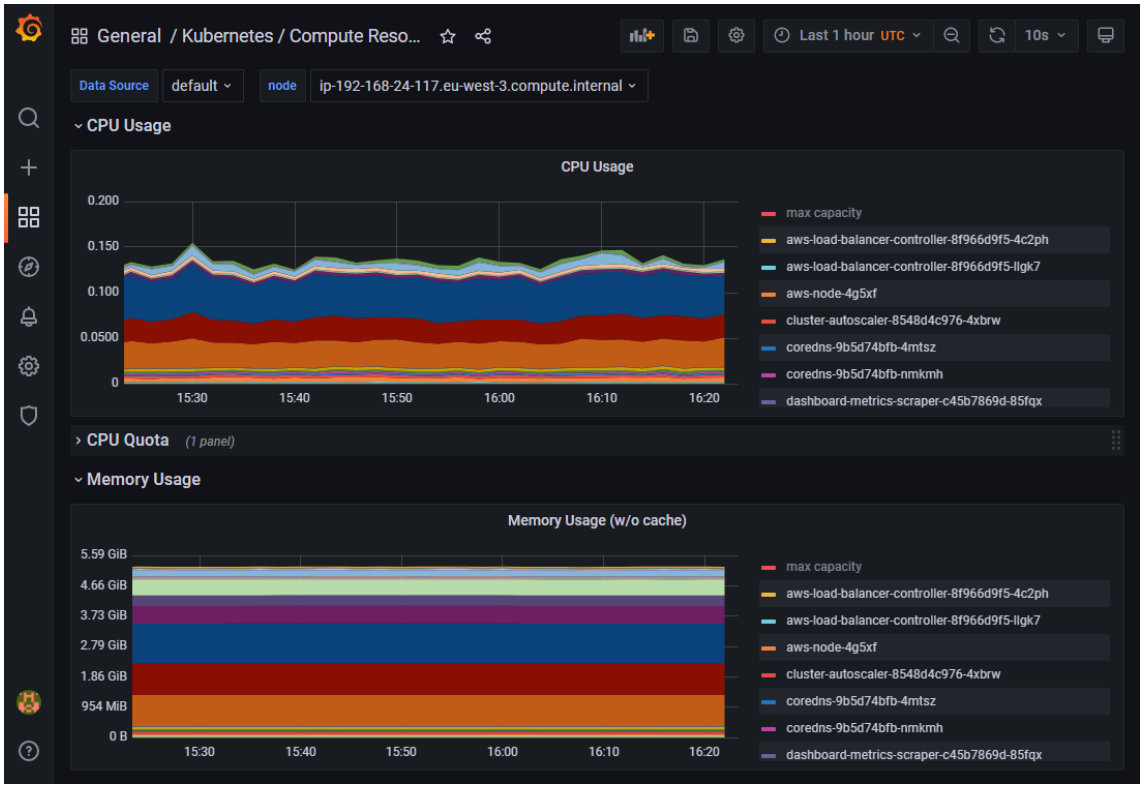


Figure 11: Grafana interface

As already stated previously in the thesis, the MEC will only deploy the data-type pipelines for the requested SLA if there are enough resources to comply with it. Thanks to the API of Prometheus, metrics can be easily obtained to compute the available resources, and let a pipeline to be deployed. This operation can be done through a bash script that can be found in Annex A.4. The Listing 9.5 shows how different metrics are gathered through Prometheus API, just as the script does.

Listing 9.5: Metric gathering through Prometheus API

```
allocatable_memory=$(curl -s http://$nodeip:9090/api/v1/query\?query\=
    eagle_node_resource_allocatable_memory_bytes | jq '.data.result[0].
    value[1] | tonumber')
```

```
memory_usage=$(curl -s http://$nodeip:9090/api/v1/query\?query\=
    eagle_node_resource_usage_memory_bytes | jq '.data.result[0].value
    [1] | tonumber')
memory_requests=$(curl -s http://$nodeip:9090/api/v1/query\?query\=
    eagle_node_resource_requests_memory_bytes | jq '.data.result[0].
    value[1] | tonumber')
memory_limits=$(curl -s http://$nodeip:9090/api/v1/query\?query\=
    eagle_node_resource_limits_memory_bytes | jq '.data.result[0].value
    [1] | tonumber')
```

9.2.4 M4 & M6: MEC and Cloud Broker and data handling

The data pipelines deployed in a MEC server consume raw data samples from IoT frameworks and transfer them to the cloud infrastructure. For managing all the data coming from S&D a message broker is needed. The technology chosen for message handling is ActiveMQ as specified in Section 7. ActiveMQ will be the central IoT messaging technology in the platform, which has universal support from existing Sensors and IoT frameworks.

The different data producers will push the data into MEC servers through the AMPQ protocol. Then the on-demand instantiated pipeline will process the selected data and put it again into the message broker. Finally, the data will be transferred to the cloud platform so that a third-party can access it. The solution chosen for operating data in the cloud is Kafka as it is ideal for working with a huge amount of data, is very scalable and has very stable performance.

In a nutshell, module M4 will manage the data in the MEC via ActiveMQ and module M6 will be responsible for handling all the data coming from all the servers, thanks to the Kafka platform.

Both solutions work with publish and subscribe messaging. In a publish and subscribe message system, producers send messages on a topic. In this model, the producer is known as a publisher and the consumer is known as a subscriber. One or many publishers can publish on the same topic, and a message from one or many publishers can be received by many subscribers. Subscribers subscribe to topics, and all messages published to the topic are received by all subscribers on the topic.

The platform will play with the different data-types arriving on the platform and with the topics created in the both message brokers. When any producer starts pushing a data-type to a MEC server, a topic with the data-type name will be created in the message broker. When a request coming from a third-party is made, a connection between the MEC and the cloud will be opened, and through a Kafka connector, AMQP all the requested data topics will be retrieved and forwarded to the cloud into the same Kafka topics. Finally, the data from those topics will be aggregated and filtered into a private topic created exclusively for the user's request.

For M4 implementation the latest official ActiveMQ has been used. As there was not any official Helm chart for deploying the message broker, a chart has been developed for the easy deployment into a Kubernetes cluster. On the other hand, to deploy the M6 or Kafka platform, the Confluent solution has been chosen to deliver the distribution of Kafka in a customizable way. Again, it has been deployed through the official Confluent Helm chart. The configurations used can be found in Annex A.5.

Five Kafka modules has been used for building the cloud's data management system.

1. **Kafka Broker and Kafka Zookeper**, the basic modules for operating with Kafka.
2. **Kafka Connect**, the tool for providing a scalable and reliable way to copy data between Kafka and other datastore. Through this component, a Kafka Connector is configured for the retrieval of AMQP topics from ActiveMQ and forwarding them to the same Kafka topics in the Cloud.
3. **Kafka Registry**, used transparently by the Connector, the consumer and the producer to validate the schema of the messages in Avro. Avro is the Serialization and Deserialization method for the messages in the platform.
4. **KSQLdb**, a streaming SQL engine that enables real-time data processing against Kafka. Is used to provide specific topics after the data has been filtered for data consumers.

Finally, figure 12 depicts a sequence diagram to clearly understand the functioning of the modules on how the data flows between the components.

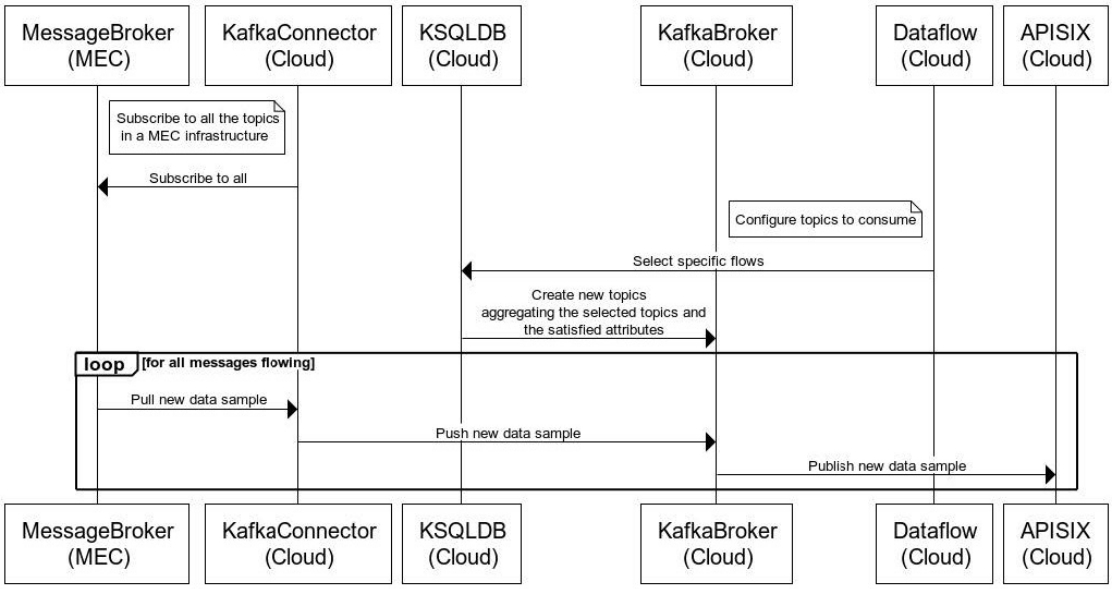


Figure 12: Data handling modules sequence diagram

9.2.5 M5: MEC Security

One of the main tasks of the MEC is to provide common data pipelines which connect data producers and consumers. All the pipelines are based on microservices packaged in Docker images. Every time a third-party request a data-type, a pipeline will be instantiated in the MEC. Furthermore, the platform offers the possibility to host third-party CAM services that require real-time processing. For that reason, all the containers need further verification to be trusted and be deployed inside the infrastructure to avoid malicious software and to guarantee that a Docker image has not been modified.

To verify the containers being deployed in a Kubernetes cluster, a solution called Connaisseur has been implemented. Connaisseur is an admission controller to integrate container image signature verification and trust pinning into a K8s cluster. It ensures the

integrity and provenance of container images. To do so, it intercepts resource creation or update requests sent to the Kubernetes cluster, identifies all container images and verifies their signatures against pre-configured public keys. Based on the result, it either accepts or denies those requests.

All the containers imaged behind the pipelines must be signed after being built by employing a pair of private/public certificates. If not, the pipeline deployment will be denied. All public keys of the repositories permitted by the platform will be added to Connaisseur. For doing that a validator has to be created in the configuration files of Connaisseur. An example of a validator can be shown in Listing 9.6.

Listing 9.6: Connaisseur validator schema for trusting a repository

```
- name: default
type: notaryv1  # or other supported validator (e.g. "cosign")
host: notary.docker.io # configure the notary server for notaryv1 or
   rekor url for cosign
trust_roots:
- name: My repository
  key: |
    -----BEGIN PUBLIC KEY-----
    XXX==
    -----END PUBLIC KEY-----
auth:
  username: My user
  password: My pass
```

This solution has been installed using the official Helm Chart. Annex A.6 contains the values files used for the implementation.

From other hand, networking is a vital component of any infrastructure, and having network policies to control communication is essential. By default, in a Kubernetes cluster, There are no network restrictions within pods, so every pod can automatically communicate with all other pods in the cluster. This isn't ideal from a security perspective and an external K8s plugin is needed to add extra networking policies.

This is where Calico comes in. Calico is an open-source networking and network security solution. It provides network connectivity and IP address management plugins, as well as additional network policies to control incoming (Ingress) and outgoing (Egress) traffic. Network policies are applied through Kubernetes manifest files. In the next Code Listing, an example of a network policy manifest is shown:

Listing 9.7: Calico network policy manifest file

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
  - Ingress
  - Egress
  ingress:
```

```
  - from:
    - ipBlock:
        cidr: 172.17.0.0/16
        except:
        - 172.17.1.0/24
    - namespaceSelector:
        matchLabels:
          project: myproject
    - podSelector:
        matchLabels:
          role: frontend
    ports:
    - protocol: TCP
      port: 6379
  egress:
  - to:
    - ipBlock:
        cidr: 10.0.0.0/24
    ports:
    - protocol: TCP
      port: 5978
```

Regarding the data privacy aspect, there are different modules that anonymize the data coming from S&D. For example in video streams, faces and vehicular plates are anonymized. For C-ITS messages, anonymization of specific fields of C-ITS will be performed. The data privacy feature is one of the main requirements that need to be guaranteed by the platform and it consists of protecting the sensitive information related to the user's identity.

Thanks to this module, the integrity of the MEC server and the privacy of the users can be ensured, allowing only trusted entities to participate actively in the platform, avoiding any network intrusion and anonymizing all the ingested data.

9.2.6 M7: Cloud APIs

Cloud APIs are the APIs that will be used by developers to directly access the data flows offered by the different MEC servers. These APIs will be implemented using the OpenAPI approach, leveraging on its advantages. The access will be managed by an AAA module and an API gateway belonging to Cloud's Security block. The gateway serves as the single entry point for third-party users of the platform.

There are several key aspects to consider when implementing APIs:

- Quick Integration, meaning easy and universal integration using Web Technologies.
- Documentation, bringing documentation to the features provided, required inputs and potential outputs and formats.
- Access Control, to prevent unauthorized or unplanned connections are performed between local or remote systems.
- Web-based testing, providing a visual site to test an API to integrate and check responses, boosting the learning curve.

To satisfy all these requirements the specification of OpenAPIs 3.0 has been followed. To this end, we agreed on using Swagger tools for the fast prototyping and YAML declaration of the API. Then Flask will provide a visual front-end of the API to check the API in the development process of the backend and to understand how and an API works for the integration. OAUTH2.0 will be also used for the identification of actors and Keycloak for controlling and limiting access from other building blocks or systems.

The OpenAPI Specification (OAS) defines a standard, language-agnostic interface to RESTful APIs. Thanks to the OpenAPI definition, you can understand the capabilities of the service without access to source code or documentation, or through network traffic inspection. It gives the possibility to interact with the remote service with a minimal amount of implementation logic. Furthermore, an OpenAPI definition can then be used by documentation generation tools to display the API, code generation tools to generate servers and clients in various programming languages, testing tools, and many other use cases. It follows the API First approach. It is an approach to designing every API around a contract written in an API description language. This contract ensures the robustness, consistency, reusability and abstraction from specific implementations.

Swagger is a set of open-source software tools to design, build, document, and use RESTful web services. Swagger includes automated documentation, code generation (into many programming languages), and test-case generation.

Flask is a micro web framework written in Python that provides useful tools and features that make creating web applications in Python easier and accessible for new developers. It provides only the necessary tools, but it extends its functionality with additional libraries and frameworks.

The summary of the different stages to be accomplished when designing, developing and integrating a new API is depicted in Figure 13.

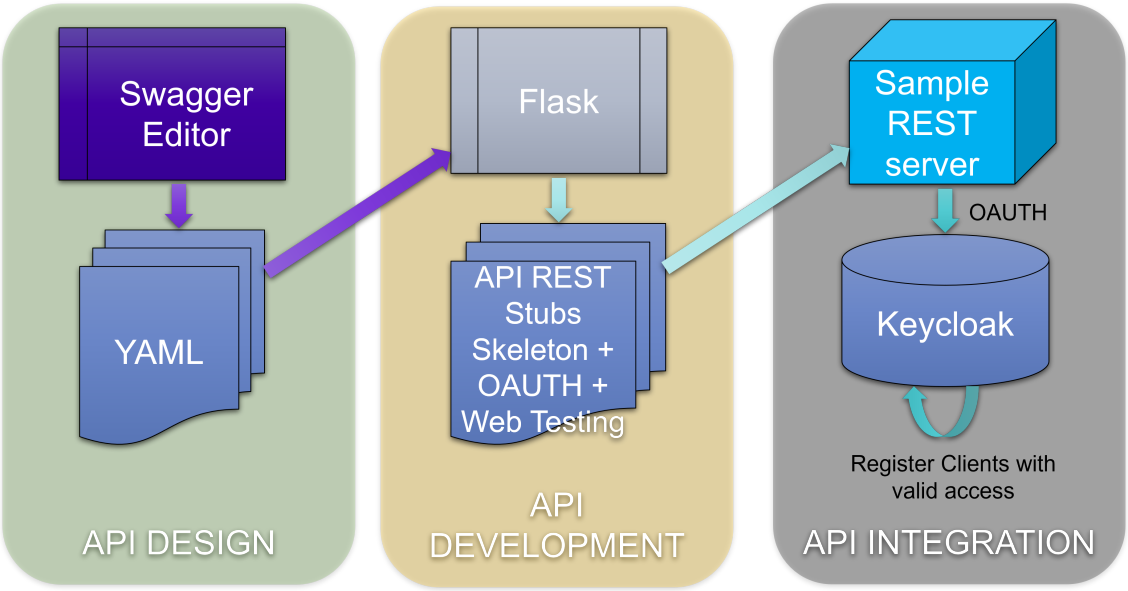


Figure 13: APIs stages and key technologies and formats

The API should use the Status-Line part of the HTTP response to inform the client of their request's result. The status codes are divided into five categories:

- **1xx:** Informational, communicates transfer protocol-level information.

- **2xx:** Success, indicates that the client's request was accepted successfully.
- **3xx:** Redirection, indicates that the client must take some additional action in order to complete their request.
- **4xx:** Client Error, this category of error status codes points the finger at the client.
- **5xx:** Server Error, the server takes the responsibility for these error status code.

Different APIs have been developed for module M7. That module contains all the APIs that the third-party users will use to make requests to the platform, to get dataflows, or send events and notifications for example. The API developed during this thesis is for managing SLAs.

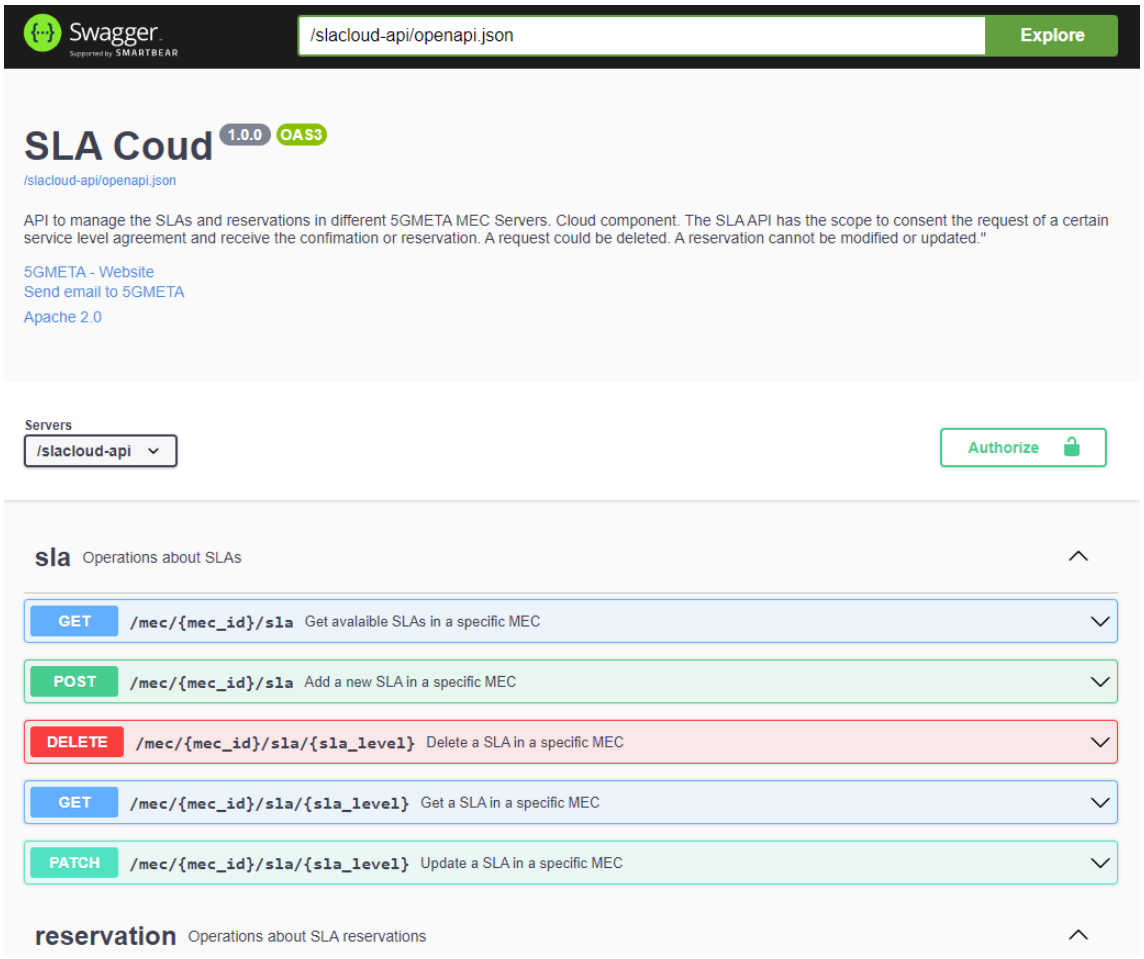


Figure 14: SLA API Swagger User Interface

Before requesting a data-type in the cloud, is necessary to select an SLA level, depending on the data that is selected. Each data type would need different computing resources to cope with incoming traffic. The number of assigned resources for a pipeline depends on the applicable SLA. For example, a GPU-enabled SLA is available through GPU virtualization in the MEC for processing and optimizing capabilities to process live video. When an SLA is requested, the API will check if there are available resources in the MEC, using the M3 module, to deploy the data-type pipeline for the requested SLA. If available, the resources are reserved, and the request will be forwarded to the OSM Orchestration API for the deployment of the pipeline.

Endpoints provided by the API:

- **/sla** POST: Add a new SLA
- **/sla** GET: Get SLAs
- **/sla /{slalevel}** GET: Find SLA by ID
- **/sla /{slalevel}** PATCH: Update a SLA
- **/sla /{slalevel}** DELETE: Delete a SLA
- **/reservation** POST: Make a SLA reservation
- **/reservation** GET: Get reservations
- **/reservation** /{reservationid} GET: Find reservation by ID
- **/reservation** /{reservationid} DELETE: Delete a SLA reservation

The OpenAPI definition and the python code of the API controllers can be found in Annex A.7.

10. Validation of the solution

This section covers the checking of different modules of the architecture's requirements (declared in Section 6). In this way, the platform implementation ensures that no feature or corner is not considered. As the project on which the submitted thesis was based is still active, the validation will verify compliance with the platform requirements. With the correct fulfillment, we will finish the validation of the work. In the future, different Key Performance Indicator (KPIs) will be defined to demonstrate how effectively the platform is achieving the objectives, but it is outside the scope of the work.

Table 8: Requeriments validated by the platform

Tier	Requirement	Req ID	Validated by Module	Means of verification
MEC	Use of VNF	REQ01	M1 M2	OSM MANO permits the use of VNFs that will be deployed in Docker/K8s virtualization technology.
	Datatype Pipeline execution	REQ02	M1 M2	OSM manages the lifecycle of the pipelines. Different VNF packages tied to specific datatype are executed using OSM API.
	Pipeline allocation	REQ03	M1, M2, M3 M5	The monitoring module permits the resources and SLA check before allocating a pipeline. If enough, the security module guarantees that the signature of the pipeline container is valid.
	Data Processing Scalability	REQ04	M1, M4	Active MQ providse hierarchical messaging structures to connect multiple live dataflows.
	Scalability at MEC level	REQ05	M1	Kubernetes allows the horizontal scale based on the application requirements, which may change over time.
	Uplink and Downlink dataflows	REQ06	M4	ActiveMQ dedicated topics permit the uplink and downlink dataflows
	Intra services flows confidentiality	REQ07	M4	Calico ensures the limited access to internal dataflows.
	Specialised Processing Capacity	REQ08	M1, M2 M3	The resources reservation of a certain SLA level assures the VNFs have dedicated resources for the data proccessing.
Cloud	Resources allocation on MEC	REQ12	M7	The SLA API permits to create the requested data pipelines.
	Data Interoperability	REQ15	M6	Kafka ecosystem technologies provide a scalable and reliable way to copy data between the cloud and the MEC. M6 module retrieves AMQP topics from a MEC and forwards all the topics and data into the same Kafka topics in the Cloud.
	Uplink and Downlink dataflows	REQ18	M6	Kafka Connector will open the connections for uplink and downlink dataflows.
	API format	REQ25	M7	OpenAPI specification standard REST APIs define the structure and syntax of every API.
	API data broker	REQ28	M6	Kafka API permits the connection to the broker.
	API data gateway	REQ29	M7	Kafka API guarantees a single endpoint where all the dataflows are signaled and provided.

11. Description of tasks

This project has been divided into different work packages or phases that will be made up of different tasks that, in their global computation, will form the planning that has been followed.

To carry out the tasks detailed below, the work team has been made up of two positions, with different roles and responsibilities as stated in Table 9:

Table 9: Work team

Name	Responsibility	Played role
Jasone Astorga	Senior Engineer	Director of the project
Maider Huarte	Senior Engineer	Co-director of the project
Mikel Serón	Junior Engineer	Developer of the project

The project director and co-director will be in charge of the management, direction and coordination of the project, guiding and supervising the junior engineer while keeping focused on the objectives set out in this Master's Final Project. On the other hand, the junior engineer will be in charge of carrying out the development of work and its respective report.

The Work Packages (WPs), and their corresponding tasks, into which the thesis has been divided are detailed below:

- **WP1 - Project management, coordination and documentation:** This management phase is essential in any type of project. All the tasks that make up this WP are executed continuously during the entire development of the project. Writing of the results and the thesis. WP duration: 180 hours.
  - **T1.1 - Management and coordination:** The management and coordination of the project is a task that focuses on ensuring that the project's development complies with the stipulated planning. It seeks to verify that the development reaches the different milestones set. Another objective of this task is to deal with or avoid any deviation in the planning. Task duration: 60 hours.
  - **T1.2 - Project definition:** In this task, the definition of the tasks that this project will fulfill are defined. Task duration: 10 hours.
  - **T1.3 - Review:** In this task, a follow-up is carried out to verify that the established deadlines and tasks are met. The work done is reviewed. Task duration: 30 hours.
  - **T1.4 - Design of the structure of the document and LaTex template:** In this phase, a LaTeX template is designed and created so that the writing of the thesis is carried out in a homogeneous and simple way.

  - **T1.5 - Writing of the thesis:** This task consists of writing down the progress and results that are being achieved during the development process of the other phases. This way, a reliable record can be kept that the requirements are met, and the objectives are achieved. Additionally, it also consists of writing the final report during the course of the project and at the end of it.

- **WP2 - Analysis of the background and state of the art:** This WP involves a study of the context and the related work. In a nutshell, technologies, projects and publications whose aim resembles the project's scope. WP duration: 100 hours.

  - **T2.1 - Analysis of the background:** This task will analyze the context in which the project is framed. Task duration: 25 hours.
  - **T2.2 - Analysis of the state of the art in Edge computing:** This task involves a study of the related work, projects and publications whose aim resembles the project's scope. Task duration: 25 hours.
  - **T2.3 - Analysis of open-source Edge computing frameworks:** Research on the existing open-source Edge computing frameworks. Task duration: 25 hours.
  - **T2.4 - Analysis of the link of the different use cases to existing solutions:** Analyze the requirements of the use cases in connected vehicles (input from the 5GMETA European project) and understand their implications. Task duration: 25 hours.

- **WP3 - Analysis of requirements:** In this WP, an exhaustive analysis of the requirements for the proposed platform is carried out. WP duration: 50 hours.

  - **T3.1 - Identification of gaps and requirements for the platform solution:** The gaps of the analyzed solutions are analyzed, and different requirements that the platform to be developed must meet are set. Task duration: 50 hours.

- **WP4 - Proposal of an open-source Edge computing framework for automotive data processing:** In this work package, the solution to be developed during the project is designed, taking into account the results of the analysis of requirements and alternatives. WP duration: 250 hours.

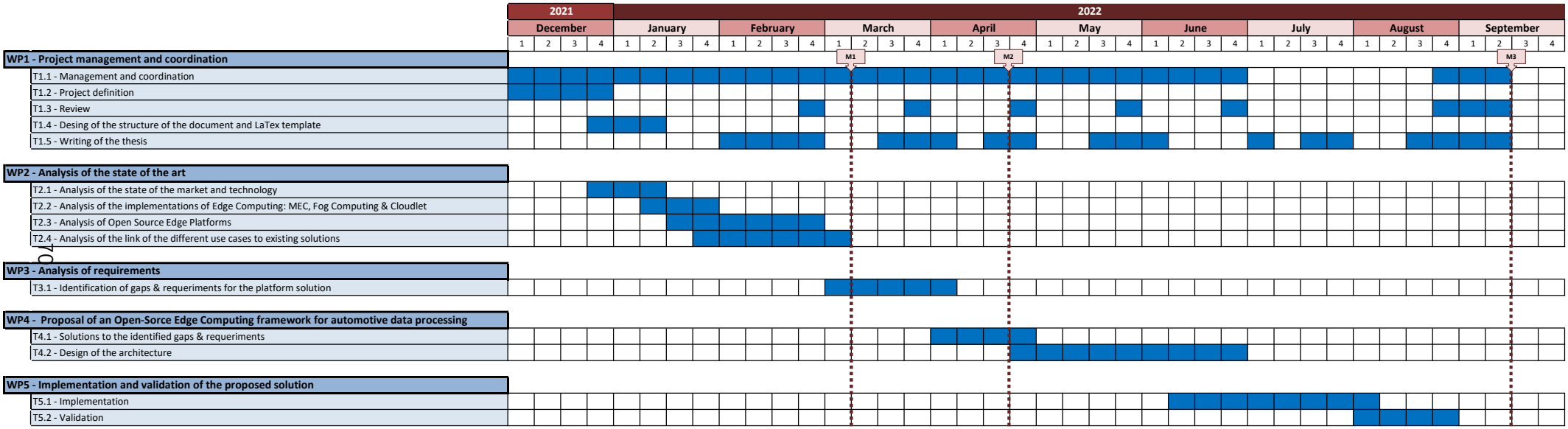
  - **T4.1 - Solutions to the identified gaps and requirements:** Analysis and proposal of different components that aim to resolve all the gaps and requirements identified. Task duration: 80 hours.
  - **T4.2 - Design of the architecture:** Basic and high-level design of the usage architecture. Design the architecture, with its components. Task duration: 170 hours.

- **WP5 - Implementation and validation of the proposed solution:** This work package is mainly based on the implementation of the solution. Once having implemented the solution of the project, it must be checked that the system works properly according to its design and specifications. WP duration: 120 hours.

  - **T5.1 - Implementation:** Implementation of the solution according to the proposed solution. Configuration of equipment, deployment of infrastructure and development of modules.
  - **T5.2 - Validation:** Validation of all the requirements identified in WP3. Task duration: 20 hours.

Below is the Gantt chart in which the development of the project is graphically observed over time and the duration of each task carried out during this project. The Gantt Chart is a graph that allows a clear and complete visualization of the location of activities over time. It is in the form of a table, in which each column represents a unit of time, and each row an activity.

11.1 Gantt diagram



M1: Thesis final definition
M2: First draft of the proposal
M3: Thesis submission

12. Description of the budget

In this section, the costs involved in carrying out this study will be presented in detail.

12.1 Manpower

To carry out this master's thesis, two senior engineers and a junior engineer have been necessary as human resources. The hourly rate of each of them is:

Table 10: Human resources

Name	Position	Hourly Rate (€/h)
Jasone Astorga (JA)	Project director, senior engineer	60
Maider Huarte (MH)	Project co-director, senior engineer	60
Mikel Serón (MS)	Junior engineer	20

The hours worked per person for each work package are detailed below:

Table 11: Cost of human resources

Work Package (WP)	Responsible	Hours	Hourly Rate (€/h)	Cost (€)
WP1 - Project management, coordination and documentation	JA, MH, MS	100 (JA & MH) 80 (MS)	60 20	6000 1600
WP2 - An alysis of the state of the art	MS	100	20	2000
WP3 - Analysis of requirements	MS	50	20	1000
WP4 - Proposal of an Open-Sorce Edge Computing framework for automotive data processing	MS	250	20	5000
WP5 - Implementation and validation of the proposed solution	MS	120	20	2400
TOTAL:		100 (JA & MH) 600 (MS)		**18000**

As can be seen, the final price of human resources throughout the entire job is €18,000.

12.2 Amortizable costs

In this section, all the material that has been used in this project will be taken into account. The time of use of the material has been approximately 9 months. We can classify amortizable material into two categories, software and hardware expenses.

- **Software expenses:** In order to continue with the Open-source software philosophy, we have proceeded to use totally free tools available to any developer. Therefore, the expense of software is non-existent.
- **Hardware expenses:** We proceed to calculate the expenses incurred in terms of hardware. For the development of this project, the following hardware has been used:

Table 12: Amortizable costs

Name	Initial value (€)	Useful life (Months)	Usage time (Months)	Cost (€)
Laptop	850	48	9	159,375
Server	3200	48	9	355,555
TOTAL:				**514,93**

It can be concluded that the cost of amortizable material in relation to this thesis amounts to 514,96€.

12.3 Non-amortizable costs

In addition to internal hours and materials, expenses that are not directly related to the project, such as electricity, internet and cloud provider expenses, must be included. The price per kilowatt hour of the electricity rate contracted at the workplace amounts to 0,104€/hour. Also an uninterrupted use of the internet has been made. The internet rate in the workplace amounts to 38€/month. Finally, the costs for deploying the cloud platform in AWS cloud provider amounts to 90€/month. The solution has been deployed during 2 months.

Table 13: Non-amortizable costs

Description	Cost (€)
Office supplies	30
Electricity expenses	72,8
Internet expenses	342
Cloud provider expenses (AWS)	180
TOTAL:	**624,8**

In summary, the sum of the indirect expenses is 624,8€.

12.4 Total cost

The total cost of the development is the sum of the totals of each subsection, that is, manpower, amortizable costs and non-amortizable costs.

Table 14: Total cost

Expense type	**Expense incurred**
Manpower	18000
Amortizable costs	514,93
Non amortizable costs	624,8
TOTAL:	**19139,73**

13. Conclusions and future work

The CAM applications will benefit from the performance, reliability, and capacity promised by 5G cellular networks. Furthermore, the connected car data is expected to unlock new business for traditional actors and new entrants on top of innovative CAM applications. Here, the MEC architectures from 5G networks can fuel the creation of a common data marketplace where data owners produce and share their data and data services consume the available data demanding the processing of common pipelines in the MEC which deal with interoperability, privacy, and efficiency.

The most relevant requirements have been analyzed before specifying the reference architecture, its application interfaces and its network interfaces. Then, a deeper look was provided on how these requirements were addressed through the different layers of the architecture and their building blocks.

This project proposes an architectural approach designed and implemented to facilitate share of automotive data. Furthermore, some novel features provided by the proposed architecture, when compared to market solutions, are the separation of data producers and consumers through dedicated IoT topics where pipelines read raw samples and write processed data, the allocation of computing resources adapted to the data consumption demand and the hosting of services which require low latency in MEC infrastructures applying the container-based life cycle management for pipelines to third-party services.

This work has fed both the European 5GMETA project and a publication [79].

In terms of efficiency, there is a lot of work to do to get the most from the computing resources used by the running pipelines, reusing processed data for different applications with different sampling rates from different SLAs and with partially overlapped ROIs while preventing promiscuous access to data.

Furthermore, the application of Network Slicing technologies to the proposed architecture and data workflows will isolate different data flows for specific applications. Specifically, applying RAN slicing techniques is essential to prioritize data communications for safety-related CAM applications over non-real-time-sensitive ones.

Bibliography

[1] Marconi's first wireless transmission. https://www.ool.co.uk/blog/marconis-first-wireless-transmission/. [Online; accessed 17-January-2022].

[2] The invention of mobiles phones. https://www.sciencemuseum.org.uk/objects-and-stories/invention-mobile-phones. [Online; accessed 18-January-2022].

[3] How many phones are in the-world? https://www.bankmycell.com/blog/how-many-phones-are-in-the-world. [Online; accessed 18-January-2022].

[4] ITU. Setting the scene for 5g: opportunities and challenges. *Technical report, ITU.*, 2018.

[5] What edge computing means for infrastructure and operations leaders. https://www.gartner.com/smarterwithgartner/what-edge-computing-means-for-infrastructure-and-operations-leaders. [Online; accessed 21-January-2022].

[6] Ashkan Yousefpour, Caleb Fung, Tam Nguyen, Krishna Kadiyala, Fatemeh Jalali, Amirreza Niakanlahiji, Jian Kong, and Jason P. Jue. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *Journal of Systems Architecture*, 98:289–330, 2019.

[7] Julien Gedeon, Florian Brandherm, Rolf Egert, Tim Grube, and Max Mühlhäuser. What the fog? edge computing revisited: Promises, applications and future challenges. *IEEE Access*, 7:152847–152878, 2019.

[8] Ericsson mobility report. https://www.ericsson.com/en/mobility-report/reports/november-2019/iot-connections-outlook. [Online; accessed 19-January-2022].

[9] 5GAA. Toward fully connected vehicles: Edge computing for advanced automotive communications. *White Paper, 5GAA.*, 2014.

[10] Development in the mobility technology ecosystem—how can 5g help? https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/development-in-the-mobility-technology-ecosystem-how-can-5g-help. [Online; accessed 21-January-2022].

[11] M. E. Lesk and B. W. Kernighan. Computer typesetting of technical journals on unix. pages 879–888, June 1977. 1977 American Federation of Information Processing Societies National Computer Conference, AFIPS 1977 ; Conference date: 13-06-1977 Through 16-06-1977.

[12] 5gmeta project's website. https://5gmeta-project.eu/. [Online; accessed 16-September-2022].

[13] Salah Eddine Elayoubi, Mikael Fallgren, Panagiotis Spapis, Gerd Zimmermann, David Martín-Sacristán, Changqing Yang, Sébastien Jeux, Patrick Agyapong, Luis Campoy, Yinan Qi, and Shubhranshu Singh. 5g service requirements and operational use cases: Analysis and metis ii vision. In *2016 European Conference on Networks and Communications (EuCNC)*, pages 158–162, 2016.

[14] Diego Kreutz, Fernando M. V. Ramos, Paulo Esteves Veríssimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2015.

[15] Bo Yi, Xingwei Wang, Keqin Li, Sajal K. Das, and Min Huang. A comprehensive survey of network function virtualization. *Comput. Networks*, 133:212–262, 2018.

[16] T.D. Nadeau and K. Gray. *SDN: Software Defined Networks.* O'Reilly, 2013.

[17] Iqbal Alam, Kashif Sharif, Fan Li, Zohaib Latif, M. M. Karim, Sujit Biswas, Boubakr Nour, and Yu Wang. A survey of network virtualization techniques for internet of things using sdn and nfv. *ACM Comput. Surv.*, 53(2), apr 2020.

[18] ETSI. Etsi gs nfv 002 v1.2.1 (2014-12), "network functions virtualisation (nfv); architectural framework". *Technical report, ETSI, Tech. Rep.*, 2014.

[19] ETSI. Network functions virtualisation (nfv) release 3; management and orchestration; vi-vnfm reference point - interface and information model specification. *Technical report, ETSI, Tech. Rep.*, 2018.

[20] Rashid Mijumbi, Joan Serrat, Juanluis Gorricho, Niels Bouten, Filip De Turck, and R. Boutaba. Network function virtualization: State-of-the-art and research challenges. *IEEE Communications Surveys Tutorials*, 18, 09 2015.

[21] Openvim. https://osm.etsi.org/gitweb/?p=osm/openvim.git. [Online; accessed 12-April-2022].

[22] Omar Sefraoui, Mohammed Aissaoui, and Mohsine Eleuldj. Openstack: Toward an open-source solution for cloud computing. *International Journal of Computer Applications*, 55:38–42, 10 2012.

[23] Open infrastructure foundation. https://openinfra.dev/. [Online; accessed 12-April-2022].

[24] Teodora Sechkova, Michele Paolino, and Daniel Raho. Virtualized infrastructure managers for edge computing: Openvim and openstack comparison. In *2018 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*, pages 1–6, 2018.

[25] Open source mano (osm). https://osm.etsi.org/. [Online; accessed 13-April-2022].

[26] Open network automation platform (onap). https://www.onap.org/. [Online; accessed 13-April-2022].

[27] Christos John Bouras, Panagiotis Ntarzanos, and Andreas Papazois. Cost modeling for sdn/nfv based mobile 5g networks. *2016 8th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*, pages 56–61, 2016.

[28] Juliver Gil Herrera and Juan Felipe Botero. Resource allocation in nfv: A comprehensive survey. *IEEE Transactions on Network and Service Management*, 13(3):518–532, 2016.

[29] Abderrahime Filali, Amine Abouaomar, Soumaya Cherkaoui, Abdellatif Kobbane, and Mohsen Guizani. Multi-access edge computing: A survey. *IEEE Access*, 8:197017–197046, 01 2020.

[30] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: Enabling innovation in campus networks. *Computer Communication Review*, 38:69–74, 04 2008.

[31] Rob Enns, Martin Bjorklund, and Juergen Schoenwaelder. Netconf configuration protocol. Technical report, RFC 4741, December, 2006.

[32] Pankaj Berde, Matteo Gerola, Jonathan Hart, Yuta Higuchi, Masayoshi Kobayashi, Toshio Koide, Bob Lantz, Brian O'Connor, Pavlin Radoslavov, William Snow, and Guru Parulkar. Onos: Towards an open, distributed sdn os. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, page 1–6, New York, NY, USA, 2014. Association for Computing Machinery.

[33] Jan Medved, Robert Varga, Anton Tkacik, and Ken Gray. Opendaylight: Towards a model-driven sdn controller architecture. In *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*, pages 1–6, 2014.

[34] FUJITA Tomonori. Introduction to ryu sdn framework. *Open Networking Summit*, pages 1–14, 2013.

[35] Ryan Wallner and Robert Cannistra. An sdn approach: Quality of service using big switch's floodlight open-source controller. *Proceedings of the Asia-Pacific Advanced Network*, 35:14, 06 2013.

[36] Ola Salman, Imad H. Elhajj, Ayman Kayssi, and Ali Chehab. Sdn controllers: A comparative study. In *2016 18th Mediterranean Electrotechnical Conference (MELECON)*, pages 1–6, 2016.

[37] Carla Mouradian, Diala Naboulsi, Sami Yangui, Roch H. Glitho, Monique J. Morrow, and Paul A. Polakos. A comprehensive survey on fog computing: State-of-the-art and research challenges. *IEEE Communications Surveys Tutorials*, 20(1):416–464, 2018.

[38] Blesson Varghese, Nan Wang, Sakil Barbhuiya, Peter Kilpatrick, and Dimitrios S. Nikolopoulos. Challenges and opportunities in edge computing. In *2016 IEEE International Conference on Smart Cloud (SmartCloud)*, pages 20–26, 2016.

[39] Mahadev Satyanarayanan, Paramvir Bahl, Ramon Caceres, and Nigel Davies. The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Computing*, 8(4):14–23, 2009.

[40] Yujin Li and Wenye Wang. The unheralded power of cloudlet computing in the vicinity of mobile devices. In *2013 IEEE Global Communications Conference (GLOBECOM)*, pages 4994–4999, 2013.

[41] Url. `https://www.sigmobile.org/mobicom/2011/vanet2011/program.html`. [Online; accessed 30-March-2022].

[42] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, page 13–16, New York, NY, USA, 2012. Association for Computing Machinery.

[43] Koustabh Dolui and Soumya Kanti Datta. Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing. In *2017 Global Internet of Things Summit (GIoTS)*, pages 1–6, 2017.

[44] Rehmat Ullah, Muhammad Atif Ur Rehman, and Byung-Seo Kim. Design and implementation of an open source framework and prototype for named data networking-based edge cloud computing system. *IEEE Access*, 7:57741–57759, 2019.

[45] ETSI. Mobile-edge computing – introductory technical white paper. *White Paper, ETSI*, 2019.

[46] Multi-access edge computing (mec). `https://www.etsi.org/technologies/multi-access-edge-computing`. [Accessed 15-July-2022].

[47] ETSI. Etsi gs mec 003 v2.2.1 (2020-12), "multi-access edge computing (mec); framework and reference architecture". *Technical report, ETSI, Tech. Rep.*, 2020.

[48] ETSI. Etsi gs mec 001 v2.1.1 (2019-01), "multi-access edge computing (mec); terminology". *Technical report, ETSI, Tech. Rep.*, 2019.

[49] ETSI. Etsi gs mec 002 v1.1.1 (2016-03), "mobile edge computing (mec); technical requirements". *Technical report, ETSI, Tech. Rep.*, 2016.

[50] ETSI. Etsi gs mec 009 v2.2.1 (2020-10), "multi-access edge computing (mec); general principles, patterns and common aspects of mec service apis". *Technical report, ETSI, Tech. Rep.*, 2020.

[51] ETSI. Etsi gs mec 010-2 v2.1.1 (2019-11), "multi-access edge computing (mec); mec management; part 2: Application lifecycle, rules and requirements management". *Technical report, ETSI, Tech. Rep.*, 2019.

[52] ETSI. Etsi gs mec 013 v2.1.1 (2019-09), "multi-access edge computing (mec); location api". *Technical report, ETSI, Tech. Rep.*, 2019.

[53] ETSI. Etsi gr mec 022 v2.1.1 (2018-09), "multi-access edge computing (mec); study on mec support for v2x use cases". *Technical report, ETSI, Tech. Rep.*, 2018.

[54] ETSI. Etsi gs mec 030 v2.1.1 (2020-04), "multi-access edge computing (mec); v2x information service api". *Technical report, ETSI, Tech. Rep.*, 2020.

[55] What is edge computing? components, examples, and best practices. `https://www.spiceworks.com/tech/edge-computing/articles/what-is-edge-computing/`. [Accessed 16-July-2022].

[56] Yanjun Shi, Yaohui Pan, Zihui Zhang, Yanqiang Li, and Yu Xiao. A 5g-v2x based collaborative motion planning for autonomous industrial vehicles at road intersections. In *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 3744–3748, 2018.

[57] European commission. cooperative, connected and automated mobility (ccam). `https://ec.europa.eu/transport/themes/its/c-its_en`. [Accessed 16-July-2022].

[58] Ccam, connected vehicles, c-its. https://www.mobilityits.eu/ccam-connected-vehicles. [Accessed 16-July-2022].

[59] Yancong Wang, Jian Wang, Yuming Ge, Bingyan Yu, Cheng Li, and Lu Li. Mec support for c-v2x system architecture. *2019 IEEE 19th International Conference on Communication Technology (ICCT)*, pages 1375–1379, 2019.

[60] Claudia Campolo, Antonio Iera, Antonella Molinaro, and Giuseppe Ruggeri. Mec support for 5g-v2x use cases through docker containers. In *2019 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6, 2019.

[61] Tiia Ojanperä, Hans van den Berg, Wieger IJntema, Ramon de Souza Schwartz, and Miodrag Djurica. Application synchronization among multiple mec servers in connected vehicle scenarios. In *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*, pages 1–5, 2018.

[62] Qingyang Zhang, Yifan Wang, Xingzhou Zhang, Liangkai Liu, Xiaopei Wu, Weisong Shi, and Hong Zhong. Openvdap: An open vehicular data analytics platform for cavs. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 1310–1320, 2018.

[63] Zhenyun Zhou, Houjian Yu, Chen Xu, Zheng Chang, Shahid Mumtaz, and Jonathan Rodriguez. Begin: Big data enabled energy-efficient vehicular edge computing. *IEEE Communications Magazine*, 56(12):82–89, 2018.

[64] Wei Duan, Xiaohui Gu, Miaowen Wen, Yancheng Ji, Jianhua Ge, and Guoan Zhang. Resource management for intelligent vehicular edge computing networks. *IEEE Transactions on Intelligent Transportation Systems*, 23(7):9797–9808, 2022.

[65] Jiawen Kang, Rong Yu, Xumin Huang, Maoqiang Wu, Sabita Maharjan, Shengli Xie, and Yan Zhang. Blockchain for secure and efficient data sharing in vehicular edge computing and networks. *IEEE Internet of Things Journal*, 6(3):4660–4670, 2019.

[66] Zhuoxing Qin, Supeng Leng, Jihu Zhou, and Sun Mao. Collaborative edge computing and caching in vehicular networks. In *2020 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6, 2020.

[67] Kubeedge. https://kubeedge.io/en/. [Online; accessed 04-February-2022].

[68] Afraino. https://www.lfedge.org/projects/akraino/. [Online; accessed 03-February-2022].

[69] Edgex. https://www.edgexfoundry.org/. [Online; accessed 07-February-2022].

[70] Kube5g. https://mosaic5g.io/kube5g/. [Online; accessed 08-February-2022].

[71] Starlingx. https://www.starlingx.io/. [Online; accessed 07-February-2022].

[72] Baetyl. https://baetyl.io/. [Online; accessed 08-February-2022].

[73] David Artuñedo Guillen, Bessem Sayadi, Pascal Bisson, Jean Phillippe Wary, Håkon Lonsethagen, Carles Antón, Antonio de la Oliva, Alexandros Kaloxylos, and Valerio Frascolla. Edge computing for 5g networks - white paper, March 2020.

[74] Matteo Petracca, Paolo Pagano, Riccardo Pelliccia, Marco Ghibaudi, Claudio Salvadori, and Christian Nastasi. On-board unit hardware and software design for vehicular ad-hoc networks. 2013.

[75] Abdallah Dabboussi, Raed Kouta, Jaafar Gaber, Bachar ElHassan, Maxime Wack, and Lina Nachabe. A new approach for the reliability of vehicular ad hoc networks. *SSRN Electronic Journal*, 01 2018.

[76] Sony Guntuka, Elhadi M. Shakshuki, Ansar Yasar, and Hana Gharrad. Vehicular data offloading by road-side units using intelligent software defined network. *Procedia Computer Science*, 177:151–161, 2020. The 11th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN 2020) / The 10th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH 2020) / Affiliated Workshops.

[77] Javier Barrachina, Piedad Garrido, Manuel Fogue, Francisco J. Martinez, Juan-Carlos Cano, Carlos T. Calafate, and Pietro Manzoni. Road side unit deployment: A density-based approach. *IEEE Intelligent Transportation Systems Magazine*, 5(3):30–39, 2013.

[78] Bing maps tile system. online: [`https://docs.microsoft.com/en-us/bingmaps/articles/bing-maps-tile-system`, howpublished = "`https://openinfra.dev/`", note = "[accessed 15-july-2022]"].

[79] Mikel Seron, Angel Martin, and Gorka Velez. Life cycle management of automotive data functions in mec infrastructures. In *IEEE Future Networks World Forum (FNWF)*, In press.

A. Annex I: Platform deployment scripts

Listing A.1: MEC instatiation ansible-playbook

```
---
- name: Deploy 5GMETA MEC server
  hosts: localhost
  vars:
    ansible_python_interpreter: /usr/bin/python3
# REQUIRED VARS
#    southbound_ip: "X.X.X.X" #IP for the communication with S&D
#    northbound_ip: "X.X.X.X" #Public IP for communication with the
    Cloud Platform
    tiles:
#      - 031333123201211
#      - 031333123201033
# OPTIONAL VARS
#    messagebroker_port: 5673 #Default 5673
#    videobroker_port: 8443 #Default 8443
#    registrationapi_port: 12346 #Default 12346
#    organization: Vicomtech
#    city: San Sebastian
#    latitude: "43.2922071"
#    longitude: "-1.987018,17"
  tasks:
  - name: Debug
    ansible.builtin.debug:
      msg:
      - "{{ ansible_env.HOME }}"
      - "{{ ansible_env.USER }}"

  # https://germaniumhq.com/2019/02/14/2019-02-14-Disabling-Swap-for-
     Kubernetes-in-an-Ansible-Playbook/
  - name: Save original iptables & fstab
    become: yes
    shell: "{{ item }}"
    with_items:
      - sudo cp /etc/fstab /etc/fstab_ORIGINAL
      - sudo iptables-save > /etc/iptables/iptables_ORIGINAL.txt

  - name: Install required system packages
    become: yes
    apt:
      name: "{{ packages }}"
      state: present
```

```
      update_cache: yes
    vars:
      packages:
      - apt-transport-https
      - ca-certificates
      - curl
      - gnupg
      - lsb-release
      - software-properties-common
      - build-essential
      - git
      - subversion
      - python3-dev
      - python3-docker
      - libcurl4-openssl-dev
      - libssl-dev

  - name: Upgrade pip
    pip:
      name: pip
      extra_args: --upgrade

  - name: Install python pip packages
    pip:
      name: "{{ packages }}"
    vars:
      packages:
      - kubernetes
      - openshift
      - pycurl
      - pyGeoTile
      - prettytable
      - click

  - name: Create 5gmeta directory
    file:
      path: "{{ ansible_env.HOME }}/5gmeta/"
      state: directory
      recurse: yes

  - name: Import needed files for stack deployment from 5gmeta's
     repositories
    subversion:
      repo: https://github.com/XXX/orchestrator/trunk/src/
      dest: "{{ ansible_env.HOME }}/5gmeta/"
      export: yes
      username: XXX
      password: XXX
      force: yes

  - name: Add docker gpg key
    become: yes
    apt_key:
      url: https://download.docker.com/linux/ubuntu/gpg
      state: present

  - name: Add docker apt repository
    become: yes
```

```
    apt_repository:
      repo: deb [arch=amd64] https://download.docker.com/linux/ubuntu
         xenial stable
      state: present

  - name: Install docker and its dependecies
    become: yes
    apt:
      name: "{{ packages }}"
      state: present
      update_cache: yes
    vars:
      packages:
      - docker-ce
      - docker-ce-cli
      - containerd.io
#      - docker-compose-plugin
#    notify:
#    - Docker started

  - name: Add current user to docker group
    become: yes
    user:
      name: "{{ ansible_env.USER }}"
      groups: docker
      append: yes

  - name: Modify docker mtu
    become: yes
    copy:
      dest: /etc/docker/daemon.json
      content: |
        {
          "mtu": 1450
        }

  - name: Add kubernetes gpg key
    become: yes
    apt_key:
      url: https://packages.cloud.google.com/apt/doc/apt-key.gpg
      state: present

  - name: Add kubernetes apt repository
    become: yes
    apt_repository:
      repo: deb http://apt.kubernetes.io/ kubernetes-xenial main
      state: present

  - name: Remove swapfile from /etc/fstab
    become: yes
    mount:
      name: "{{ item }}"
      fstype: swap
      state: absent
    with_items:
    - swap
    - none
```

```
  - name: Disable swap
    become: yes
    shell: swapoff -a
    when: ansible_swaptotal_mb > 0

  - name: Install kubernetes binaries
    become: yes
    apt:
      name: "{{ packages }}"
      state: present
      update_cache: yes
    vars:
      packages:
      - kubelet=1.20.11-00
      - kubeadm=1.20.11-00
      - kubectl=1.20.11-00
      # 1.23.3-00

  - name: Put kubernetes packages on hold
    become: yes
    shell: apt-mark hold "{{ packages }}"
    vars:
      packages:
      - kubelet
      - kubeadm
      - kubectl

  - name: Initialize the cluster
    become: yes
    shell: "{{ item }}"
    with_items:
      - 'echo "apiVersion: kubeadm.k8s.io/v1beta2\nkind:
         ClusterConfiguration\nnetworking:\n  podSubnet:
         10.244.0.0/16\napiServer:\n  extraArgs:\n    service-node-
         port-range: "80-65535"" > {{ ansible_env.HOME }}/5gmeta/tmp/
         cluster-config.yaml'
      - kubeadm init --ignore-preflight-errors=SystemVerification --
         config {{ ansible_env.HOME }}/5gmeta/tmp/cluster-config.yaml
         > {{ ansible_env.HOME }}/5gmeta/logs/cluster_init
    args:
      creates: /etc/kubernetes/admin.conf

  - name: Create .kube directory
    file:
      path: "{{ ansible_env.HOME }}/.kube"
      state: directory

  - name: Copy admin.conf to user's kube config
    become: yes
    copy:
      src: /etc/kubernetes/admin.conf
      dest: "{{ ansible_env.HOME }}/.kube/config"
      remote_src: yes
      owner: "{{ ansible_env.USER }}"
      group: "{{ ansible_env.USER }}"
      mode: '0600'

  - name: Taint master
```

```
    shell: "{{ item }}"
    with_items:
      - kubectl taint node $(kubectl get nodes | awk '$3~/master/'| awk
          '{print $1}') node-role.kubernetes.io/master:NoSchedule-
      - touch {{ ansible_env.HOME }}/5gmeta/logs/master_tainted
    args:
      creates: "{{ ansible_env.HOME }}/5gmeta/logs/master_tainted"

  - name: Install helm
    shell: curl https://raw.githubusercontent.com/helm/helm/master/
       scripts/get-helm-3 | bash
    args:
      creates: /usr/local/bin/helm

  - name: Download flannel cni pod network manifest
    get_url:
      url: https://raw.githubusercontent.com/coreos/flannel/master/
         Documentation/kube-flannel.yml
      dest: "{{ ansible_env.HOME }}/5gmeta/tmp/kube-flannel.yml"

  - name: Apply flannel cni pod network manifest
    k8s:
      state: present
      src: "{{ ansible_env.HOME }}/5gmeta/tmp/kube-flannel.yml"

  - name: Add openebs repo
    kubernetes.core.helm_repository:
      name: openebs
      repo_url: https://openebs.github.io/charts

  - name: Deploy openebs
    kubernetes.core.helm:
      name: openebs
      release_namespace: openebs
      create_namespace: true
      chart_ref: openebs/openebs
      chart_version: 3.1.0
      # 1.12.0
      update_repo_cache: yes

  - name: Define default storageclass
    shell: kubectl patch storageclass openebs-hostpath -p '{"metadata
       ":{"annotations":{"storageclass.kubernetes.io/is-default-class
       ":"true"}}}'
    register: result
    until: result.stdout.find("storageclass.storage.k8s.io/openebs-
       hostpath patched") != -1
    retries: 30
    delay: 10

  - name: Add metallb repo
    kubernetes.core.helm_repository:
      name: metallb
      repo_url: https://metallb.github.io/metallb

  - name: Deploy metallb
    kubernetes.core.helm:
      name: metallb
```

```
    release_namespace: metallb-system
    create_namespace: true
    chart_ref: metallb/metallb
    chart_version: 0.11.0
    update_repo_cache: yes
    values:
      configInline:
        address-pools:
        - name: default
          protocol: layer2
          addresses:
          - '{{ ansible_default_ipv4.address }}/32'

- name: Download kubernetes dashboard manifest
  get_url:
    url: https://raw.githubusercontent.com/kubernetes/dashboard/v2
       .4.0/aio/deploy/recommended.yaml
    dest: "{{ ansible_env.HOME }}/5gmeta/tmp/k8s-dashboard.yaml"

- name: Apply kubernetes dashboard manifest
  k8s:
    state: present
    src: "{{ ansible_env.HOME }}/5gmeta/tmp/k8s-dashboard.yaml"

- name: Download metrics-server manifest
  get_url:
    url: https://github.com/kubernetes-sigs/metrics-server/releases/
       latest/download/components.yaml
    dest: "{{ ansible_env.HOME }}/5gmeta/tmp/metrics-server.yaml"

- name: Modify metrics-server manifest
  lineinfile:
    path: "{{ ansible_env.HOME }}/5gmeta/tmp/metrics-server.yaml"
    insertafter: "        - --metric-resolution=15s"
    line: "        - --kubelet-insecure-tls"
    state: present

- name: Apply metrics-server manifest
  k8s:
    state: present
    src: "{{ ansible_env.HOME }}/5gmeta/tmp/metrics-server.yaml"

- name: Add prometheus-community helm repo
  kubernetes.core.helm_repository:
    name: prometheus-community
    repo_url: https://prometheus-community.github.io/helm-charts

- name: Deploy kube-prometheus-stack
  kubernetes.core.helm:
    name: prometheus-stack
    release_namespace: monitoring
    create_namespace: true
    chart_ref: prometheus-community/kube-prometheus-stack
    update_repo_cache: yes

- name: Expose grafana's dashboard
  k8s:
    state: present
```

```
      kind: Service
      namespace: monitoring
      name: prometheus-stack-grafana
      definition:
        spec:
          ports:
          - nodePort: 3000
            port: 80
          type: NodePort

  - name: Expose prometheus's dashboard
    k8s:
      state: present
      kind: Service
      namespace: monitoring
      name: prometheus-stack-kube-prom-prometheus
      definition:
        spec:
          ports:
          - nodePort: 9090
            port: 9090
          type: NodePort

  - name: Expose alertmanager's dashboard
    k8s:
      state: present
      kind: Service
      namespace: monitoring
      name: prometheus-stack-kube-prom-alertmanager
      definition:
        spec:
          ports:
          - nodePort: 9093
            port: 9093
          type: NodePort

  - name: Create 5gmeta namespace
    k8s:
      name: 5gmeta
      kind: Namespace
      state: present

  - name: Add kube-eagle helm repo
    kubernetes.core.helm_repository:
      name: kube-eagle
      repo_url: https://raw.githubusercontent.com/cloudworkz/kube-eagle
         -helm-chart/master

  - name: Deploy kube-eagle
    kubernetes.core.helm:
      name: kube-eagle
      release_namespace: monitoring
      chart_ref: kube-eagle/kube-eagle
      values:
        serviceMonitor:
          create: true
          releaseLabel: prometheus-stack
      update_repo_cache: yes
```

```
- name: Add bitnami repo
  kubernetes.core.helm_repository:
    name: bitnami
    repo_url: https://charts.bitnami.com/bitnami

- name: Deploy mysql cluster
  kubernetes.core.helm:
    name: mysql-cluster
    release_namespace: mysql
    create_namespace: true
    chart_ref: bitnami/mysql
    update_repo_cache: yes

- name: Add mysql-cluster secret to 5gmeta namespace
  shell: "{{ item }}"
  with_items:
    - "kubectl get secret mysql-cluster --namespace=mysql -o yaml |
       sed 's/namespace: .*/namespace: 5gmeta/' | kubectl apply -f
       -"
    - touch {{ ansible_env.HOME }}/5gmeta/logs/secret_copied
  args:
    creates: "{{ ansible_env.HOME }}/5gmeta/logs/secret_copied"

- name: Install osm 10 in ubuntu 18.04
  script: "{{ ansible_env.HOME }}/5gmeta/scripts/Osm10Install.sh > {{
      ansible_env.HOME }}/5gmeta/logs/osm_install"
  args:
    creates: "{{ ansible_env.HOME }}/5gmeta/logs/osm10_installed"
  when:
    - ansible_facts['distribution'] == "Ubuntu"
    - ansible_facts['distribution_major_version'] == "18"

- name: Install osm 11 in ubuntu 20.04
  script: "{{ ansible_env.HOME }}/5gmeta/scripts/Osm11Install.sh > {{
      ansible_env.HOME }}/5gmeta/logs/osm_install"
  args:
    creates: "{{ ansible_env.HOME }}/5gmeta/logs/osm11_installed"
  when:
    - ansible_facts['distribution'] == "Ubuntu"
    - ansible_facts['distribution_major_version'] == "20"

- name: Add k8s cluster to osm
  shell: "{{ item }}"
  with_items:
    - osm vim-create --name 5gmeta-vim --user u --password p --tenant
        p --account_type dummy --auth_url http://localhost/dummy
    - "osm k8scluster-add 5gmeta-cluster --creds {{ ansible_env.HOME
       }}/.kube/config --vim 5gmeta-vim --k8s-nets '{k8s_net1: null
       }' --version 'v1.20.11' --description='K8s cluster'"
    - touch {{ ansible_env.HOME }}/5gmeta/logs/osm_cluster_added
  args:
    creates: "{{ ansible_env.HOME }}/5gmeta/logs/osm_cluster_added"

- name: Add 5gmeta osm repos
  shell: "{{ item }}"
  with_items:
    - osm repo-add --type osm 5gmeta-osm XXX
```

```
      - osm repo-add --type helm-chart 5gmeta XXX
      - touch {{ ansible_env.HOME }}/5gmeta/logs/osm_repos_added
    args:
      creates: "{{ ansible_env.HOME }}/5gmeta/logs/osm_repos_added"

#  - name: Log into DockerHub to reach 5gmeta's repositories
#    docker_login:
#      username: XXX
#      password: XXX

  - name: Log into DockerHub to reach 5gmeta's repositories
    shell: "{{ item }}"
    with_items:
      - sg docker -c 'docker login --username XXX --password XXX'
      - touch {{ ansible_env.HOME }}/5gmeta/logs/docker_login
    args:
      creates: "{{ ansible_env.HOME }}/5gmeta/logs/docker_login"

  - name: Create regcred secret for pulling images from 5gmeta's
     repositories
    shell: "{{ item }}"
    with_items:
      - kubectl create secret generic regcred --from-file=.
          dockerconfigjson={{ ansible_env.HOME }}/.docker/config.json
          --type=kubernetes.io/dockerconfigjson -n 5gmeta
      - touch {{ ansible_env.HOME }}/5gmeta/logs/docker_secret
    args:
      creates: "{{ ansible_env.HOME }}/5gmeta/logs/docker_secret"

  - name: Add 5gmeta helm repo
    kubernetes.core.helm_repository:
      name: 5gmeta
      repo_url: XXX

  - name: Deploy message-broker
    kubernetes.core.helm:
      name: message-broker
      release_namespace: 5gmeta
      chart_ref: 5gmeta/messagebroker-chart
      update_repo_cache: yes

  - name: Deploy video-broker
    kubernetes.core.helm:
      name: video-broker
      release_namespace: 5gmeta
      chart_ref: 5gmeta/videobroker-chart
      update_repo_cache: yes

  - name: Deploy registration-api
    kubernetes.core.helm:
      name: registration-api
      release_namespace: 5gmeta
      chart_ref: 5gmeta/registrationapi-chart
      update_repo_cache: yes

  - name: Deploy slaedge-api
    kubernetes.core.helm:
      name: slaedge-api
```

```
      release_namespace: 5gmeta
      chart_ref: 5gmeta/slaedgeapi-chart
      update_repo_cache: yes

  - name: Get IP geolocation data
    community.general.ipinfoio_facts:

  - name: Get token for accessing cloud APIs
    uri:
      url: https://5gmeta-platform.eu/identity/realms/5gmeta/protocol/
         openid-connect/token
      method: POST
      return_content: yes
      headers:
        Content-Type: application/x-www-form-urlencoded
      body_format: form-urlencoded
      body:
        grant_type: password
        username: XXX
        password: XXX
        client_id: 5gmeta_login
      creates: "{{ ansible_env.HOME }}/5gmeta/logs/services_registered"
    register: json_response
    changed_when: json_response.status | default(0) == 200

  - name: "Set token variable"
    set_fact:
      token: "{{ json_response.json.access_token }}"
    when: json_response is changed

  - name: Register MEC server in 5GMETA cloud
    uri:
      url: https://5gmeta-platform.eu/discovery-api/mec
      method: POST
      return_content: yes
      headers:
        Authorization: "{{ 'Bearer ' + token }}"
      body_format: json
      body: { "geolocation": [], "lat": "{{ latitude | default( loc |
         split(',') | first ) }}", "lng": "{{ loc | split(',') | last
         }}", "name": "{{ city | default( city ) }}", "organization":
         "{{ organization | default( 'Null' ) }}", "props": {}, "
         resources": { "cpu": "{{ ansible_processor_count | string
         }}", "gpu": "true", "memory": "{{ '{:0.2f}'.format(
         ansible_memory_mb.real.total | int / 1024) }}", "storage":
         "{{ '{:0.2f}'.format((ansible_mounts|selectattr('mount', '
         equalto', '/')|list)[0].size_total | int / 1073741824) }}" },
          "sb_services": [ { "description": "Message Broker", "ip":
         "{{ southbound_ip | default( ip ) }}", "port": "{{
         messagebroker_port | default( 5673 ) }}", "service_name": "
         message-broker" }, { "description": "Video Stream Broker", "
         ip": "{{ southbound_ip | default( ip ) }}", "port": "{{
         videobroker_port | default( 8443 ) }}", "service_name": "
         video-broker" }, { "description": "Registration API", "ip":
         "{{ southbound_ip | default( ip ) }}", "port": "{{
         registrationapi_port | default( 12346 ) }}", "service_name":
         "registration-api" } ] }
    when: token is defined
```

```
    register: json_response
    changed_when: json_response.status == 200

  - name: "Set mec_id variable"
    set_fact:
      mec_id: "{{ json_response.json.mec_id }}"
    when: json_response.changed

  - name: Add tiles to MEC
    uri:
      url: https://5gmeta-platform.eu/discovery-api/mec/{{ mec_id }}/
         tile/{{ item }}
      method: POST
      headers:
        Authorization: "{{ 'Bearer ' + token }}"
    when: mec_id is defined and token is defined
    loop: "{{ tiles }}"

  - name: Register message-broker service in 5GMETA cloud
    uri:
      url: https://5gmeta-platform.eu/discovery-api/mec/{{ mec_id }}/
         nbservices
      method: POST
      return_content: yes
      headers:
        Authorization: "{{ 'Bearer ' + token }}"
      body_format: json
      body: { "description": "Message Broker", "ip": "{{ northbound_ip
         | default( ip ) }}", "port": "61616", "props": "{}", "
         service_name": "message-broker" }
    when: mec_id is defined and token is defined

  - name: Register slaedge-api service in 5GMETA cloud
    uri:
      url: https://5gmeta-platform.eu/discovery-api/mec/{{ mec_id }}/
         nbservices
      method: POST
      return_content: yes
      headers:
        Authorization: "{{ 'Bearer ' + token }}"
      body_format: json
      body: { "description": "API to manage the SLAs and reservations
         in a 5GMETA MEC Server", "ip": "{{ northbound_ip | default(
         ip ) }}", "port": "5000", "props": "{}", "service_name": "
         slaedge-api" }
    when: mec_id is defined and token is defined

  - name: Log registered services
    become: yes
    shell: touch {{ ansible_env.HOME }}/5gmeta/logs/services_registered
    args:
      creates: "{{ ansible_env.HOME }}/5gmeta/logs/services_registered"

  - name: "Final message"
    ansible.builtin.debug:
      msg:
#      - Remember to add users to docker group with "usermod -aG docker
    <username>". Use "newgrp docker" to use the group immediately
```

```
      - "MEC stack correctly deployed, server registered in discovery
         module with ID {{ mec_id }}"

  handlers:
    - name: Docker start
      service:
        name: docker
        state: started
    - name: Docker restart
      service:
        name: docker
        state: restarted
```

Listing A.2: OSM installer for an already existing K8s cluster

```
#!/bin/bash

add_repo() {
  REPO_CHECK="^$1"
  grep "${REPO_CHECK/\[arch=amd64\]/\\[arch=amd64\\]}" /etc/apt/sources
     .list > /dev/null 2>&1
  if [ $? -ne 0 ]
  then
    need_packages_lw="software-properties-common apt-transport-https"
    echo -e "Checking required packages to add ETSI OSM debian repo:
       $need_packages_lw"
    dpkg -l $need_packages_lw &>/dev/null \
      || ! echo -e "One or several required packages are not installed.
          Updating apt cache requires root privileges." \
      || sudo apt-get -qy update \
      || ! echo "failed to run apt-get update" \
      || exit 1
    dpkg -l $need_packages_lw &>/dev/null \
      || ! echo -e "Installing $need_packages_lw requires root
         privileges." \
      || sudo apt-get install -y $need_packages_lw \
      || ! echo "failed to install $need_packages_lw" \
      || exit 1
    wget -qO - "$REPOSITORY_BASE/$RELEASE/OSM%20ETSI%20Release%20Key.
       gpg" | sudo APT_KEY_DONT_WARN_ON_DANGEROUS_USAGE apt-key add -
    sudo DEBIAN_FRONTEND=noninteractive add-apt-repository -y "$1"
    sudo APT_KEY_DONT_WARN_ON_DANGEROUS_USAGE=1 DEBIAN_FRONTEND=
       noninteractive apt-get -y update
    return 0
  fi

  return 1
}

clean_old_repo() {
dpkg -s 'osm-devops' &> /dev/null
if [ $? -eq 0 ]; then
  # Clean the previous repos that might exist
  sudo sed -i "/osm-download.etsi.org/d" /etc/apt/sources.list
fi
}

function install_lxd() {
     # Apply sysctl production values for optimal performance
    sudo cp ${OSM_DEVOPS}/installers/60-lxd-production.conf /etc/sysctl
       .d/60-lxd-production.conf
    sudo sysctl --system

    # Install LXD snap
    sudo apt-get remove --purge -y liblxc1 lxc-common lxcfs lxd lxd-
       client
    sudo snap install lxd --channel $LXD_VERSION/stable

    # Configure LXD
    sudo usermod -a -G lxd `whoami`
    cat ${OSM_DEVOPS}/installers/lxd-preseed.conf | sed 's/^config: {}/
```

```
        config:\n  core.https_address: '$DEFAULT_IP':8443/' | sg lxd -c
         "lxd init --preseed"
    sg lxd -c "lxd waitready"
    DEFAULT_IF=$(ip route list|awk '$1=="default" {print $5; exit}')
    [ -z "$DEFAULT_IF" ] && DEFAULT_IF=$(route -n |awk '$1~/^0.0.0.0/ {
        print $8; exit}')
    [ -z "$DEFAULT_IF" ] && FATAL "Not possible to determine the
        interface with the default route 0.0.0.0"
    DEFAULT_MTU=$(ip addr show ${DEFAULT_IF} | perl -ne 'if (/mtu\s(\d
        +)/) {print $1;}')
    sg lxd -c "lxc profile device set default eth0 mtu $DEFAULT_MTU"
    sg lxd -c "lxc network set lxdbr0 bridge.mtu $DEFAULT_MTU"
    #sudo systemctl stop lxd-bridge
    #sudo systemctl --system daemon-reload
    #sudo systemctl enable lxd-bridge
    #sudo systemctl start lxd-bridge
}

function install_juju() {
    echo "Installing juju"
    sudo snap install juju --classic --channel=$JUJU_VERSION/stable
    [[ ":$PATH": != *":/snap/bin:"* ]] && PATH="/snap/bin:${PATH}"
    sleep 20
    update_juju_images
    echo "Finished installation of juju"
    return 0
}

function update_juju_images(){
    crontab -l | grep update-juju-lxc-images || (crontab -l 2>/dev/null
        ; echo "0 4 * * 6 $USER ${OSM_DEVOPS}/installers/update-juju-
        lxc-images --xenial --bionic") | crontab -
    ${OSM_DEVOPS}/installers/update-juju-lxc-images --xenial --bionic
}

function parse_juju_password {
    password_file="${HOME}/.local/share/juju/accounts.yaml"
    local controller_name=$1
    local s='[[:space:]]*' w='[a-zA-Z0-9_-]*' fs=$(echo @|tr @ '\034')
    sed -ne "s|^\($s\):|\1|" \
         -e "s|^\($s\)\($w\)$s:$s[\"']\(.*\)[\"']$s\$|\1$fs\2$fs\3|p" \
         -e "s|^\($s\)\($w\)$s:$s\(.*\)$s\$|\1$fs\2$fs\3|p"
       $password_file |
    awk -F$fs -v controller=$controller_name '{
        indent = length($1)/2;
        vname[indent] = $2;
        for (i in vname) {if (i > indent) {delete vname[i]}}
        if (length($3) > 0) {
            vn=""; for (i=0; i<indent; i++) {vn=(vn)(vname[i])("_")}
            if (match(vn,controller) && match($2,"password")) {
                printf("%s",$3);
            }
        }
    }'
}

function juju_createcontroller_k8s(){
    cat $HOME/.kube/config | juju add-k8s $OSM_VCA_K8S_CLOUDNAME --
```

```
    client \
    || FATAL "Failed to add K8s endpoint and credential for client in
    cloud $OSM_VCA_K8S_CLOUDNAME"
    juju bootstrap -v --debug $OSM_VCA_K8S_CLOUDNAME $OSM_STACK_NAME  \
            --config controller-service-type=loadbalancer \
            --agent-version=$JUJU_AGENT_VERSION \
    || FATAL "Failed to bootstrap controller $OSM_STACK_NAME in cloud
    $OSM_VCA_K8S_CLOUDNAME"
}

function juju_addlxd_cloud(){
    mkdir -p /tmp/.osm
    OSM_VCA_CLOUDNAME="lxd-cloud"
    LXDENDPOINT=$DEFAULT_IP
    LXD_CLOUD=/tmp/.osm/lxd-cloud.yaml
    LXD_CREDENTIALS=/tmp/.osm/lxd-credentials.yaml

    cat << EOF > $LXD_CLOUD
clouds:
  $OSM_VCA_CLOUDNAME:
    type: lxd
    auth-types: [certificate]
    endpoint: "https://$LXDENDPOINT:8443"
    config:
      ssl-hostname-verification: false
EOF
    openssl req -nodes -new -x509 -keyout /tmp/.osm/client.key -out /
    tmp/.osm/client.crt -days 365 -subj "/C=FR/ST=Nice/L=Nice/O=ETSI/OU
    =OSM/CN=osm.etsi.org"
    cat << EOF > $LXD_CREDENTIALS
credentials:
  $OSM_VCA_CLOUDNAME:
    lxd-cloud:
      auth-type: certificate
      server-cert: /var/snap/lxd/common/lxd/server.crt
      client-cert: /tmp/.osm/client.crt
      client-key: /tmp/.osm/client.key
EOF
    lxc config trust add local: /tmp/.osm/client.crt
    juju add-cloud -c $OSM_STACK_NAME $OSM_VCA_CLOUDNAME $LXD_CLOUD --
    force
    juju add-credential -c $OSM_STACK_NAME $OSM_VCA_CLOUDNAME -f
    $LXD_CREDENTIALS
    sg lxd -c "lxd waitready"
    juju controller-config features=[k8s-operators]
}

function juju_createcontroller() {
    if ! juju show-controller $OSM_STACK_NAME &> /dev/null; then
        # Not found created, create the controller
        sudo usermod -a -G lxd ${USER}
        sg lxd -c "juju bootstrap --bootstrap-series=xenial --agent-
    version=$JUJU_AGENT_VERSION $OSM_VCA_CLOUDNAME $OSM_STACK_NAME"
    fi
    [ $(juju controllers | awk "/^${OSM_STACK_NAME}[\*| ]/{print $1}"|
    wc -l) -eq 1 ] || FATAL "Juju installation failed"
    juju controller-config features=[k8s-operators]
}
```

```
function juju_createproxy() {
    check_install_iptables_persistent

    if ! sudo iptables -t nat -C PREROUTING -p tcp -m tcp -d
   $DEFAULT_IP --dport 17070 -j DNAT --to-destination $OSM_VCA_HOST;
   then
        sudo iptables -t nat -A PREROUTING -p tcp -m tcp -d $DEFAULT_IP
    --dport 17070 -j DNAT --to-destination $OSM_VCA_HOST
        sudo netfilter-persistent save
    fi
}

function check_install_iptables_persistent(){
    echo -e "\nChecking required packages: iptables-persistent"
    if ! dpkg -l iptables-persistent &>/dev/null; then
        echo -e "    Not installed.\nInstalling iptables-persistent
   requires root privileges"
        echo iptables-persistent iptables-persistent/autosave_v4
   boolean true | sudo debconf-set-selections
        echo iptables-persistent iptables-persistent/autosave_v6
   boolean true | sudo debconf-set-selections
        sudo apt-get -yq install iptables-persistent
    fi
}

function set_vca_variables() {
    OSM_VCA_CLOUDNAME="lxd-cloud"

    OSM_VCA_HOST=`sg lxd -c "juju show-controller $OSM_STACK_NAME"|grep
    api-endpoints|awk -F\' '{print $2}'|awk -F\: '{print $1}'`
    [ -z "$OSM_VCA_HOST" ] && FATAL "Cannot obtain juju controller IP
   address"

    OSM_VCA_SECRET=$(parse_juju_password $OSM_STACK_NAME)
    [ -z "$OSM_VCA_SECRET" ] && FATAL "Cannot obtain juju secret"

    OSM_VCA_PUBKEY=$(cat $HOME/.local/share/juju/ssh/juju_id_rsa.pub)
    [ -z "$OSM_VCA_PUBKEY" ] && FATAL "Cannot obtain juju public key"

    OSM_VCA_CACERT=$(juju controllers --format json | jq -r --arg
   controller $OSM_STACK_NAME '.controllers[$controller]["ca-cert"]' |
    base64 | tr -d \\n)
    [ -z "$OSM_VCA_CACERT" ] && FATAL "Cannot obtain juju CA
   certificate"
}

function check_for_readiness() {
    # Default input values
    sampling_period=2       # seconds
    time_for_readiness=20   # seconds ready
    time_for_failure=200    # seconds broken
    OPENEBS_NAMESPACE=openebs
    METALLB_NAMESPACE=metallb-system
    # STACK_NAME=osm          # By default, "osm"

    # Equivalent number of samples
    oks_threshold=$((time_for_readiness/${sampling_period}))     # No.
```

```
ok samples to declare the system ready
 failures_threshold=$((time_for_failure/${sampling_period}))  # No.
nok samples to declare the system broken
 failures_in_a_row=0
 oks_in_a_row=0

 ############################################################
 # Loop to check system readiness
 ############################################################
 while [[ (${failures_in_a_row} -lt ${failures_threshold}) && (${
oks_in_a_row} -lt ${oks_threshold}) ]]
 do
     # State of OpenEBS
     OPENEBS_STATE=$(kubectl get pod -n ${OPENEBS_NAMESPACE} --no-
headers 2>&1)
     OPENEBS_READY=$(echo "${OPENEBS_STATE}" | awk '$2=="1/1" || $2
=="2/2" {printf ("%s\t%s\t\n", $1, $2)}')
     OPENEBS_NOT_READY=$(echo "${OPENEBS_STATE}" | awk '$2!="1/1" &&
 $2!="2/2" {printf ("%s\t%s\t\n", $1, $2)}')
     COUNT_OPENEBS_READY=$(echo "${OPENEBS_READY}"| grep -v -e '^$'
| wc -l)
     COUNT_OPENEBS_NOT_READY=$(echo "${OPENEBS_NOT_READY}" | grep -v
 -e '^$' | wc -l)

     # State of MetalLB
     METALLB_STATE=$(kubectl get pod -n ${METALLB_NAMESPACE} --no-
headers 2>&1)
     METALLB_READY=$(echo "${METALLB_STATE}" | awk '$2=="1/1" || $2
=="2/2" {printf ("%s\t%s\t\n", $1, $2)}')
     METALLB_NOT_READY=$(echo "${METALLB_STATE}" | awk '$2!="1/1" &&
 $2!="2/2" {printf ("%s\t%s\t\n", $1, $2)}')
     COUNT_METALLB_READY=$(echo "${METALLB_READY}" | grep -v -e '^$'
 | wc -l)
     COUNT_METALLB_NOT_READY=$(echo "${METALLB_NOT_READY}" | grep -v
 -e '^$' | wc -l)

     # OK sample
     if [[ $((${COUNT_OPENEBS_NOT_READY}+${COUNT_METALLB_NOT_READY})
) -eq 0 ]]
     then
         ((++oks_in_a_row))
         failures_in_a_row=0
         echo -ne ===\> Successful checks: "${oks_in_a_row}"/${
oks_threshold}\\r
     # NOK sample
     else
         ((++failures_in_a_row))
         oks_in_a_row=0
         echo
         echo Bootstraping... "${failures_in_a_row}" checks of ${
failures_threshold}

         # Reports failed pods in OpenEBS
         if [[ "${COUNT_OPENEBS_NOT_READY}" -ne 0 ]]
         then
             echo "OpenEBS: Waiting for ${COUNT_OPENEBS_NOT_READY}
of $((${COUNT_OPENEBS_NOT_READY}+${COUNT_OPENEBS_READY})) pods to
be ready:"
```

```
                echo "${OPENEBS_NOT_READY}"
                echo
            fi

            # Reports failed statefulsets
            if [[ "${COUNT_METALLB_NOT_READY}" -ne 0 ]]
            then
                echo "MetalLB: Waiting for ${COUNT_METALLB_NOT_READY}
   of $((${COUNT_METALLB_NOT_READY}+${COUNT_METALLB_READY})) pods to
   be ready:"
                echo "${METALLB_NOT_READY}"
                echo
            fi
        fi

        #------------ NEXT SAMPLE
        sleep ${sampling_period}
    done

    ############################################################
    # OUTCOME
    ############################################################
    if [[ (${failures_in_a_row} -ge ${failures_threshold}) ]]
    then
        echo
        FATAL "K8S CLUSTER IS BROKEN"
    else
        echo
        echo "K8S CLUSTER IS READY"
    fi
}

function generate_docker_images() {
    echo "Pulling docker images"

    sg docker -c "docker pull wurstmeister/zookeeper" || FATAL "cannot
   get zookeeper docker image"
    sg docker -c "docker pull wurstmeister/kafka:${KAFKA_TAG}" || FATAL
     "cannot get kafka docker image"
    sg docker -c "docker pull mongo" || FATAL "cannot get mongo docker
   image"
    sg docker -c "docker pull prom/prometheus:${PROMETHEUS_TAG}" ||
   FATAL "cannot get prometheus docker image"
    sg docker -c "docker pull google/cadvisor:${PROMETHEUS_CADVISOR_TAG
   }" || FATAL "cannot get prometheus cadvisor docker image"
    sg docker -c "docker pull grafana/grafana:${GRAFANA_TAG}" || FATAL
   "cannot get grafana docker image"
    sg docker -c "docker pull mariadb:${KEYSTONEDB_TAG}" || FATAL "
   cannot get keystone-db docker image"
    sg docker -c "docker pull mysql:5" || FATAL "cannot get mysql
   docker image"

    echo "Pulling OSM docker images"
    for module in MON POL NBI KEYSTONE RO LCM NG-UI osmclient; do
        module_lower=${module,,}
        module_tag="${OSM_DOCKER_TAG}"

        echo "Pulling ${DOCKER_USER}/${module_lower}:${module_tag}
```

```
    docker image"
        sg docker -c "docker pull ${DOCKER_USER}/${module_lower}:${
    module_tag}" || FATAL "cannot pull $module docker image"
    done
    echo "Finished pulling and generating docker images"
}

function generate_k8s_manifest_files() {
    #kubernetes resources
    sudo cp -bR ${OSM_DEVOPS}/installers/docker/osm_pods
    $OSM_DOCKER_WORK_DIR
    sudo rm -f $OSM_K8S_WORK_DIR/mongo.yaml
#    sudo rm -f $OSM_K8S_WORK_DIR/light-ui.yaml
}

function generate_docker_env_files() {
    echo "Doing a backup of existing env files"
    sudo cp $OSM_DOCKER_WORK_DIR/keystone-db.env{,~}
    sudo cp $OSM_DOCKER_WORK_DIR/keystone.env{,~}
    sudo cp $OSM_DOCKER_WORK_DIR/lcm.env{,~}
    sudo cp $OSM_DOCKER_WORK_DIR/mon.env{,~}
    sudo cp $OSM_DOCKER_WORK_DIR/nbi.env{,~}
    sudo cp $OSM_DOCKER_WORK_DIR/pol.env{,~}
    sudo cp $OSM_DOCKER_WORK_DIR/ro-db.env{,~}
    sudo cp $OSM_DOCKER_WORK_DIR/ro.env{,~}

    echo "Generating docker env files"
    # LCM
    if [ ! -f $OSM_DOCKER_WORK_DIR/lcm.env ]; then
        echo "OSMLCM_DATABASE_COMMONKEY=${OSM_DATABASE_COMMONKEY}" |
    sudo tee -a $OSM_DOCKER_WORK_DIR/lcm.env
    fi

    if ! grep -Fq "OSMLCM_VCA_HOST" $OSM_DOCKER_WORK_DIR/lcm.env; then
        echo "OSMLCM_VCA_HOST=${OSM_VCA_HOST}" | sudo tee -a
    $OSM_DOCKER_WORK_DIR/lcm.env
    else
        sudo sed -i "s|OSMLCM_VCA_HOST.*|OSMLCM_VCA_HOST=$OSM_VCA_HOST|
    g" $OSM_DOCKER_WORK_DIR/lcm.env
    fi

    if ! grep -Fq "OSMLCM_VCA_SECRET" $OSM_DOCKER_WORK_DIR/lcm.env;
    then
        echo "OSMLCM_VCA_SECRET=${OSM_VCA_SECRET}" | sudo tee -a
    $OSM_DOCKER_WORK_DIR/lcm.env
    else
        sudo sed -i "s|OSMLCM_VCA_SECRET.*|OSMLCM_VCA_SECRET=
    $OSM_VCA_SECRET|g" $OSM_DOCKER_WORK_DIR/lcm.env
    fi

    if ! grep -Fq "OSMLCM_VCA_PUBKEY" $OSM_DOCKER_WORK_DIR/lcm.env;
    then
        echo "OSMLCM_VCA_PUBKEY=${OSM_VCA_PUBKEY}" | sudo tee -a
    $OSM_DOCKER_WORK_DIR/lcm.env
    else
        sudo sed -i "s|OSMLCM_VCA_PUBKEY.*|OSMLCM_VCA_PUBKEY=${
    OSM_VCA_PUBKEY}|g" $OSM_DOCKER_WORK_DIR/lcm.env
    fi
```

```
if ! grep -Fq "OSMLCM_VCA_CACERT" $OSM_DOCKER_WORK_DIR/lcm.env;
then
    echo "OSMLCM_VCA_CACERT=${OSM_VCA_CACERT}" | sudo tee -a
$OSM_DOCKER_WORK_DIR/lcm.env
else
    sudo sed -i "s|OSMLCM_VCA_CACERT.*|OSMLCM_VCA_CACERT=${
OSM_VCA_CACERT}|g" $OSM_DOCKER_WORK_DIR/lcm.env
fi

if [ -n "$OSM_VCA_APIPROXY" ]; then
    if ! grep -Fq "OSMLCM_VCA_APIPROXY" $OSM_DOCKER_WORK_DIR/lcm.
env; then
        echo "OSMLCM_VCA_APIPROXY=${OSM_VCA_APIPROXY}" | sudo tee -
a $OSM_DOCKER_WORK_DIR/lcm.env
    else
        sudo sed -i "s|OSMLCM_VCA_APIPROXY.*|OSMLCM_VCA_APIPROXY=${
OSM_VCA_APIPROXY}|g" $OSM_DOCKER_WORK_DIR/lcm.env
    fi
fi

if ! grep -Fq "OSMLCM_VCA_ENABLEOSUPGRADE" $OSM_DOCKER_WORK_DIR/lcm
.env; then
    echo "# OSMLCM_VCA_ENABLEOSUPGRADE=false" | sudo tee -a
$OSM_DOCKER_WORK_DIR/lcm.env
fi

if ! grep -Fq "OSMLCM_VCA_APTMIRROR" $OSM_DOCKER_WORK_DIR/lcm.env;
   then
    echo "# OSMLCM_VCA_APTMIRROR=http://archive.ubuntu.com/ubuntu/"
        | sudo tee -a $OSM_DOCKER_WORK_DIR/lcm.env
fi

if ! grep -Fq "OSMLCM_VCA_CLOUD" $OSM_DOCKER_WORK_DIR/lcm.env; then
    echo "OSMLCM_VCA_CLOUD=${OSM_VCA_CLOUDNAME}" | sudo tee -a
       $OSM_DOCKER_WORK_DIR/lcm.env
else
    sudo sed -i "s|OSMLCM_VCA_CLOUD.*|OSMLCM_VCA_CLOUD=${
       OSM_VCA_CLOUDNAME}|g" $OSM_DOCKER_WORK_DIR/lcm.env
fi

if ! grep -Fq "OSMLCM_VCA_K8S_CLOUD" $OSM_DOCKER_WORK_DIR/lcm.env;
   then
    echo "OSMLCM_VCA_K8S_CLOUD=${OSM_VCA_K8S_CLOUDNAME}" | sudo tee
        -a $OSM_DOCKER_WORK_DIR/lcm.env
else
    sudo sed -i "s|OSMLCM_VCA_K8S_CLOUD.*|OSMLCM_VCA_K8S_CLOUD=${
       OSM_VCA_K8S_CLOUDNAME}|g" $OSM_DOCKER_WORK_DIR/lcm.env
fi

# RO
MYSQL_ROOT_PASSWORD=$(generate_secret)
if [ ! -f $OSM_DOCKER_WORK_DIR/ro-db.env ]; then
    echo "MYSQL_ROOT_PASSWORD=${MYSQL_ROOT_PASSWORD}" | sudo tee
       $OSM_DOCKER_WORK_DIR/ro-db.env
fi
if [ ! -f $OSM_DOCKER_WORK_DIR/ro.env ]; then
    echo "RO_DB_ROOT_PASSWORD=${MYSQL_ROOT_PASSWORD}" | sudo tee
```

```
        $OSM_DOCKER_WORK_DIR/ro.env
fi
if ! grep -Fq "OSMRO_DATABASE_COMMONKEY" $OSM_DOCKER_WORK_DIR/ro.
   env; then
    echo "OSMRO_DATABASE_COMMONKEY=${OSM_DATABASE_COMMONKEY}" |
        sudo tee -a $OSM_DOCKER_WORK_DIR/ro.env
fi

# Keystone
KEYSTONE_DB_PASSWORD=$(generate_secret)
SERVICE_PASSWORD=$(generate_secret)
if [ ! -f $OSM_DOCKER_WORK_DIR/keystone-db.env ]; then
    echo "MYSQL_ROOT_PASSWORD=${MYSQL_ROOT_PASSWORD}" | sudo tee
        $OSM_DOCKER_WORK_DIR/keystone-db.env
fi
if [ ! -f $OSM_DOCKER_WORK_DIR/keystone.env ]; then
    echo "ROOT_DB_PASSWORD=${MYSQL_ROOT_PASSWORD}" | sudo tee
        $OSM_DOCKER_WORK_DIR/keystone.env
    echo "KEYSTONE_DB_PASSWORD=${KEYSTONE_DB_PASSWORD}" | sudo tee
        -a $OSM_DOCKER_WORK_DIR/keystone.env
    echo "SERVICE_PASSWORD=${SERVICE_PASSWORD}" | sudo tee -a
        $OSM_DOCKER_WORK_DIR/keystone.env
fi

# NBI
if [ ! -f $OSM_DOCKER_WORK_DIR/nbi.env ]; then
    echo "OSMNBI_AUTHENTICATION_SERVICE_PASSWORD=${SERVICE_PASSWORD
        }" | sudo tee $OSM_DOCKER_WORK_DIR/nbi.env
    echo "OSMNBI_DATABASE_COMMONKEY=${OSM_DATABASE_COMMONKEY}" |
        sudo tee -a $OSM_DOCKER_WORK_DIR/nbi.env
fi

# MON
if [ ! -f $OSM_DOCKER_WORK_DIR/mon.env ]; then
    echo "OSMMON_KEYSTONE_SERVICE_PASSWORD=${SERVICE_PASSWORD}" |
        sudo tee -a $OSM_DOCKER_WORK_DIR/mon.env
    echo "OSMMON_DATABASE_COMMONKEY=${OSM_DATABASE_COMMONKEY}" |
        sudo tee -a $OSM_DOCKER_WORK_DIR/mon.env
    echo "OSMMON_SQL_DATABASE_URI=mysql://root:${
        MYSQL_ROOT_PASSWORD}@mysql:3306/mon" | sudo tee -a
        $OSM_DOCKER_WORK_DIR/mon.env
fi

if ! grep -Fq "OS_NOTIFIER_URI" $OSM_DOCKER_WORK_DIR/mon.env; then
    echo "OS_NOTIFIER_URI=http://${DEFAULT_IP}:8662" | sudo tee -a
        $OSM_DOCKER_WORK_DIR/mon.env
else
    sudo sed -i "s|OS_NOTIFIER_URI.*|OS_NOTIFIER_URI=http://
        $DEFAULT_IP:8662|g" $OSM_DOCKER_WORK_DIR/mon.env
fi

if ! grep -Fq "OSMMON_VCA_HOST" $OSM_DOCKER_WORK_DIR/mon.env; then
    echo "OSMMON_VCA_HOST=${OSM_VCA_HOST}" | sudo tee -a
        $OSM_DOCKER_WORK_DIR/mon.env
else
    sudo sed -i "s|OSMMON_VCA_HOST.*|OSMMON_VCA_HOST=$OSM_VCA_HOST|
        g" $OSM_DOCKER_WORK_DIR/mon.env
fi
```

```
    if ! grep -Fq "OSMMON_VCA_SECRET" $OSM_DOCKER_WORK_DIR/mon.env;
       then
        echo "OSMMON_VCA_SECRET=${OSM_VCA_SECRET}" | sudo tee -a
           $OSM_DOCKER_WORK_DIR/mon.env
    else
        sudo sed -i "s|OSMMON_VCA_SECRET.*|OSMMON_VCA_SECRET=
           $OSM_VCA_SECRET|g" $OSM_DOCKER_WORK_DIR/mon.env
    fi

    if ! grep -Fq "OSMMON_VCA_CACERT" $OSM_DOCKER_WORK_DIR/mon.env;
       then
        echo "OSMMON_VCA_CACERT=${OSM_VCA_CACERT}" | sudo tee -a
           $OSM_DOCKER_WORK_DIR/mon.env
    else
        sudo sed -i "s|OSMMON_VCA_CACERT.*|OSMMON_VCA_CACERT=${
           OSM_VCA_CACERT}|g" $OSM_DOCKER_WORK_DIR/mon.env
    fi


    # POL
    if [ ! -f $OSM_DOCKER_WORK_DIR/pol.env ]; then
        echo "OSMPOL_SQL_DATABASE_URI=mysql://root:${
           MYSQL_ROOT_PASSWORD}@mysql:3306/pol" | sudo tee -a
           $OSM_DOCKER_WORK_DIR/pol.env
    fi

    echo "Finished generation of docker env files"
}

#deploy charmed services
function deploy_charmed_services() {
    juju add-model $OSM_STACK_NAME $OSM_VCA_K8S_CLOUDNAME
    # The channel prefix is not always recognized (maybe it should be
       cs: not ch:
    # anyway by default it will get it from the Charm Store)
    # juju deploy ch:mongodb-k8s -m $OSM_STACK_NAME
    juju deploy mongodb-k8s -m $OSM_STACK_NAME
}

#creates secrets from env files which will be used by containers
function kube_secrets(){
    kubectl create ns $OSM_STACK_NAME
    kubectl create secret generic lcm-secret -n $OSM_STACK_NAME --from-
       env-file=$OSM_DOCKER_WORK_DIR/lcm.env
    kubectl create secret generic mon-secret -n $OSM_STACK_NAME --from-
       env-file=$OSM_DOCKER_WORK_DIR/mon.env
    kubectl create secret generic nbi-secret -n $OSM_STACK_NAME --from-
       env-file=$OSM_DOCKER_WORK_DIR/nbi.env
    kubectl create secret generic ro-db-secret -n $OSM_STACK_NAME --
       from-env-file=$OSM_DOCKER_WORK_DIR/ro-db.env
    kubectl create secret generic ro-secret -n $OSM_STACK_NAME --from-
       env-file=$OSM_DOCKER_WORK_DIR/ro.env
    kubectl create secret generic keystone-secret -n $OSM_STACK_NAME --
       from-env-file=$OSM_DOCKER_WORK_DIR/keystone.env
    kubectl create secret generic pol-secret -n $OSM_STACK_NAME --from-
       env-file=$OSM_DOCKER_WORK_DIR/pol.env
}
```

```
function update_manifest_files() {
    osm_services="nbi lcm ro pol mon ng-ui keystone prometheus"
    list_of_services=""
    for module in $osm_services; do
        module_upper="${module^^}"
        if ! echo $TO_REBUILD | grep -q $module_upper ; then
            list_of_services="$list_of_services $module"
        fi
    done
}

function namespace_vol() {
    osm_services="nbi lcm ro pol mon kafka mysql prometheus"
    for osm in $osm_services; do
        sudo  sed -i "s#path: /var/lib/osm#path: $OSM_NAMESPACE_VOL#g"
           $OSM_K8S_WORK_DIR/$osm.yaml
    done
}

#deploys osm pods and services
function deploy_osm_services() {
    sudo sed -i 's/nodePort: 3000/nodePort: 3001/' $OSM_K8S_WORK_DIR/
       grafana.yaml
    kubectl apply -n $OSM_STACK_NAME -f $OSM_K8S_WORK_DIR
}

function install_k8s_monitoring() {
    # install OSM monitoring
    sudo chmod +x $OSM_DEVOPS/installers/k8s/*.sh
    sudo $OSM_DEVOPS/installers/k8s/install_osm_k8s_monitoring.sh ||
       FATAL_TRACK install_k8s_monitoring "k8s/
       install_osm_k8s_monitoring.sh failed"
}

function install_osmclient(){
    CLIENT_RELEASE=${RELEASE#"-R "}
    CLIENT_REPOSITORY_KEY="OSM%20ETSI%20Release%20Key.gpg"
    CLIENT_REPOSITORY=${REPOSITORY#"-r "}
    CLIENT_REPOSITORY_BASE=${REPOSITORY_BASE#"-u "}
    key_location=$CLIENT_REPOSITORY_BASE/$CLIENT_RELEASE/
       $CLIENT_REPOSITORY_KEY
    curl $key_location | sudo APT_KEY_DONT_WARN_ON_DANGEROUS_USAGE=1
       apt-key add -
    sudo add-apt-repository -y "deb [arch=amd64]
       $CLIENT_REPOSITORY_BASE/$CLIENT_RELEASE $CLIENT_REPOSITORY
       osmclient IM"
    sudo apt-get update
    sudo apt-get install -y python3-pip
    sudo -H LC_ALL=C python3 -m pip install -U pip
    sudo -H LC_ALL=C python3 -m pip install -U python-magic pyangbind
       verboselogs
    sudo apt-get install -y python3-osm-im python3-osmclient
    if [ -f /usr/lib/python3/dist-packages/osm_im/requirements.txt ];
       then
        python3 -m pip install -r /usr/lib/python3/dist-packages/osm_im
           /requirements.txt
    fi
```

```
    if [ -f /usr/lib/python3/dist-packages/osmclient/requirements.txt
        ]; then
         sudo apt-get install -y libcurl4-openssl-dev libssl-dev
         python3 -m pip install -r /usr/lib/python3/dist-packages/
            osmclient/requirements.txt
    fi
    #sed 's,OSM_SOL005=[^$]*,OSM_SOL005=True,' -i ${HOME}/.bashrc
    #echo 'export OSM_HOSTNAME=localhost' >> ${HOME}/.bashrc
    #echo 'export OSM_SOL005=True' >> ${HOME}/.bashrc
    echo -e "\nOSM client installed"
    echo -e "OSM client assumes that OSM host is running in localhost
        (127.0.0.1)."
    echo -e "In case you want to interact with a different OSM host,
        you will have to configure this env variable in your .bashrc
        file:"
    echo "     export OSM_HOSTNAME=<OSM_host>"
    return 0
}

function add_local_k8scluster() {
    /usr/bin/osm --all-projects vim-create \
      --name _system-osm-vim \
      --account_type dummy \
      --auth_url http://dummy \
      --user osm --password osm --tenant osm \
      --description "dummy" \
      --config '{management_network_name: mgmt}'
    /usr/bin/osm --all-projects k8scluster-add \
      --creds ${HOME}/.kube/config \
      --vim _system-osm-vim \
      --k8s-nets '{"net1": null}' \
      --version '1.15' \
      --description "OSM Internal Cluster" \
      _system-osm-k8s
}

function generate_secret() {
    head /dev/urandom | tr -dc A-Za-z0-9 | head -c 32
}

function install_osm() {
    [ "$USER" == "root" ] && FATAL "You are running the installer as
        root. The installer is prepared to be executed as a normal user
         with sudo privileges."

    echo "Installing OSM"

    echo "Determining IP address of the interface with the default
        route"
    DEFAULT_IF=$(ip route list|awk '$1=="default" {print $5; exit}')
    [ -z "$DEFAULT_IF" ] && DEFAULT_IF=$(route -n |awk '$1~/^0.0.0.0/ {
        print $8; exit}')
    [ -z "$DEFAULT_IF" ] && FATAL "Not possible to determine the
        interface with the default route 0.0.0.0"
    DEFAULT_IP=`ip -o -4 a s ${DEFAULT_IF} |awk '{split($4,a,"/");
        print a[1]}'`
    [ -z "$DEFAULT_IP" ] && FATAL "Not possible to determine the IP
        address of the interface with the default route"
```

```
need_packages_lw="snapd"
echo -e "Checking required packages: $need_packages_lw"
dpkg -l $need_packages_lw &>/dev/null \
    || ! echo -e "One or several required packages are not
       installed. Updating apt cache requires root privileges." \
    || sudo apt-get update \
    || FATAL "failed to run apt-get update"
dpkg -l $need_packages_lw &>/dev/null \
    || ! echo -e "Installing $need_packages_lw requires root
       privileges." \
    || sudo apt-get install -y $need_packages_lw \
    || FATAL "failed to install $need_packages_lw"
install_lxd

echo "Creating folders for installation"
[ ! -d "$OSM_DOCKER_WORK_DIR" ] && sudo mkdir -p
   $OSM_DOCKER_WORK_DIR

check_for_readiness
install_juju
juju_createcontroller_k8s
juju_addlxd_cloud
juju_createcontroller
juju_createproxy
set_vca_variables

OSM_DATABASE_COMMONKEY=$(generate_secret)
[ -z "OSM_DATABASE_COMMONKEY" ] && FATAL "Cannot generate common db
    secret"

generate_docker_images
generate_k8s_manifest_files
generate_docker_env_files

deploy_charmed_services
kube_secrets
update_manifest_files
namespace_vol
deploy_osm_services

#install_k8s_monitoring

install_osmclient

echo -e "Checking OSM health state..."
$OSM_DEVOPS/installers/osm_health.sh -s ${OSM_STACK_NAME} -k -f 8
   || \
echo -e "OSM is not healthy, but will probably converge to a
   healthy state soon." && \
echo -e "Check OSM status with: kubectl -n ${OSM_STACK_NAME} get
   all" && \

add_local_k8scluster

wget -q -O- https://osm-download.etsi.org/ftp/osm-11.0-eleven/
   README2.txt &> /dev/null
return 0
```

```
}

LXD_VERSION=4.0
JUJU_VERSION=2.9
JUJU_AGENT_VERSION=2.9.22
RELEASE="ReleaseELEVEN"
REPOSITORY="stable"
OSM_DEVOPS="/usr/share/osm-devops"
OSM_VCA_CLOUDNAME="localhost"
OSM_VCA_K8S_CLOUDNAME="k8scloud"
OSM_STACK_NAME=osm
REPOSITORY_KEY="OSM%20ETSI%20Release%20Key.gpg"
REPOSITORY_BASE="https://osm-download.etsi.org/repository/osm/debian"
OSM_WORK_DIR="/etc/osm"
OSM_DOCKER_WORK_DIR="${OSM_WORK_DIR}/docker"
OSM_K8S_WORK_DIR="${OSM_DOCKER_WORK_DIR}/osm_pods"
OSM_HOST_VOL="/var/lib/osm"
OSM_NAMESPACE_VOL="${OSM_HOST_VOL}/${OSM_STACK_NAME}"
OSM_DOCKER_TAG=11
DOCKER_USER=opensourcemano
KAFKA_TAG=2.11-1.0.2
PROMETHEUS_TAG=v2.4.3
GRAFANA_TAG=latest
PROMETHEUS_NODE_EXPORTER_TAG=0.18.1
PROMETHEUS_CADVISOR_TAG=latest
KEYSTONEDB_TAG=10
#ELASTIC_VERSION=6.4.2
#ELASTIC_CURATOR_VERSION=5.5.4

#main
source $OSM_DEVOPS/common/all_funcs
clean_old_repo
add_repo "deb [arch=amd64] $REPOSITORY_BASE/$RELEASE $REPOSITORY devops
    "
sudo DEBIAN_FRONTEND=noninteractive apt-get -q update
sudo DEBIAN_FRONTEND=noninteractive apt-get install osm-devops

need_packages="git wget curl tar"

echo -e "Checking required packages: $need_packages"
dpkg -l $need_packages &>/dev/null \
  || ! echo -e "One or several required packages are not installed.
     Updating apt cache requires root privileges." \
  || sudo apt-get update \
  || FATAL "failed to run apt-get update"
dpkg -l $need_packages &>/dev/null \
  || ! echo -e "Installing $need_packages requires root privileges." \
  || sudo apt-get install -y $need_packages \
  || FATAL "failed to install $need_packages"
sudo snap install jq

[ "${OSM_STACK_NAME}" == "osm" ] || OSM_DOCKER_WORK_DIR="$OSM_WORK_DIR/
    stack/$OSM_STACK_NAME"
OSM_K8S_WORK_DIR="$OSM_DOCKER_WORK_DIR/osm_pods" && OSM_NAMESPACE_VOL="
    ${OSM_HOST_VOL}/${OSM_STACK_NAME}"

#Installation starts here
wget -q -O- https://osm-download.etsi.org/ftp/osm-11.0-eleven/README.
```

```
    txt &> /dev/null

install_osm
touch ${HOME}/5gmeta/logs/osm11_installed
echo -e "\nDONE"
exit 0
```

Listing A.3: OSM "cits" pipeline's KNFD and NSD descriptors

```
vnfd:
  id: cits_knfd
  product-name: cits_knfd
  description: CITS Pipeline
  version: '1.0'
  mgmt-cp: knf-cp0-ext
  df:
  - id: default-df
  kdu:
  - helm-chart: 5gmeta/helloworld-chart
    name: cits
  k8s-cluster:
    nets:
    - id: mgmtnet
  ext-cpd:
  - id: knf-cp0-ext
    k8s-cluster-net: mgmtnet

-----------------------------------------

nsd:
  nsd:
  - id: cits_nsd
    name: cits
    description: CITS Pipeline
    version: '1.0'
    vnfd-id:
    - cits_knfd
    df:
    - id: default-df
      vnf-profile:
      - id: "1"
        vnfd-id: cits_knfd
        virtual-link-connectivity:
        - virtual-link-profile-id: cits_nsd_vld0
          constituent-cpd-id:
          - constituent-base-element-id: "1"
            constituent-cpd-id: knf-cp0-ext
    virtual-link-desc:
    - id: cits_nsd_vld0
      mgmt-network: true
```

Listing A.4: Resource introspection script

```
#!/bin/bash

[ -z "$nodeip" ] && nodeip=$(kubectl get nodes -o jsonpath="{.items[0].
    status.addresses[0].address}")
```

```
allocatable_memory()
{
   allocatable_memory=$(curl -s http://$nodeip:9090/api/v1/query\?query
      \=eagle_node_resource_allocatable_memory_bytes | jq '.data.
      result[0].value[1] | tonumber')
   echo $allocatable_memory | awk '{print $1/1024/1024/1024}'
}

allocatable_cpu()
{
   allocatable_cpu=$(curl -s http://$nodeip:9090/api/v1/query\?query\=
      eagle_node_resource_allocatable_cpu_cores | jq '.data.result[0].
      value[1] | tonumber')
   echo $allocatable_cpu
}

schedulable_memory()
{
   allocatable_memory=$(curl -s http://$nodeip:9090/api/v1/query\?query
      \=eagle_node_resource_allocatable_memory_bytes | jq '.data.
      result[0].value[1] | tonumber')
   memory_usage=$(curl -s http://$nodeip:9090/api/v1/query\?query\=
      eagle_node_resource_usage_memory_bytes | jq '.data.result[0].
      value[1] | tonumber')
   memory_requests=$(curl -s http://$nodeip:9090/api/v1/query\?query\=
      eagle_node_resource_requests_memory_bytes | jq '.data.result[0].
      value[1] | tonumber')
   memory_limits=$(curl -s http://$nodeip:9090/api/v1/query\?query\=
      eagle_node_resource_limits_memory_bytes | jq '.data.result[0].
      value[1] | tonumber')
   max_memory=$(printf '%d\n' $memory_usage $memory_requests
      $memory_limits | sort -nr | head -1)

   schedulable_memory=$((allocatable_memory-max_memory))
   echo $schedulable_memory | awk '{print $1/1024/1024/1024}'
}

schedulable_cpu()
{
   allocatable_cpu=$(curl -s http://$nodeip:9090/api/v1/query\?query\=
      eagle_node_resource_allocatable_cpu_cores | jq '.data.result[0].
      value[1] | tonumber')
   cpu_usage=$(curl -s http://$nodeip:9090/api/v1/query\?query\=
      eagle_node_resource_usage_cpu_cores | jq '.data.result[0].value
      [1] | tonumber')
   cpu_requests=$(curl -s http://$nodeip:9090/api/v1/query\?query\=
      eagle_node_resource_requests_cpu_cores | jq '.data.result[0].
      value[1] | tonumber')
   cpu_limits=$(curl -s http://$nodeip:9090/api/v1/query\?query\=
      eagle_node_resource_limits_cpu_cores | jq '.data.result[0].value
      [1] | tonumber')
   max_cpu=$(printf '%f\n' $cpu_usage $cpu_requests $cpu_limits | sort
      -nr | head -1)

   schedulable_cpu=$(echo "$allocatable_cpu-$max_cpu" | bc)
   echo $schedulable_cpu
}
```

```
help()
{
   echo "Program to get the allocatable and schedulable resources from
      a k8s node."
   echo
   echo "Syntax: introspection [-m|c|g|h]"
   echo "options:"
   echo "m:     Print the allocatable memory."
   echo "c:     Print the allocatable CPUs."
   echo "M:     Print the schedulable memory."
   echo "C:     Print the schedulable CPUs."
   echo "g:     Print the schedulable GPUs (if available)."
   echo "h:     Help."
   echo
}

[ $# -eq 0 ] && help
while getopts "mcMCgh" option; do
   case $option in
      m) # Display allocatable memory
         allocatable_memory
         exit;;
      c) # Display allocatable cpu
         allocatable_cpu
         exit;;
      M) # Display schedulable memory
         schedulable_memory
         exit;;
      C) # Display schedulable cpu
         schedulable_cpu
         exit;;
      g) # Display schedulable gcpu
#        schedulable_gpu
         exit;;
      h | *) # Display help
         help
         exit;;
     \?) # Invalid option
         echo "Error: Invalid option"
         exit;;
   esac
done
```

Listing A.5: Confluent Kafka values file

```
## ------------------------------------------------------
## Zookeeper
## ------------------------------------------------------
cp-zookeeper:
  enabled: true
  servers: 3
  image: confluentinc/cp-zookeeper
  imageTag: 7.1.0
  ## Optionally specify an array of imagePullSecrets. Secrets must be
     manually created in the namespace.
  ## https://kubernetes.io/docs/concepts/containers/images/#specifying-
     imagepullsecrets-on-a-pod
```

```
  imagePullSecrets:
  #  - name: "regcred"
  heapOptions: "-Xms512M -Xmx512M"
  persistence:
    enabled: true
    ## The size of the PersistentVolume to allocate to each Zookeeper
       Pod in the StatefulSet. For
    ## production servers this number should likely be much larger.
    ##
    ## Size for Data dir, where ZooKeeper will store the in-memory
       database snapshots.
    dataDirSize: 5Gi
    # dataDirStorageClass: ""
    ## Size for data log dir, which is a dedicated log device to be
       used, and helps avoid competition between logging and snaphots.
    dataLogDirSize: 5Gi
    # dataLogDirStorageClass: ""

  # TODO: find correct security context for user in this zk-image
  securityContext:
    runAsUser: 0

  resources:
  ## If you do want to specify resources, uncomment the following lines
     , adjust them as necessary,
  ## and remove the curly braces after 'resources:'
    limits:
#      cpu: 100m
      memory: 1Gi
    requests:
#      cpu: 100m
      memory: 768Mi

## ------------------------------------------------------
## Kafka
## ------------------------------------------------------
cp-kafka:
  enabled: true
  brokers: 3
  image: confluentinc/cp-enterprise-kafka
  imageTag: 7.1.0
  ## Optionally specify an array of imagePullSecrets. Secrets must be
     manually created in the namespace.
  ## https://kubernetes.io/docs/concepts/containers/images/#specifying-
     imagepullsecrets-on-a-pod
  imagePullSecrets:
  #  - name: "regcred"
  heapOptions: "-Xms512M -Xmx512M"
  persistence:
    enabled: true
    # storageClass: ""
    size: 20Gi
    disksPerBroker: 1
  resources:
  ## If you do want to specify resources, uncomment the following lines
     , adjust them as necessary,
  ## and remove the curly braces after 'resources:'
    limits:
```

```
#      cpu: 100m
       memory: 1Gi
     requests:
#      cpu: 100m
       memory: 768Mi
  securityContext:
    runAsUser: 0

## ------------------------------------------------------
## Schema Registry
## ------------------------------------------------------
cp-schema-registry:
  enabled: true
  image: confluentinc/cp-schema-registry
  imageTag: 6.1.0
  ## Optionally specify an array of imagePullSecrets. Secrets must be
     manually created in the namespace.
  ## https://kubernetes.io/docs/concepts/containers/images/#specifying-
     imagepullsecrets-on-a-pod
  imagePullSecrets:
  #  - name: "regcred"
  heapOptions: "-Xms512M -Xmx512M"
  resources: {}
  ## If you do want to specify resources, uncomment the following lines
     , adjust them as necessary,
  ## and remove the curly braces after 'resources:'
  #  limits:
  #   cpu: 100m
  #   memory: 128Mi
  #  requests:
  #   cpu: 100m
  #   memory: 128Mi

## ------------------------------------------------------
## REST Proxy
## ------------------------------------------------------
cp-kafka-rest:
  enabled: true
  image: confluentinc/cp-kafka-rest
  imageTag: 7.1.0
  ## Optionally specify an array of imagePullSecrets. Secrets must be
     manually created in the namespace.
  ## https://kubernetes.io/docs/concepts/containers/images/#specifying-
     imagepullsecrets-on-a-pod
  imagePullSecrets:
  #  - name: "regcred"
  heapOptions: "-Xms512M -Xmx512M"
  resources: {}
  ## If you do want to specify resources, uncomment the following lines
     , adjust them as necessary,
  ## and remove the curly braces after 'resources:'
  #  limits:
  #   cpu: 100m
  #   memory: 128Mi
  #  requests:
  #   cpu: 100m
  #   memory: 128Mi
```

```
## ------------------------------------------------------
## Kafka Connect
## ------------------------------------------------------
cp-kafka-connect:
  enabled: true
#  image: confluentinc/cp-kafka-connect
  image: 5gmeta/kafka-connect
  imageTag: 6.1.0
  ## Optionally specify an array of imagePullSecrets. Secrets must be
     manually created in the namespace.
  ## https://kubernetes.io/docs/concepts/containers/images/#specifying-
     imagepullsecrets-on-a-pod
  imagePullSecrets:
    - name: "regcred"
  heapOptions: "-Xms512M -Xmx512M"
  resources: {}
  ## If you do want to specify resources, uncomment the following lines
     , adjust them as necessary,
  ## and remove the curly braces after 'resources:'
  #  limits:
  #   cpu: 100m
  #   memory: 128Mi
  #  requests:
  #   cpu: 100m
  #   memory: 128Mi

## ------------------------------------------------------
## KSQL Server
## ------------------------------------------------------
cp-ksql-server:
  enabled: true
  image: confluentinc/cp-ksqldb-server
  imageTag: 7.1.0
  ## Optionally specify an array of imagePullSecrets. Secrets must be
     manually created in the namespace.
  ## https://kubernetes.io/docs/concepts/containers/images/#specifying-
     imagepullsecrets-on-a-pod
  imagePullSecrets:
  #  - name: "regcred"
  heapOptions: "-Xms512M -Xmx512M"
  ksql:
    headless: false

## ------------------------------------------------------
## Control Center
## ------------------------------------------------------
cp-control-center:
  enabled: false
  image: confluentinc/cp-enterprise-control-center
  imageTag: 7.1.0
  ## Optionally specify an array of imagePullSecrets. Secrets must be
     manually created in the namespace.
  ## https://kubernetes.io/docs/concepts/containers/images/#specifying-
     imagepullsecrets-on-a-pod
  imagePullSecrets:
  #  - name: "regcred"
  heapOptions: "-Xms512M -Xmx512M"
  resources: {}
```

```
  ## If you do want to specify resources, uncomment the following lines
     , adjust them as necessary,
  ## and remove the curly braces after 'resources:'
  #  limits:
  #   cpu: 100m
  #   memory: 128Mi
  #  requests:
  #   cpu: 100m
  #   memory: 128Mi
```

Listing A.6: Connaisseur Helm values file

```
# configure Connaisseur deployment
deployment:
  replicasCount: 3
  image: securesystemsengineering/connaisseur:v2.6.3
  imagePullPolicy: IfNotPresent
  # imagePullSecrets contains an optional list of Kubernetes Secrets,
     in Connaisseur namespace,
  # that are needed to access the registry containing Connaisseur image
     .
  # imagePullSecrets:
  # - name: "my-container-secret"
  failurePolicy: Fail  # use 'Ignore' to fail open if Connaisseur
     becomes unavailable
  # Use 'reinvocationPolicy: IfNeeded' to be called again as part of
     the admission evaluation if the object
  # being admitted is modified by other admission plugins after the
     initial webhook call
  # Note that if Connaisseur is invoked a second time, the policy to be
      applied might change in between.
  # Make sure, your Connaisseur policies are set up to handle multiple
     mutations of the image originally
  # specified in the manifest, e.g. my.private.registry/image:1.0.0 and
      my.private.registry/image@sha256:<hash-of-1.0.0-image>
  reinvocationPolicy: Never
  resources:
    limits:
      cpu: 1000m
      memory: 512Mi
    requests:
      cpu: 100m
      memory: 128Mi
  nodeSelector: {}
  tolerations: []
  affinity:
    podAntiAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
        - podAffinityTerm:
            labelSelector:
              matchExpressions:
                - key: app.kubernetes.io/instance
                  operator: In
                  values:
                    - connaisseur
            topologyKey: kubernetes.io/hostname
          weight: 100
  #annotations:  # uncomment when using Kubernetes prior v1.19
```

```
  #  seccomp.security.alpha.kubernetes.io/pod: runtime/default  #
     uncomment when using Kubernetes prior v1.19
  securityContext:
    allowPrivilegeEscalation: false
    capabilities:
      drop:
        - ALL
    privileged: false
    readOnlyRootFilesystem: true
    runAsNonRoot: true
    runAsUser: 10001  # remove when using openshift or OKD 4
    runAsGroup: 20001  # remove when using openshift or OKD 4
    seccompProfile:  # remove when using Kubernetes prior v1.19,
       openshift or OKD 4
      type: RuntimeDefault  # remove when using Kubernetes prior v1.19,
          openshift or OKD 4
  # PodSecurityPolicy is deprecated as of Kubernetes v1.21, and will be
     removed in v1.25
  podSecurityPolicy:
    enabled: false
    name: ["connaisseur-psp"]  # list of PSPs to use, "connaisseur-psp"
        is the project-provided default
  envs: {} # dict of additional environment variables, which will be
     stored as a secret and injected into the Connaisseur pods
  # Extra config
  extraContainers: []
  extraVolumes: []
  extraVolumeMounts: []

# configure Connaisseur service
service:
  type: ClusterIP
  port: 443

### VALIDATORS ###
# validators are a set of configurations (types, public keys,
   authentication)
# that can be used for validating one or multiple images (or image
   signatures).
# they are tied to their respective image(s) via the image policy below
   . there
# are a few handy validators pre-configured.
validators:
  # static validator that allows each image
  - name: allow
    type: static
    approve: true
  # static validator that denies each image
  - name: deny
    type: static
    approve: false
  # the `default` validator is used if no validator is specified in
     image policy
  - name: default
    type: notaryv1  # or other supported validator (e.g. "cosign")
    host: notary.docker.io # configure the notary server for notaryv1
       or rekor url for cosign
    trust_roots:
```

```
    # the `default` key is used if no key is specified in image policy
    - name: default
      key: |  # enter your public key below
        -----BEGIN PUBLIC KEY-----
        XXX==
        -----END PUBLIC KEY-----
    #cert: |  # in case the trust data host is using a self-signed
       certificate
    #  -----BEGIN CERTIFICATE-----
    #  ...
    #  -----END CERTIFICATE-----
    auth:  # credentials in case the trust data requires authentication
    #  # either (preferred solution)
    #  secret_name: mysecret  # reference a k8s secret in the form
       required by the validator type (check the docs)
    #  # or (only for notaryv1 validator)
      username: XXX
      password: XXX
  # pre-configured nv1 validator for public notary from Docker Hub
  - name: dockerhub-basics
    type: notaryv1
    host: notary.docker.io
    trust_roots:
      # public key for official docker images (https://hub.docker.com/
         search?q=&type=image&image_filter=official)
      # !if not needed feel free to remove the key!
      - name: docker-official
        key: |
          -----BEGIN PUBLIC KEY-----
          MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEOXYta5TgdCwXTCnLU09W5T4M4r9f

          QQrqJuADP6U7g5r9ICgPSmZuRHP/1AYUfOQW3baveKsT969EfELKj1lfCA==
          -----END PUBLIC KEY-----
      # public key securesystemsengineering repo including Connaisseur
         images
      # !this key is critical for Connaisseur!
      - name: securesystemsengineering-official
        key: |
          -----BEGIN PUBLIC KEY-----
          MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEsx28WV7BsQfnHF1kZmpdCTTLJaWe

          d0CA+JOi8H4REuBaWSZ5zPDe468WuOJ6f71E7WFg3CVEVYHuoZt2UYbN/Q==
          -----END PUBLIC KEY-----

### IMAGE POLICY ###
# the image policy ties validators and images together whereby always
   only the most specific rule (pattern)
# is applied. specify if and how images should be validated by which
   validator via the validator name.
policy:
  - pattern: "*:*"
  - pattern: "docker.io/library/*:*"
    validator: dockerhub-basics
    with:
      trust_root: docker-official
  - pattern: "k8s.gcr.io/*:*"
    validator: allow
  - pattern: "docker.io/securesystemsengineering/*:*"
```

```
    validator: dockerhub-basics
    with:
      trust_root: securesystemsengineering-official

# in detection mode, deployment will not be denied, but only prompted
# and logged. this allows testing the functionality without
# interrupting operation.
detectionMode: false

# namespaced validation allows to restrict the namespaces that will be
    subject to Connaisseur verification.
# when enabled, based on namespaced validation mode ('ignore' or '
    validate')
# - either all namespaces with label "securesystemsengineering.
    connaisseur/webhook=ignore" are ignored
# - or only namespaces with label "securesystemsengineering.connaisseur
    /webhook=validate" are validated.
# warning: enabling namespaced validation, allows roles with edit
    permission on a namespace to disable
# validation for that namespace
namespacedValidation:
  enabled: false
  mode: ignore  # 'ignore' or 'validate'

# automatic child approval determines how admission of Kubernetes child
     resources is handled by Connaisseur.
# per default, Connaisseur validates and mutates all resources, e.g.
    deployments, replicaSets, pods, and
# automatically approves child resources of those to avoid duplicate
    validation and inconsistencies with the
# image policy. when disabled Connaisseur will only validate and mutate
     pods. check the docs for more
# information.
# NOTE: configuration of automatic child approval is in EXPERIMENTAL
    state.
automaticChildApproval:
  enabled: true

# debug: true


# The "logLevel" configuration option adds a partial redundancy to the
    `debug` setting.
# Removing the `debug` setting is a breaking change though - we are
    going to remove the `debug` setting in the context of a larger
    refactoring to avoid multiple breaking releases.
# Option to configure the log level. Either one of `DEBUG`, `INFO`, `
    WARNING`, `ERROR`, `CRITICAL`. Defaults to `INFO`
logLevel: INFO

# alerting is implemented in form of simple POST requests with json
    payload
# you can use and/or adapt the predefined Slack/OpsGenie/Keybase
    templates and the examples below
# to channel alert notifications to Slack/OpsGenie/Keybase or create a
    custom template for a customized alert
# payload to use with a simple POST request to the receiver_url to
    receive alerts.
```

```
# Parameters you can use in your templates are "alert_message", "
    priority", "connaisseur_pod_id", "cluster",
# "timestamp", "request_id" and "images" each one basically meaning
    what their names indicate
#
# Below is an example config

#alerting:
#  cluster_identifier: example-cluster-staging-europe # defaults to "
    not specified"
#  admit_request:
#    templates:
#      # <template> needs to be chosen such that <template>.json
    matches one of the file names
#      # in the ./alert_payload_templates directory
#      - template: opsgenie #REQUIRED!
#        receiver_url: https://api.eu.opsgenie.com/v2/alerts #REQUIRED!
#        priority: 4 #(defaults to 3)
#        custom_headers: ["Authorization: GenieKey <Your-Genie-Key>"]
#        payload_fields:
#          responders:
#            - username: "testuser@testcompany.de"
#              type: user
#          visibleTo:
#            - username: "testuser@testcompany.de"
#              type: user
#          tags:
#            - "deployed_an_image"
#        fail_if_alert_sending_fails: True  # (defaults to False,
    turning it to True will make Connaisseur deny your
#                                           # deployment (even in
    detection mode))
#      - template: slack #REQUIRED!
#        receiver_url: https://hooks.slack.com/services/<Your-Slack-
    Hook-Path>
#        priority: 1
#  reject_request:
#    templates:
#      - template: keybase  #REQUIRED!
#        receiver_url: https://bots.keybase.io/webhookbot/<Your-Keybase
    -Hook-Token>
#        fail_if_alert_sending_fails: True
```

Listing A.7: SLA API definition and logic

```
openapi: 3.0.1
info:
  title: SLA Edge
  description: API to manage the SLAs and reservations in a 5GMETA MEC
     Server. The SLA API has the
    scope to consent the request of a certain service level agreement
       and receive
    the confimation or reservation. A request could be deleted. A
       reservation cannot
    be modified or updated."
#  termsOfService: http://swagger.io/terms/
  contact:
    name: 5GMETA
```

```
    url: https://5gmeta-project.eu/
  license:
    name: Apache 2.0
    url: http://www.apache.org/licenses/LICENSE-2.0.html
  version: 1.0.0
externalDocs:
  description: Find out more about 5GMETA
  url: https://5gmeta-project.eu/
servers:
- url: http://localhost/
- url: https://localhost/
tags:
- name: sla
  description: Operations about SLAs
#  externalDocs:
#    description: Find out more
#    url: http://swagger.io
- name: reservation
  description: Operations about SLA reservations
paths:
  /sla:
    post:
      tags:
      - sla
      summary: Add a new SLA
      operationId: post_sla
      requestBody:
        description: SLA object that needs to be added
        content:
          application/json:
            schema:
              x-body-name: payload
              $ref: '#/components/schemas/SLA'
        required: true
      responses:
        200:
          description: SLA successfully added
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/SLA'
        400:
          description: Invalid SLA
        401:
          description: The SLA level already exists
      x-openapi-router-controller: openapi_server.controllers.
          sla_controller
    get:
      tags:
      - sla
      summary: Get SLAs
      description: Get SLAs
      operationId: get_sla
      responses:
        200:
          description: Success
          content:
            application/json:
```

```
            schema:
              $ref: '#/components/schemas/SLA'
    x-openapi-router-controller: openapi_server.controllers.
       sla_controller
/sla/{sla_level}:
  get:
    tags:
    - sla
    summary: Get a SLA
    description: Returns a single SLA
    operationId: get_sla_item
    parameters:
    - name: sla_level
      in: path
      description: Specify the SLA level to get the information
      required: true
      schema:
        type: string
    responses:
      200:
        description: Success
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/SLA'
      400:
        description: Invalid SLA
      404:
        description: SLA not found
    x-openapi-router-controller: openapi_server.controllers.
       sla_controller
  patch:
    tags:
    - sla
    summary: Update a SLA
    operationId: patch_sla
    parameters:
    - name: sla_level
      in: path
      description: Specify the SLA level to modify the requeriments
      required: true
      schema:
        type: string
    requestBody:
      content:
        application/json:
          schema:
            x-body-name: payload
            $ref: '#/components/schemas/SLA'
      required: true
    responses:
      200:
        description: SLA successfully updated
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/SLA'
      400:
```

```
          description: Invalid SLA
        404:
          description: SLA not found
      x-openapi-router-controller: openapi_server.controllers.
          sla_controller
    delete:
      tags:
      - sla
      summary: Delete a SLA
      operationId: delete_sla
      parameters:
      - name: sla_level
        in: path
        description: Specify the SLA level to delete the SLA
        required: true
        schema:
          type: string
      responses:
        200:
          description: SLA successfully deleted
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/SLA'
        400:
          description: Invalid SLA
        404:
          description: SLA not found
      x-openapi-router-controller: openapi_server.controllers.
          sla_controller
  /reservation:
    post:
      tags:
      - reservation
      summary: Make a SLA reservation
      operationId: post_reservation
      requestBody:
        content:
          application/json:
            schema:
              x-body-name: payload
              $ref: '#/components/schemas/Reservation'
        required: true
      responses:
        200:
          description: Reservation successfully made
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Reservation'
        400:
          description: Invalid reservation
        404:
          description: The selected SLA level is not available on this
             Edge server
        405:
          description: The selected datatype is not available on this
             Edge server
```

```
        501:
          description: There are no enough resources to make the
             reservation
        502:
          description: Error orchestrating the pipeline
      x-openapi-router-controller: openapi_server.controllers.
         reservation_controller
    get:
      tags:
      - reservation
      summary: Get SLA reservations
      description: Get reservations
      operationId: get_reservation
      responses:
        200:
          description: Success
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Reservation'
      x-openapi-router-controller: openapi_server.controllers.
         reservation_controller
  /reservation/{reservation_id}:
    get:
      tags:
      - reservation
      summary: Get a SLA reservation
      description: Returns a single reservation
      operationId: get_reservation_item
      parameters:
      - name: reservation_id
        in: path
        description: Specify the Reservation ID to get the information
        required: true
        schema:
          type: string
      responses:
        200:
          description: Success
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Reservation'
        400:
          description: Invalid reservation
        404:
          description: Reservation not found
      x-openapi-router-controller: openapi_server.controllers.
         reservation_controller
    delete:
      tags:
      - reservation
      summary: Delete a SLA reservation
      operationId: delete_reservation
      parameters:
      - name: reservation_id
        in: path
        description: Specify the Reservation ID to delete the
```

```
            reservation
        required: true
        schema:
          type: string
      responses:
        200:
          description: Reservation successfully deleted
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Reservation'
        400:
          description: Invalid reservation
        404:
          description: Reservation not found
      x-openapi-router-controller: openapi_server.controllers.
          reservation_controller
components:
  schemas:
    SLA:
      title: SLA
      example:
        sla_level: medium
        total_cpu: 4
        total_memory: 4
        gpu: false
      required:
      - total_cpu
      - total_memory
      - gpu
      type: object
      properties:
        sla_level:
          type: string
          description: SLA level
        total_cpu:
          type: integer
          description: Total CPUs available in the SLA
          format: int64
        total_memory:
          type: integer
          description: Total memory available in the SLA
          format: int64
        gpu:
          type: boolean
          description: GPU required True/False
      xml:
        name: SLA
    Reservation:
      title: Reservation
      example:
        data_type: cits
        sla_level: medium
        client_name: 5gmeta
      required:
      - data_type
      - sla_level
      type: object
```

```
      properties:
        sla_level:
          type: string
          description: Requested SLA level
        data_type:
          type: string
          description: Requested data type
        client_name:
          type: string
          description: Client who requested data
  responses:
    MaskError:
      description: When any error occurs on mask
      content: {}
    ParseError:
      description: When a mask can't be parsed
      content: {}
  securitySchemes:
    auth:
      type: oauth2
      flows:
        authorizationCode:
          authorizationUrl: http://192.168.15.175:8080/auth/realms/5
              gmeta/protocol/openid-connect/auth
          tokenUrl: http://192.168.15.175:8080/auth/realms/5gmeta/
              protocol/openid-connect/token
          scopes:
            write:pets: modify pets in your account
            read:pets: read your pets
#            uid: Unique identifier of the user accessing the service.
      x-tokenInfoFunc: openapi_server.controllers.auth_controller.
          check_petstore_auth
#      x-tokenInfoFunc: openapi_server.controllers.auth_controller.
    token_info
#      x-scopeValidateFunc: openapi_server.controllers.auth_controller.
    validate_scope_petstore_auth
openapi: 3.0.1
info:
  title: SLA Edge
  description: API to manage the SLAs and reservations in a 5GMETA MEC
      Server. The SLA API has the
    scope to consent the request of a certain service level agreement
        and receive
    the confimation or reservation. A request could be deleted. A
        reservation cannot
    be modified or updated."
#  termsOfService: http://swagger.io/terms/
  contact:
    name: 5GMETA
    email: 5gmeta@vicomtech.org
    url: https://5gmeta-project.eu/
  license:
    name: Apache 2.0
    url: http://www.apache.org/licenses/LICENSE-2.0.html
  version: 1.0.0
externalDocs:
  description: Find out more about 5GMETA
  url: https://5gmeta-project.eu/
```

```
servers:
- url: http://localhost/
- url: https://localhost/
tags:
- name: sla
  description: Operations about SLAs
#  externalDocs:
#    description: Find out more
#    url: http://swagger.io
- name: reservation
  description: Operations about SLA reservations
paths:
  /sla:
    post:
      tags:
      - sla
      summary: Add a new SLA
      operationId: post_sla
      requestBody:
        description: SLA object that needs to be added
        content:
          application/json:
            schema:
              x-body-name: payload
              $ref: '#/components/schemas/SLA'
        required: true
      responses:
        200:
          description: SLA successfully added
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/SLA'
        400:
          description: Invalid SLA
        401:
          description: The SLA level already exists
      x-openapi-router-controller: openapi_server.controllers.
          sla_controller
    get:
      tags:
      - sla
      summary: Get SLAs
      description: Get SLAs
      operationId: get_sla
      responses:
        200:
          description: Success
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/SLA'
      x-openapi-router-controller: openapi_server.controllers.
          sla_controller
  /sla/{sla_level}:
    get:
      tags:
      - sla
```

```
    summary: Get a SLA
    description: Returns a single SLA
    operationId: get_sla_item
    parameters:
    - name: sla_level
      in: path
      description: Specify the SLA level to get the information
      required: true
      schema:
        type: string
    responses:
      200:
        description: Success
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/SLA'
      400:
        description: Invalid SLA
      404:
        description: SLA not found
    x-openapi-router-controller: openapi_server.controllers.
       sla_controller
  patch:
    tags:
    - sla
    summary: Update a SLA
    operationId: patch_sla
    parameters:
    - name: sla_level
      in: path
      description: Specify the SLA level to modify the requeriments
      required: true
      schema:
        type: string
    requestBody:
      content:
        application/json:
          schema:
            x-body-name: payload
            $ref: '#/components/schemas/SLA'
      required: true
    responses:
      200:
        description: SLA successfully updated
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/SLA'
      400:
        description: Invalid SLA
      404:
        description: SLA not found
    x-openapi-router-controller: openapi_server.controllers.
       sla_controller
  delete:
    tags:
    - sla
```

```
    summary: Delete a SLA
    operationId: delete_sla
    parameters:
    - name: sla_level
      in: path
      description: Specify the SLA level to delete the SLA
      required: true
      schema:
        type: string
    responses:
      200:
        description: SLA successfully deleted
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/SLA'
      400:
        description: Invalid SLA
      404:
        description: SLA not found
    x-openapi-router-controller: openapi_server.controllers.
       sla_controller
/reservation:
  post:
    tags:
    - reservation
    summary: Make a SLA reservation
    operationId: post_reservation
    requestBody:
      content:
        application/json:
          schema:
            x-body-name: payload
            $ref: '#/components/schemas/Reservation'
      required: true
    responses:
      200:
        description: Reservation successfully made
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Reservation'
      400:
        description: Invalid reservation
      404:
        description: The selected SLA level is not available on this
           Edge server
      405:
        description: The selected datatype is not available on this
           Edge server
      501:
        description: There are no enough resources to make the
           reservation
      502:
        description: Error orchestrating the pipeline
    x-openapi-router-controller: openapi_server.controllers.
       reservation_controller
  get:
```

```
      tags:
      - reservation
      summary: Get SLA reservations
      description: Get reservations
      operationId: get_reservation
      responses:
        200:
          description: Success
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Reservation'
      x-openapi-router-controller: openapi_server.controllers.
          reservation_controller
  /reservation/{reservation_id}:
    get:
      tags:
      - reservation
      summary: Get a SLA reservation
      description: Returns a single reservation
      operationId: get_reservation_item
      parameters:
      - name: reservation_id
        in: path
        description: Specify the Reservation ID to get the information
        required: true
        schema:
          type: string
      responses:
        200:
          description: Success
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Reservation'
        400:
          description: Invalid reservation
        404:
          description: Reservation not found
      x-openapi-router-controller: openapi_server.controllers.
          reservation_controller
    delete:
      tags:
      - reservation
      summary: Delete a SLA reservation
      operationId: delete_reservation
      parameters:
      - name: reservation_id
        in: path
        description: Specify the Reservation ID to delete the
            reservation
        required: true
        schema:
          type: string
      responses:
        200:
          description: Reservation successfully deleted
          content:
```

```
            application/json:
              schema:
                $ref: '#/components/schemas/Reservation'
        400:
          description: Invalid reservation
        404:
          description: Reservation not found
      x-openapi-router-controller: openapi_server.controllers.
         reservation_controller
components:
  schemas:
    SLA:
      title: SLA
      example:
        sla_level: medium
        total_cpu: 4
        total_memory: 4
        gpu: false
      required:
      - total_cpu
      - total_memory
      - gpu
      type: object
      properties:
        sla_level:
          type: string
          description: SLA level
        total_cpu:
          type: integer
          description: Total CPUs available in the SLA
          format: int64
        total_memory:
          type: integer
          description: Total memory available in the SLA
          format: int64
        gpu:
          type: boolean
          description: GPU required True/False
      xml:
        name: SLA
    Reservation:
      title: Reservation
      example:
        data_type: cits
        sla_level: medium
        client_name: 5gmeta
      required:
      - data_type
      - sla_level
      type: object
      properties:
        sla_level:
          type: string
          description: Requested SLA level
        data_type:
          type: string
          description: Requested data type
        client_name:
```

```
          type: string
          description: Client who requested data
  responses:
    MaskError:
      description: When any error occurs on mask
      content: {}
    ParseError:
      description: When a mask can't be parsed
      content: {}
  securitySchemes:
    auth:
      type: oauth2
      flows:
        authorizationCode:
          authorizationUrl: http://X.X.X.X:8080/auth/realms/5gmeta/
             protocol/openid-connect/auth
          tokenUrl: http://X.X.X.X:8080/auth/realms/5gmeta/protocol/
             openid-connect/token
```

```
from openapi_server.config.config import db
from openapi_server.models.sla import SLA, SLASchema
from sqlalchemy.exc import IntegrityError

def post_sla(payload):
    """Add a new SLA

    :param payload: SLA object that needs to be added
    :type payload: dict | bytes

    :rtype: None
    """
    #sla = SLA.query.filter(SLA.sla_level == payload["sla_level"]).
       one_or_none()

    #if sla is None:
    try:
        schema = SLASchema()

        # Deserialize the received data
        new_sla = schema.load(payload)

        # Add the sla to the database
        db.session.add(new_sla)
        db.session.commit()

        # Serialize and return the newly created sla in the response
        data = schema.dump(new_sla)

        return data, 200
#    else:
    except IntegrityError:
        return "The SLA level already exists", 401


def get_sla():
    """Get SLAs
```

```
    :rtype: None
    """
    slas = SLA.query.all()

    slas_schema = SLASchema(many=True)

    data = slas_schema.dump(slas)

    return data, 200


def get_sla_item(sla_level):
    """Find SLA by ID

    :param sla_level: Specify the SLA ID to get the information
    :type sla_level: string

    :rtype: SLA
    """
    try:
        sla = SLA.query.filter(SLA.sla_level == sla_level).one()

        sla_schema = SLASchema()

        data = sla_schema.dump(sla)

        return data, 200
    except:
        return "SLA not found", 404


def patch_sla(payload, sla_level):
    """Updates a SLA

    :param payload:
    :type payload: dict | bytes
    :param sla_level: Specify the SLA ID to modify the requeriments
    :type sla_level: string

    :rtype: None
    """
    try:
        old_sla = SLA.query.filter(SLA.sla_level == sla_level).one()

        schema = SLASchema()

        new_sla = schema.load(payload)

        new_sla.sla_level = old_sla.sla_level

        db.session.merge(new_sla)
        db.session.commit()

        data = schema.dump(new_sla)

        return data, 200
    except:
        return "SLA not found", 404
```

```
def delete_sla(sla_level):
    """Deletes a SLA

    :param sla_level: Specify the SLA ID to delete the SLA
    :type sla_level: string

    :rtype: None
    """
    try:
        sla = SLA.query.filter(SLA.sla_level == sla_level).one()

        db.session.delete(sla)

        db.session.commit()

        return "SLA successfully deleted", 200
    except:
        return "SLA not found", 404
```

```
import subprocess
import requests
import yaml
import sys
import uuid

from openapi_server.config.config import db
from openapi_server.models.reservation import Reservation,
   ReservationSchema
from openapi_server.models.sla import SLA, SLASchema

from sqlalchemy.sql import func

def post_reservation(payload):
    """Make SLA reservation

    :param payload:
    :type payload: dict | bytes

    :rtype: None
    """
#    if connexion.request.is_json:
#        payload = Reservation.from_dict(connexion.request.get_json())

    if SLA.query.filter(SLA.sla_level == payload["sla_level"]).
       one_or_none() is None:
        return "The selected SLA level is not available on this Edge
           Server", 404

    try:
        # Get the reserved resorced from the database
        cpu_to_reserve = SLA.query.with_entities(SLA.total_cpu).filter(
           SLA.sla_level == payload["sla_level"]).one()
        memory_to_reserve = SLA.query.with_entities(SLA.total_memory).
           filter(SLA.sla_level == payload["sla_level"]).one()
        reserved_total_cpu = 0
```

```
        reserved_total_memory = 0
        reserved_sla_types = Reservation.query.with_entities(
            Reservation.sla_level.distinct()).all()
        for sla_type in reserved_sla_types:
            sla_type_number = Reservation.query.filter(Reservation.
                sla_level == sla_type[0]).count()
            sla_type_cpu = SLA.query.with_entities(SLA.total_cpu).
                filter(SLA.sla_level == sla_type[0]).one()
            sla_type_memory = SLA.query.with_entities(SLA.total_memory)
                .filter(SLA.sla_level == sla_type[0]).one()
            reserved_total_cpu += sla_type_number * sla_type_cpu[0]
            reserved_total_memory += sla_type_number * sla_type_memory
                [0]

        # Get the allocatable and schedulable resources from the k8s
            node
        allocatable_cpu = float(subprocess.check_output(["./
            introspection.sh","-c"]))
        allocatable_memory = float(subprocess.check_output(["./
            introspection.sh","-m"]))
        schedulable_cpu = float(subprocess.check_output(["./
            introspection.sh","-C"]))
        schedulable_memory = float(subprocess.check_output(["./
            introspection.sh","-M"]))

        # Check if there is enough cpu and memory to make the
            reservation
        cpu_is_avaliable = False
        memory_is_avaliable = False
        if allocatable_cpu - reserved_total_cpu - cpu_to_reserve[0] > 0
             and schedulable_cpu - cpu_to_reserve[0] > 0:
            cpu_is_avaliable = True
        if allocatable_memory - reserved_total_memory -
            memory_to_reserve[0] > 0 and schedulable_memory -
            memory_to_reserve[0] > 0:
             memory_is_avaliable = True

        print("     allocatable_cpu: " + str(allocatable_cpu), file=sys
            .stderr)
        print("     reserved_total_cpu: " + str(reserved_total_cpu),
            file=sys.stderr)
        print("     cpu_to_reserve[0]: " + str(cpu_to_reserve[0]), file
            =sys.stderr)
        print("     schedulable_cpu: " + str(schedulable_cpu), file=sys
            .stderr)
        print("     cpu_is_avaliable: " + str(cpu_is_avaliable), file=
            sys.stderr)
        print("     allocatable_memory: " + str(allocatable_memory),
            file=sys.stderr)
        print("     reserved_total_memory: " + str(
            reserved_total_memory), file=sys.stderr)
        print("     memory_to_reserve[0]: " + str(memory_to_reserve[0])
            , file=sys.stderr)
        print("     schedulable_memory: " + str(schedulable_memory),
            file=sys.stderr)
        print("     memory_is_avaliable: " + str(memory_is_avaliable),
            file=sys.stderr)
```

```
        if cpu_is_avaliable and memory_is_avaliable:
            try:
#                orchestrator_ip = subprocess.getoutput("if [ -z
    $orchestratorip ]; then kubectl get nodes -o jsonpath='{.items[0].
    status.addresses[0].address}'; else echo $orchestratorip; fi")
#                orchestrator_ip = subprocess.check_output("if [ -z
    $nodeip ]; then kubectl get nodes -o jsonpath='{.items[0].status.
    addresses[0].address}'; else echo $nodeip; fi", shell=True, text=
    True)
                orchestrator_ip = subprocess.getoutput("if [ -z
                    $orchestratorip ]; then orchestratorip=nbi.osm.svc.
                    cluster.local && echo $orchestratorip; else echo
                    $orchestratorip; fi")

                # Get authorization token
                url = 'https://' + orchestrator_ip + ':9999/osm/admin/
                    v1/tokens'
                headers = {"Content-Type": "application/json"}
                data = '{"username": "admin","password": "admin","
                    project_id": "admin"}'
                response = requests.post(url, data=data, headers=
                    headers, verify=False)
                yaml_response = yaml.safe_load(response.content)
                bearer = "Bearer " + yaml_response["id"]

                # Get pipeline descriptor id
                url = 'https://' + orchestrator_ip + ':9999/osm/nsd/v1/
                    ns_descriptors'
                headers = {'Authorization': bearer}
                response = requests.get(url, headers=headers, verify=
                    False)
                datatype_response = yaml.safe_load(response.content)
                datatype_index = next((index for (index, key) in
                    enumerate(datatype_response) if key["name"] ==
                    payload["data_type"]), None)
                if datatype_index is None:
                    return "The selected datatype is not available on
                        this Edge Server", 405

                # Get vim id
                url = 'https://' + orchestrator_ip + ':9999/osm/admin/
                    v1/vims/'
                headers = {'Authorization': bearer}
                response = requests.get(url, headers=headers, verify=
                    False)
                vim_response = yaml.safe_load(response.content)
                vim_index = next((index for (index, key) in enumerate(
                    vim_response) if key["name"] == "5gmeta-vim"), None
                    )

                # Request to the orchestrator ip for deploying the
                    pipeline
                url = 'https://' + orchestrator_ip + ':9999/osm/nslcm/
                    v1/ns_instances_content'
                headers = {'Content-Type': 'application/json', '
                    Authorization': bearer}
                data = '{ "nsName": "' + payload["data_type"] + '", "
                    nsdId": "' + datatype_response[datatype_index]["_id
```

```
                    "] + '", "vimAccountId": "' + vim_response[
                    vim_index]["_id"] + '", "additionalParamsForVnf": [
                     { "member-vnf-index": "1", "additionalParamsForKdu
                    ": [ { "kdu_name": "' + payload["data_type"] + '",
                    "k8s-namespace": "' + payload["client_name"] + '",
                    "kdu-deployment-name": "' + payload["client_name"]
                    + '-' + payload["data_type"] + '-' + str(uuid.uuid1
                    ())[:8] + '" } ] } ] }'
                response = requests.post(url, data=data, headers=
                    headers, verify=False)
                yaml_response = yaml.safe_load(response.content)
            except:
                return "Error orchestrating the pipeline", 502

            schema = ReservationSchema()

            # Use osm instance id as reservation id
            payload["reservation_id"] = yaml_response["id"]

            # Deserialize the received data
            new_reservation = schema.load(payload)

            # Add the reservation to the database
            db.session.add(new_reservation)
            db.session.commit()

            # Serialize and return the newly created reservation in the
                response
            data = schema.dump(new_reservation)

            return data, 200
        else:
            return "There are no enough resources to make the
                reservation", 501
    except:
        return "Invalid reservation", 400


def get_reservation():
    """Get reservations

    :rtype: None
    """
    reservations = Reservation.query.all()

    reservations_schema = ReservationSchema(many=True)

    data = reservations_schema.dump(reservations)

    return data, 200


def get_reservation_item(reservation_id):
    """Find reservation by ID

    :param reservation_id: Specify the Reservation ID to get the
        information
    :type reservation_id: int
```

```
    :rtype: Reservation
    """
    try:
        reservation_schema = ReservationSchema()

        reservation = Reservation.query.filter(Reservation.
           reservation_id == reservation_id).one()

        data = reservation_schema.dump(reservation)

        return data, 200
    except:
        return "Reservation not found", 404


def delete_reservation(reservation_id):  # noqa: E501
    """Deletes reservation

    :param reservation_id: Specify the Reservation ID to delete the
       reservation
    :type reservation_id: int

    :rtype: None
    """
    try:
        try:
            orchestrator_ip = subprocess.getoutput("if [ -z
               $orchestratorip ]; then orchestratorip=nbi.osm.svc.
               cluster.local && echo $orchestratorip; else echo
               $orchestratorip; fi")

            # Get authorization token
            url = 'https://' + orchestrator_ip + ':9999/osm/admin/v1/
               tokens'
            headers = {"Content-Type": "application/json"}
            data = '{"username": "admin","password": "admin","
               project_id": "admin"}'
            response = requests.post(url, data=data, headers=headers,
               verify=False)
            yaml_response = yaml.safe_load(response.content)
            bearer = "Bearer " + yaml_response["id"]

            url = 'https://' + orchestrator_ip + ':9999/osm/nslcm/v1/
               ns_instances/' + reservation_id + '/terminate'
            headers = {'Content-Type': 'application/json', '
               Authorization': bearer}
            data = '{"autoremove": true}'
            response = requests.post(url, data=data, headers=headers,
               verify=False)
        except:
            return 400

        reservation = Reservation.query.filter(Reservation.
           reservation_id == reservation_id).one()

        db.session.delete(reservation)
```

```
        db.session.commit()

        return "Reservation successfully deleted", 200
    except:
        return "Reservation not found", 404
```