Aalto University
School of Electrical Engineering
Master's Programme in ICT Innovation - EIT Digital Master School

Giuseppe Superbo

Hard Multi-Tenancy Kubernetes approaches in a local 5G deployment:

Testing and evaluation of the available solutions

Master's Thesis
Espoo, July 22, 2022

Supervisors: Professor Fabrizio Granelli, University of Trento
Professor Jukka Manner, Aalto University
Advisor: M.Sc. (Tech.) Jan Zizka

Aalto University
School of Electrical Engineering
Master's Programme in ICT Innovation - EIT Digital Master School

ABSTRACT OF MASTER'S THESIS

Author: Giuseppe Superbo

Title:
Hard Multi-Tenancy Kubernetes approaches in a local 5G deployment: Testing and evaluation of the available solutions

Date: July 22, 2022 **Pages:** 94

Major: Cloud and Network Infrastructures **Code:** ELEC3059

Supervisors: Professor Fabrizio Granelli
Professor Jukka Manner

Advisor: M.Sc. (Tech.) Jan Zizka

With its capabilities, the fifth mobile network generation (5G) provides new opportunities and applications, which were not possible to exploit previously. In addition to novel technologies, also infrastructure management has been influenced by 5G. Micro-Operator business model was identified by University of Oulu and Nokia Oulu in 2017. They observed that TelCo stakeholders actors are no longer based only on a small group of Mobile Network Operators, but new actors are becoming relevant. Specifically, Micro-Operators are involved in the deployment of scalable services within local 5G deployments. In most cases, micro-operators deploy their containerized application by exploiting Kubernetes Orchestrator capabilities. At the same time, they share a pool of resources, which usually are provided by the infrastructure owner. Since current standards require strict data privacy and security mechanisms, hard multi-tenancy should be enforced in the design phase. Many solutions are available within the Kubernetes environment. Thus, it would be interesting to analyse how each approach affects network performance, which are very critical metrics in 5G requirements. In the context of this thesis, test and evaluation strategy consisted of deploying a set of applications with every identified hard multi-tenancy approach. The choice of the applications was based on the 5G application classes: extreme multi broadband, massive scale communication, ultra-reliable low latency communication. Once the deployment configurations were defined, data of specific critical application network performance metrics was collected for each of the approaches. By looking at the retrieved data, it is possible to observe that network performance is mainly affected by the application protocol and the corresponding transport protocol. Nevertheless, it is not possible to perform design choices only by looking at network performance. In conclusion, further investigation on other design characteristics may be required.

Keywords: 5G, Kubernetes, Multi-Tenancy, Small Cells, Micro-Operators, Edge Computing, Containerazation

Language: English

Acknowledgements

The development of this thesis would not have been possible without the support that I received during these months. First of all, I would like to thank my supervisors, Professor Jukka Manner, from Aalto University, and Professor Fabrizio Granelli, from University of Trento, for all the precious support and feedback received. They helped me to increase the quality of this thesis and provided very important technical suggestions during our meetings. I am really thankful to Nokia Oy, specifically to Markku Niiranen for giving me the opportunity to join the fantastic ECP Team and to my advisor Jan Zizka for all the technical support received during the testing activity. I am infinitely grateful to my mum and my dad that have been my first supporters since the first day of my life. They taught me how to behave correctly in this wild world but most importantly they always tought me to see only the positive side on everything. Hard work and respect have been always two main concepts that they shared with me. Thanks to them I am the person that I am today. My aunt, my grandmother and my grandfather that always shared with me their wisdom and love. All my relatives that believed in me and have always supported me with their love. My soul mate, Marta, who has never stopped to raise my mood whenever I was sad or anxious, who has never stopped to believe in me even if I annoyed her with my unsecurity but most importantly I know that I can always count on her love. Her parents Enzo and Giulia have always supported and helped me since the first day I met them. Thanks to my dear friends Elton, Gianluca, Marica, Liliana, Alessandro, Fabrizio, Giuseppe. Last but not least, I would like to thank all my university and Nokia mates for all the unforgettable moments.

Espoo, July 22, 2022

Giuseppe Superbo

Abbreviations and Acronyms

3GPP	3rd Generation Partnership Project
5G NR	5G New Radio
5GPPP	5G Infrastructure Public Private Partnership
μO	Micro Operator
API	Application programming interface
AR	Augmented reality
Cgroups	Control groups
C-RAN	Cloud-based Radio access network
CAPI	Cluster API
CAPN	Cluster API Provider Nested
CESC	Cloud Enabled Small Cell
CNCF	Cloud Native Computing Foundation
CNI	Container Network Interface
CRD	Custom resource definition
CUPS	Control and User Plane Separation
D2D	Device-to-Device communication
DRF	Dominant Resource Fairness
eMBB	Enhanced mobile broadband
EMS	Entity Management System
ETSI	European Telecommunications Standards Institute
FR	Frequency Range
GSMA	Global System for Mobile Communications Association
GPL	GNU General Public Licence
HNC	Hierarchical Namespaces Controller
ICE	Interactive Connectivty Establishment
IoT	Internet of Things
K8s	Kubernetes
LXC	Linux Container Runtime
MBS	Macrocell base stations

MCPTT	Mission Critical Push-To-Talk
M2M	Machine-to-Machine communication
MEC	Multi-access edge computing
MFC	Multi-access fog computing
MIMO	Multiple Input Multiple Output
mMTC	Massive machine type communications
MNO	Mobile Network Operator
MQTT	Message Queue Telemetry Transport
Multi-RAT	Multi Radio Access Technologies
NFV	Network Function Virtualization
NFVO	Network Virtual Function Orchestrator
NUC	Next Unit of Computing
QoE	Quality of experience
QoS	Quality of service
RBAC	Role-based access control
RTT	Round-trip time
SBS	Small cell base stations
SCaaS	Small-Cell-as-a-Service
SDMA	Spatial division multiple access
SDN	Software Defined Networking
SON	Self-Organizing Networks
STUN	Session Traversal Utilities for NAT protocol
UDN	Ultra dense network
URLLC	Ultra reliable low latency communications
VANET	Vehicles ad Hoc Networks
VM	Virtual machine
VMM	Virtual machine monitor
VNF	Virtual Network Function
VR	Virtual reality
WWW	World Wide Web
XaaS	Everything-as-a-Service

Contents

Chapter 1

Introduction

Mobile networks, also known as cellular networks, have been an essential component of modern society for the last 35 years. The ability to potentially communicate in a few seconds with different parts of the world has exponentially increased the pace of technological evolution and quality of life. In this perspective, like other modern standard technologies, the evolution of cellular networks can be described by exploiting the concept of "generation". Each generation is characterized by unique and novel features that enable new classes of applications and use cases. Despite the initial challenges of the first generations 1G and 2G in terms of global standardization and adoption rate [1], mobile networks have been successfully deployed all over the world, enabling new opportunities and services for society.

Currently, the state-of-the-art generation, 5G, is already available in many regions of the world, as Mobile Network Operators (MNO) are currently investing many resources in the deployment of novel technologies. Meanwhile, 3GPP is still developing and defining future standard releases for the fifth mobile network generation, to assess security and performance issues that have merged during the initial deployment stages in the mass market. On the other hand, from a statistical point of view, 4G is still the most widely adopted technology for mobile communication [2]. This is related to the fact that 5G is still in an early stage of the adoption phase and most of the available user equipment is not capable of exploiting 5G capabilities. In the "Mobile Economy 2021" report [3], Global System for Mobile Communications Association (GSMA) forecasts that by 2025, 20% of mobile communication will be based on 5G technologies , which means that, even in the near future, 4G will still be the most adopted cellular network generation.

So, looking at these initial statistics coming from the public market, how will 5G be exploited in the next few years? The latest trends of mobile networking are mainly characterized by highly demanding requirements that

can be summarized in three main typologies of scenarios: enhanced mobile broadband (eMBB), massive machine-type communications (mMTC) and ultra-reliable low latency communications (URLLC). Depending on the requirements of a specific use case, many key enabling technologies have been exploited for guaranteeing a specific level of performance. For instance, from a physical point of view, the 5G architecture is characterised by novel transmission access approaches. Multiple Input Multiple Output (MIMO) antennas and millimeterwave bands (24-100 GHz) increase throughput capabilities and provide optimal support for a higher number of connected devices compared to previous generations of mobile networks [4].

Furthermore, from a higher level perspective, 5G is also characterized by advanced hardware virtualization, specifically based on Software Defined Networking (SDN) and Network Function Virtualization (NFV). These novel approaches allow commodity hardware to virtually act as dedicated hardware for a specific network function, such as firewall, routing, or switching. In addition to traditional virtual machines, lightweight virtualization has an important role in achieving certain levels of performance, especially when there are some limitations or constraints on the available computing resources. Docker containers, along with the corresponding orchestration software Kubernetes, have been exploited in many 5G network deployment scenarios [5] [6], as they guarantee an efficient level of availability and maintainability within a small amount of resources.

Even though high density deployments may be costly for MNOs [7], these approaches, at both low and high levels of hardware abstraction, make infrastructure deployment and maintenance easier in comparison with the past generations. Consequently, in urban and suburban environments, the deployment of higher density architecture components is becoming a trend among MNOs [8]. In this way, it is possible to increase the availability of the services and decrease, at the same time, the overall latency, especially because the radio access components and the services themselves are very near to the end-user. Speaking about the short distance between the actual service and the end-user equipment, there are several technology trends focused on this scope. One of the most relevant trend in mobile networking is the 5G small cell. Although this class of radio access device is not a novelty in the mobile networking scenario, its usage in the 5G context is different compared to the past generations. Applications that require a very low latency, like Vehicle ad Hoc Networks (VANET), smart cities services and other IoT use cases, which are usually characterized by a large amount of connected device, can be easily enabled by the deployment and configuration of a smart grid of small cells. Furthermore, by deploying Multi-Access Edge Computing (MEC) units in correspondence to a small cell, it is possible to enable local computing ser-

vices that do not require a high amount of computing resources. Examples of applications that can exploit this typology of configuration can be ultra-high definition multimedia content delivery, augmented reality, or virtual reality. Obviously, despite all the advantages in adopting small cells in local service deployments, many challenges have to be addressed in order to guarantee the required QoS. As multiple operators can deploy their own services within a single small cell, it is crucial to understand if there is an issue with letting multiple tenants deploy their services within the same radio access or edge server unit.

1.1 Problem statement

As discussed previously, lightweight virtualization has an important role in many 5G use cases and within the corresponding architecture. Specifically, many operators deploy their containerised services by exploiting the Kubernetes Orchestrator capabilities [9] [10], as it is one of the main choices in terms of container orchestration software. In addition to this, an increasing trend in the sharing of physical resources between operators has characterised the latest local deployment configurations in urban and sub-urban contexts [11]. This implies that operators must be isolated from each other in a rigorous way, especially when they share common environments, such as a single Kubernetes cluster. Consequently, assigned resources, such as storage, memory, computing, and networks, should be carefully allocated and isolated to avoid legal or policy issues.

On the other hand, in the last years, the research community has found concerning issues in balancing hard isolation between tenants, and guaranteeing at the same time a satisfying level of performance. Most of the current adopted security approaches, which mainly aim to isolate tenants that share the same physical resources, produce an overhead, which is difficult to ignore in the context of ultra low latency applications [12] [13].

For these reasons, the main scope of this thesis, and so the main research question, is understanding what impacts the main available kubernetes hard multi-tenancy isolation approaches have on the network performance of the deployed applications.

1.2 Testing and evaluation activity

In order to give a broad overview, test activity was based on deploying three open-source applications, one for each 5G use case category (URLLC,

mMTC, eMBB). The deployment was performed by exploiting the identified Kubernetes hard multi-tenancy approaches. Thus, the three applications were tested after being deployed with each of the analysed approach. For each application, reference network metrics that are strictly related to the application class itself and the 5G requirements were identified. Data were collected using benchmark and testing tools, specifically designed for each deployed service. The evaluation was then performed by analysing the extracted data with the support of graphical visualisation tools. During the evaluation phase, it was possible to compare the analysed approaches and understand their corresponding behaviour in specific scenarios. In conclusion, from the available data, it is possible to observe that the network performance still strictly depends on the characteristics of the application itself. Specifically, tested multi-tenancy approaches impacted network performance depending on the adopted application network protocols.

1.3 Related works

Despite some concerns in considering the "Multi-tenancy" [14] concept still relevant in the cloud computing scenario, given the secure nature of the server-less computing applications, the research community has still further developed some approaches and solutions that enables multi-tenancy within a Kubernetes environment.

KubeSphere [15] is a project that aims to improve the orchestration of Kubernetes cluster resources when multiple tenants deploy their services on the same physical infrastructure. This is achieved by scheduling resource assignment and adopting a multi-resource fairness policy meta-scheduler. The additional scheduling layer, based on Dominant Resource Fairness (DRF) improves fairness between different tenants that deploy very different set of tasks.

EdgeNet [16], on the other hand, is an open-source project that is focused on extending the management model of a Kubernetes cluster. Additional functionalities like "location-based node selection", "node contribution" and, of course, "multi-tenancy" make the deployment of tenant services easier than adopting the vanilla Kubernetes management model, especially within an edge location. This enables the capability of providing a distributed public Kubernetes cluster where nodes are deployed in different locations around the world. It is important to note that an open Kubernetes Working Group for Multi-Tenancy was established at the end of 2017 (`https://groups.google.com/g/kubernetes-wg-multitenancy`). During these years, they had regular meetings and developed multiple approaches

for enabling multi-tenancy functionalities in Kubernetes.

In the mobile networking context, multi-tenancy has often been discussed along the concept of "network slicing" [17] [18], which is one of the key enablers of 5G networks. Each slice is allocated to a specific tenant, which mainly provides network isolation between other tenants. In addition to this, it also provides further customisation of the dedicated resources, depending on the requirements of the slice owner.

1.4 Structure of the Thesis

In chapter 2 an overview of the 5G architecture will be presented, along with the most common trends and use cases from which the 5G requirements have been defined. Furthermore, the Micro-Operators business model will be presented as one of the main reasons behind the multi-tenancy in 5G. Finally, 5G-Essence project will be described, as it exploits the previously presented business model. Chapter 3 will cover virtualisation and containerisation and, at the same time, highlight their crucial role in mobile networking. Technologies like Docker, LXC and Kubernetes will be discussed in order to provide an overview of what technologies are being employed in this thesis. During chapter 4, the multi-tenancy concept in the kubernetes context will be described along with the three main approaches that are currently available: Hierarchical Namespaces Controller (HNC), Multi-cluster deployment and Virtual Clusters. Furthermore, in chapter 5, the previously presented approaches will be described in detail: what applications are going to be adopted for testing purposes; what are the testing tools and methods, and the results of the testing activity. Finally, in Chapter 6, based on the testing session and the corresponding results, a final evaluation will be provided, which will suggest the best approach within the available ones. Possible future work and opportunities will be presented in order to provide some guidelines on how to further adopt this evaluation approach for future solutions.

Chapter 2

5G trends and use cases

Since 1998, 3GPP has been involved in the development of universal protocols for mobile telecommunications. Their development platform is based on a system of parallel "Releases" which enables developers to work on different projects at the same time. In 2019, 3GPP has published the final version of release 15 [19]. The development of this release started in 2017 and, during the 2 years of development, the first 5G system design standard version was defined.

Compared to the previous generation, 4G, the current state-of-the-art mobile network standard provides crucial improvements in terms of architecture, performance, and new enabled applications [20]. Thus, in this chapter a general overview of 5G architecture and use cases will be provided. A more detailed analysis will be performed on the micro-operator business model and the corresponding technical solution, 5G-Essence.

2.1 5G Architecture and requirements

While designing, configuring, and deploying a network, the concept of a "plane" can be used to classify components and data fluxes depending on their role [21]. For example, in mobile networking, the concepts of control plane and user plane have been widely adopted since the first release of the 3G standard, where, for the first time, packet-switched traffic was supported. The control plane refers to all the entities involved in handling the signaling between the network and the user equipment for management purposes; meanwhile, the entities in the user plane are involved in delivering the actual user payload. In Figure 2.1, it is possible to observe that for the first time in mobile networks there is a clear separation between the elements involved in the control plane and those involved in the user plane. This approach, com-

monly named "Control and User Plane Separation (CUPS)", lets the network infrastructure operators centralise the control plane in a single location and the user plane to be deployed in a distributed approach, usually closer to the actual user equipment. This design strategy enables a higher degree of elasticity and maintainability, which is one of the reasons why it is possible to achieve a higher level of performance compared to previous generations, especially in terms of availability [22].

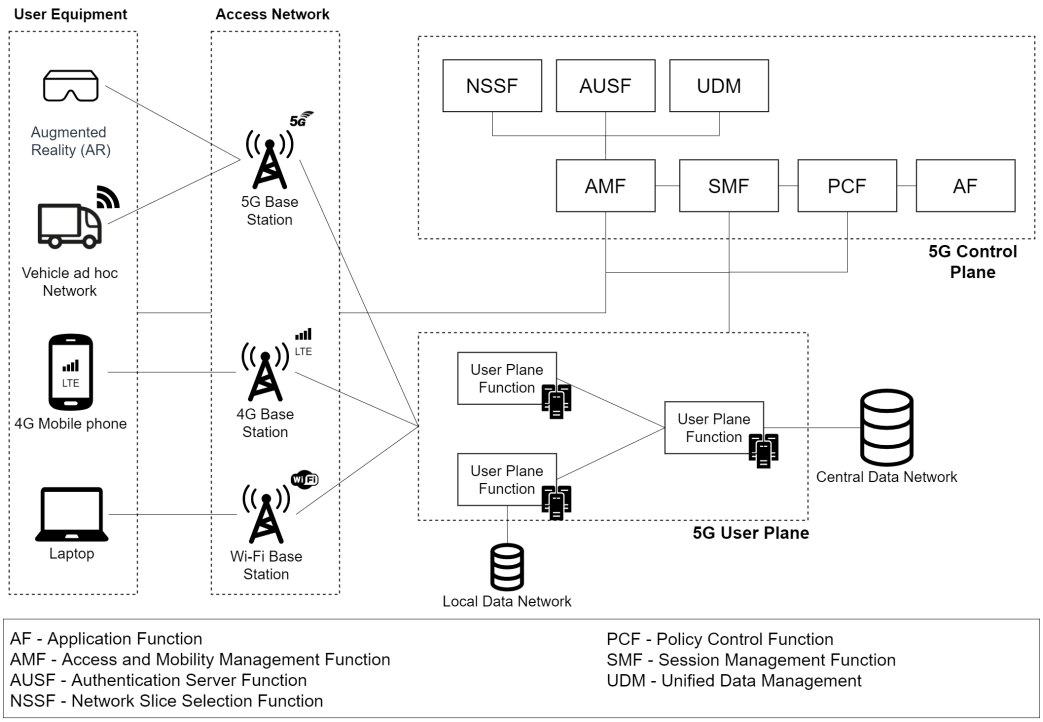


Figure 2.1: 5G architecture diagram.

As soon as new use cases, applications or trends emerge from the market, current adopted standards often show some limits. This motivates researchers and engineers to look for new approaches or technologies that can meet new needs. Mobile networking evolution has been characterized by this process, where the former state-of-the-art standard was no longer capable of satisfying mass market needs. Furthermore, in the specific case of 5G, the requirements are mainly based on recent years trends. As shown in 2.2, there are three main categories of applications, which have different degrees of requirements [23] [24] :

- **Enhanced mobile broadband (eMBB)**: this class of applications

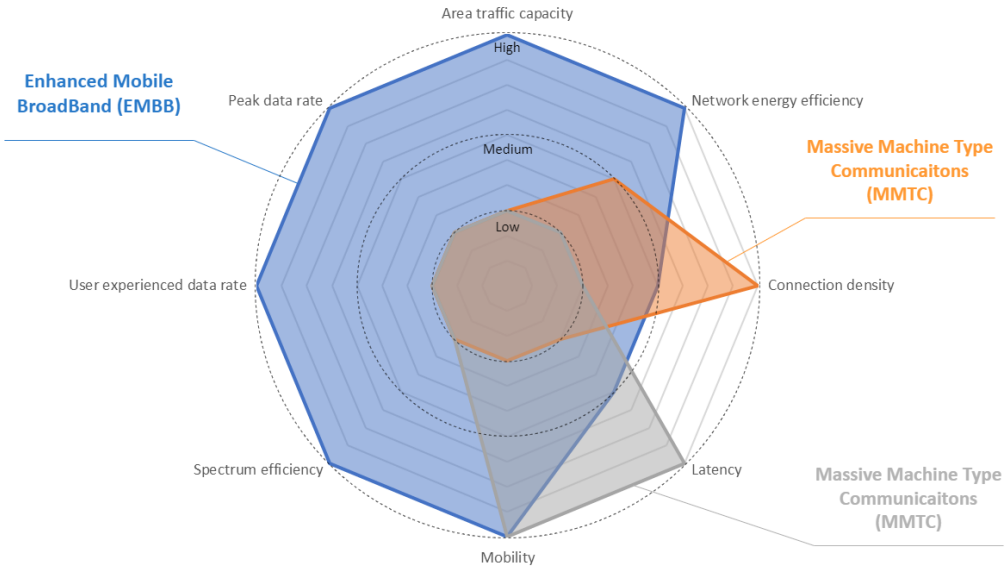


Figure 2.2: Spider plot of each 5G use case category [23].

is often identified as the evolution of existing 4G networks. Consequently, researchers are mainly focused on increasing the quality of human-centric services such as media content delivery and mobile communication applications. Thus, a very high amount of throughput must be guaranteed for a consistent number of devices at a certain location. Examples that can be classified as eMBB applications include 360° 4K video streaming, real-time video monitoring, or augmented reality (AR).

- **Massive machine-type communications (mMTC)**: in the last years, the Internet of Things (IoT) has become a strong trend in telecommunications [25]. The exponential increase in connected devices has driven researchers and engineers to adapt network capabilities for this purpose. Hence, 5G has better capabilities in managing a large amount of connected devices in a small area. At the same time, since most of these connected devices have small energy resources or are directly self-powered, 5G Radio Access components must guarantee an optimal level of power efficiency. This class of application is mainly focused on machine-to-machine communication (M2M).

- **Ultra-reliable low latency communications (URLLC)**: as it is possible to see from the name of this family of applications, latency

is defined as a very critical key metric. In recent years, many applications that involve a responsive interaction between machines or between machines and humans have been introduced into the mass market. Furthermore, the stability of the connection between entities of the network must also be guaranteed if a device is in motion, even at high speeds. This implies that an optimal deployment of radio access network antennas and efficient handover approaches must be adopted in order to guarantee the initial requirements. This specific configuration of metrics enables novel use cases, such as automated driving, remote traffic management, industrial automation, remote healthcare, and highly interactive VR or AR [26].

Table 2.1: Comparison between 4G and 5G requirements [27].

Key performance metric	**4G**	**5G**
Maximum data rate	1 Gbps	20 Gbps
User data rate	10 Mbps	100-1000 Mbps
Maximum mobility speed	350 km/h	500 km/h
Latency	10 ms	1 ms
Connected device capacity	100000 devices/km^2	1000000 devices/km^2
Data processing capacity	0.1 Mbps/m^2	10 Mbps/m^2
Power efficiency	1x	8100x

Based on these three main classes of scenarios, key performance reference metrics were defined, as shown in Table 2.1.1. Similarly to previous generations, 5G capabilities are on average 10 times higher than those available with a 4G system. At this point of the analysis, it would be interesting to briefly understand which enabling technologies have been exploited in order to reach these levels of performance.

In the research community literature, there are many articles and works available where a survey is performed on key enabling technologies for 5G. For instance, a very detailed survey from 2019 [28], describes relevant technologies in the 5G deployment context. One of the most critical sets of components for a 5G system is the access technology used. A new spectrum allocation has been allocated for 5G access networks, which is split in two ranges [29]:

- **Frequency Range 1 (FR1)**: this subset of frequencies include sub-6 GHz bands, with which it is possible to support previous standards, with an extension towards the range between 410 MHz to 7125 MHz.
- **Frequency Range 1 (FR2)**: frequency range from 24.25 to 52.6 GHz.

Consequently, it is possible to serve different communication systems within the same infrastructure that has such a wide range of frequencies. This approach is called multi-radio access technologies (multi-RAT). These frequencies are managed by deploying efficient antennas that rely on novel approaches such as massive multiple input multiple output (M-MIMO) or, in the case of a scenario with a large amount of connected user equipment, multi-user multiple input multiple output (MU-MIMO), enabled by the spatial division multiple access (SDMA). In addition to the actual physical specifications of the radio access technology, the adopted strategy for deploying a dense network of access components was also carefully analysed by the research community. Thus, current trends are pointing towards an ultra-dense network (UDN) where access points are optimally deployed in order to achieve the best use of the available spectre in a certain area. Furthermore, to simplify the management of a large number of radio access components, MNOs often adopt the Cloud-Based Radio Access Network (C-RAN) approach [30]. Basically, the management and the allocation of the radio resources is centralized in a cloud solution. This allows to have a higher degree of scalability and densification, which drastically decreases the operational costs of the infrastructure [22].

Speaking of cloud solutions, if the 5G architecture is analysed from a higher-level perspective, it is possible to notice that novel network and software technologies are adopted. Approaches like Network Function Virtualization (NFV) and Software Defined Networking (SDN), which are based on virtualisation paradigms, are important technologies in the 5G architecture. If in previous generations, a specific proprietary hardware was necessary to enable a particular functionality, with NFV, generic commodity hardware virtually enables the same functionality. Consequently, the physical components of the architecture can be reprogrammed very easily, depending on the required functionality. SDN, also based on virtualisation paradigms, enables a higher degree of elasticity in network configuration, since routing and switching are now performed by virtual components that can be remotely configured and deployed [22]. Examples of these approaches include Network Slicing, Self-Organising Networks (SON), Device-to-Device Communications (D2D), Fog Computing, and Multi-access Edge Computing (MEC). This last example will be discussed in detail afterwards, along with a very interesting business model, based on a novel radio access approach.

2.2 Micro-Operators business model

In the past mobile network generations, only a small group of MNOs was involved in delivering communication services to the mass market [31]. Therefore, most of the infrastructure was owned and managed by powerful companies, which had a business model based on large infrastructure management and long-term investments. In recent years, however, an evident shift towards a different model has been observed. In addition to traditional voice and text communication services, a wide range of heterogeneous services have been deployed to the public market. Furthermore, as different vertical classes of application are characterized by different requirements, also the infrastructure must be deployed and configured accordingly. As a matter of fact, these new trends are no longer compatible with the traditional Telco business model.

In the late stage of the development of the 5G standard (Q3 2017), the University of Oulu, in collaboration with Nokia Oulu, published an important study on a new telecommunication business model [32]. Based on this novel approach, the telco market is no longer based only on a small group of stakeholders but on a wide range of actors that are involved in very different roles. On the other hand, if this could increase the complexity of infrastructure management, it would enable an elastic market of different vertical sectors that could be deployed and configured within a local area in a very short time. However, as explained in [32], a change of this magnitude requires the adoption of different strategies in different domains: regulations, technologies and business itself. However, this change to a new model may be a complex task to achieve in a short period of time. Despite the possible challenges of such operations, a new concept has been identified within the mobile business ecosystem: micro-operator (μO).

This new actor is considered the key enabler of this new business model, as he is in charge of deploying scalable new applications within a local area, which is often an indoor scenario. In this case, connectivity and service delivery is guaranteed by small cells, which are configured specifically based on deployed application requirements. As explained above, there are many challenges in switching from the traditional business model to a novel one. Specifically, as now there are many vertical domains, each of them managed by a specific set of micro-operators, it is necessary to adopt a clear strategy in allocating spectrum licences and shared resources. In addition to this, μO must agree on common security and data processing policies to avoid future legal problems, especially if some resources are shared. The cost and the complexity of this typology of configuration can be easily overcome by

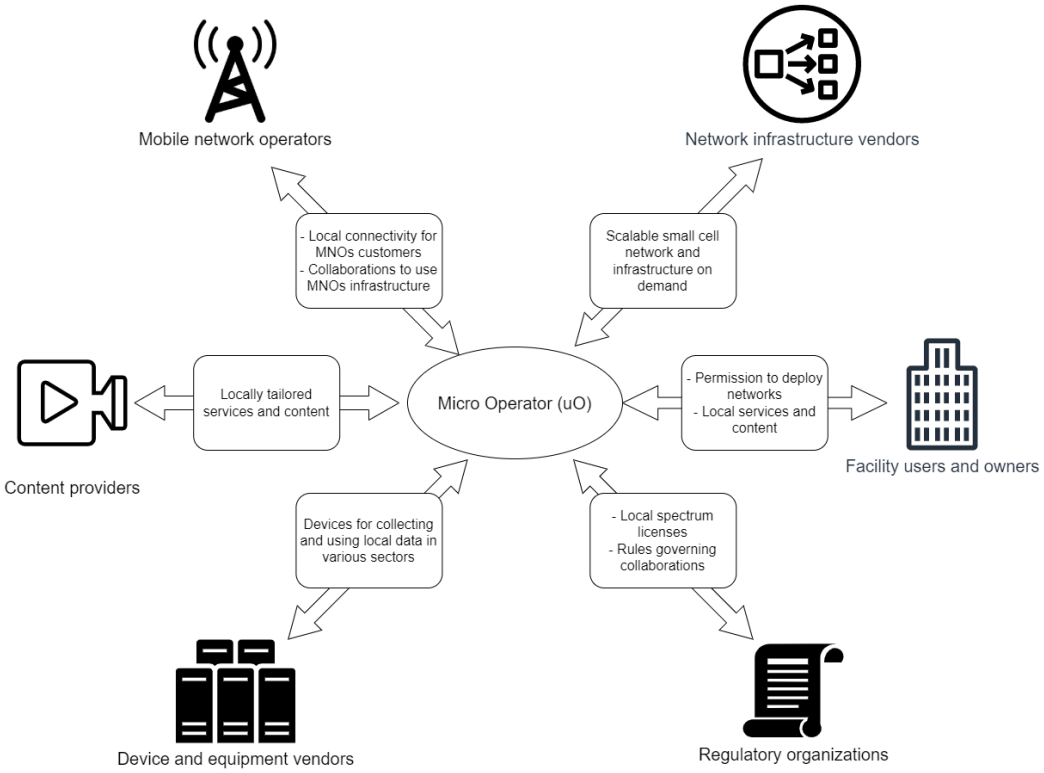


Figure 2.3: Micro operator's relations with key stakeholders [32].

adopting specific virtualization approaches, such as NFV and SDN. Furthermore, based on the key reference pillars of 5G, elasticity and maintainability are still guaranteed, as these resources can be allocated and configured on demand. In Figure 2.3 it is possible to observe which are the main stakeholders that interact with a micro operator. Each of them provides specific support or resource to the micro operator such that it is capable of successfully deploying the local service. In a more recent work, also from the University of Oulu, an additional analysis was performed on the role of end-to-end network slice architecture in the context previously described [33]. The authors of this article have identified different deployment scenarios based on the final use of the μO premises:

- **Closed micro-operator network**: it is essentially a private network where services are delivered only to a recognised user group. It is usually enabled by using highly secure approaches that can guarantee the isolation of the processed data.

- **Open micro-operator network**: in the opposite to the previous

case, here micro operators are in charge of delivering services to any MNO subscriber that is located in a specific local area. This class of configuration is usually adopted when MNOs have no interest in deploying applications within a specific area and so the task is assigned to a specialized μO.

- **Mixed micro-operator network**: both of the cases described above are deployed within a set of allocated resources. If the μO is in charge of deploying a private service for a vertical set of users and, at the same, time manage MNO subscribers within the same area, an high level of isolation must be guaranteed.

What is in common between all of these configurations is that they all rely on the concept of network slicing. Thanks to SDN, it is possible to define specific routing configurations that isolate a subset of devices from the rest of the physical network. Each of these isolated networks is commonly named slices, as they can also be configured to behave in a specific way in terms of performance, such as delay, throughput, and jitter. Furthermore, it is also relevant to highlight that deploying services in a local area, and so in a location very near to the end users, was also exploited in order to optimize computing resources and capabilities, thanks to MEC concept. In the next section, a brief analysis of this network architecture paradigm will be performed, as it is relevant in the context of this novel business model.

2.3 Cloud, Multi-access Edge and Fog Computing

In the last decade, significant progress has been made in the design approaches of the infrastructure. In the early stages of the Internet, most service providers had to deploy and configure their own on-premise infrastructure, which was mainly made with cheap and commodity hardware. The successive exponential increase in requirements, in terms of performance and reliability, pushed into the market a concept that was originally defined by Professor John McCarthy in 1961: utility computing. Basically, computational resources could be identified as a public utility, similar to the traditional telephone or electrical system. Due to the exponential growth of hardware capabilities and novel virtualisation approaches, which will be discussed in Chapter 3, this pioneering idea was actually implemented around fifty years later under the label of "Cloud Computing". In one of the early works on the definition of cloud requirements [34], cloud computing was defined as a service delivery and access approach in which scalable and virtualised resources

are dynamically delivered as a service. Consequently, today most businesses are no more concerned about having their own on-premise IT infrastructure to host their services, as they can rely on third-party infrastructure providers [35].

From the mobile network sector perspective, during the early stage of 5G development, European Telecommunications Standards Institute (ETSI) identified one of their network architecture paradigms, Mobile Edge Computing, as one of the key enabling technologies for 5G [36]. In an early survey on this topic [37], Mobile Edge Computing, now known as Multi-access Edge Computing, was defined as a network architecture paradigm that is capable of enabling cloud computing services focused on business within the radio access network. The motivations that continue to drive the research community and engineers to invest in this approach are related to a wide range of applications that rely on the capabilities of MEC. As explained in a more recent survey [38], the main capabilities of this approach are mainly based on the positioning of the MEC nodes: a lower level of latency is successfully achieved by deploying these nodes in the vicinity of end users. In addition to this, further optimisation of the QoS is achieved by collecting information from the access environment: real-time radio statistics are collected in order to tune the network behaviour accordingly. The ability to be aware of the environment makes each MEC node autonomous in terms of local resource management, so it can operate perfectly even if the node is isolated from the rest of the network. Finally, even though some computational tasks are performed directly within MEC node, the heaviest one are forwarded to a centralized node, which is based on a cloud data centre. Many use cases that today have a strong presence in some industrial sectors are enabled by the MEC paradigm [39] [40]. Examples of these application classes are AR/VR, VANET, live security and control, location tracking, and IoT solutions. Specifically, all the applications in the IoT environment have really demanding requirements in terms of latency and energy efficiency. Thus, MEC is essential for the deployment of this typology of applications, as, in addition to the low latency capabilities, computational offloading and content awareness are also guaranteed in most of the cases. These two last capabilities are crucial, as most of the IoT devices, such as outdoor sensors, rely on a very limited amount of power, and consequently it is necessary to centralise most of the computational task into the cloud and optimise radio access communication approach [41].

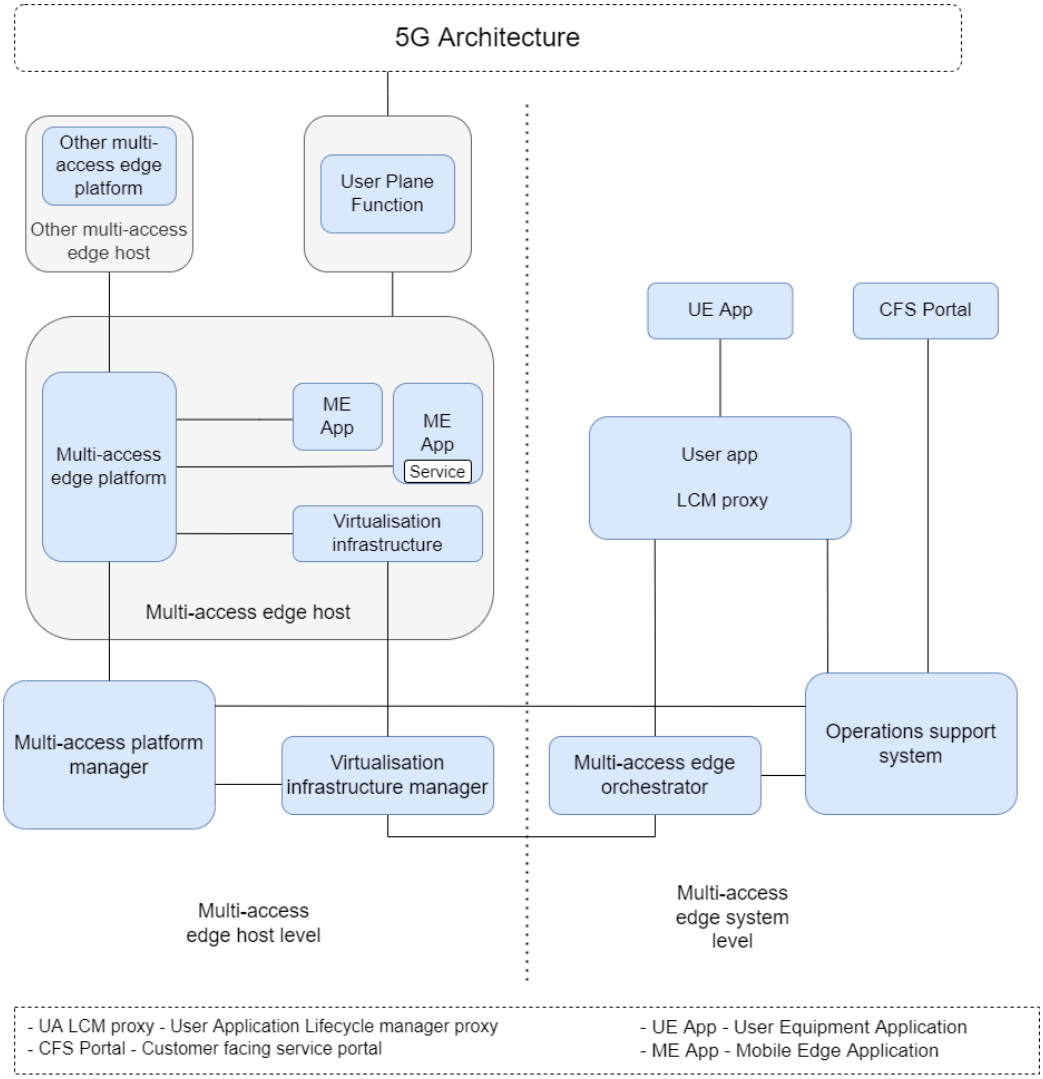


Figure 2.4: ETSI MEC reference architecture [42] [38].

In Figure 2.4, a high-level diagram of the ETSI MEC architecture is represented. As most of the actual architecture entities are virtualised, it is possible to observe that there is a clear separation between the system level and the host level within the MEC system. Available physical resources are managed by the Multi-access edge orchestrator, which is also aware of the available services and the current status of the node topology. It is defined

as the core of the MEC system as it is in charge of most of the operational tasks such as application relocation, deployment, and termination. The MEC platform manager is responsible for managing the lifecycle of a set of MEC applications by controlling network policies and application rules. In addition to this, this component collects any problems and performance measurements. Finally, the virtualization infrastructure is in charge of managing virtual resources and the required images for deploying MEC applications.

While MEC is based on the definition of Edge computing, in recent years, an extension of this paradigm was defined, Fog Computing. A group of Cisco engineers published an analysis on the role of fog computing in IoT scenarios [43]. Within this work, Fog Computing is defined as a virtual provider of computational, storage, and networking resources deployed between edge locations and centralised cloud locations. It can be seen as an additional layer that provides further cooperation and optimisation between different edge locations and consequently enables further opportunities and use cases. Despite all the possible advantages that Fog computing can offer, how can this be interfaced with MEC systems? An attempt was made in [44]. Here, the authors tried to associate for the first time the term Mobile Fog Computing (MFC) with all mobile IoT applications that exploit the fog layer of a complex system. Specifically, it is explained why many IoT applications require intermediary nodes between edge locations and central cloud ones. For instance, in the case of VANET or UAV networks, having a dynamic system that offloads the computing tasks not only from edge nodes to centralised cloud nodes, but also towards fog nodes dramatically increases the reactiveness of the system. Despite being a very promising approach for many IoT use cases, there are many challenges that still limit the success of it. Speaking about connectivity of moving entities, mobility is still a concerning issue as it is still complex to develop an efficient autonomous management system that coordinates all the fog and edge clouds when they have to react to a very fast moving equipment.

2.4 5G-Essence Project

Since 5G networks are characterized by the ultra-dense deployment of radio components, it is interesting to analyze in which way they could be designed and optimized within specific scenarios. Even though this mobile network generation relies on a heterogeneous set of radio access antennas, small cells base stations (SBS) have a crucial role. Depending on the use case and the corresponding application, different sizes, such as picocells, femtocells, and ultradense small cells, are deployed to cover areas between 100s and 10s

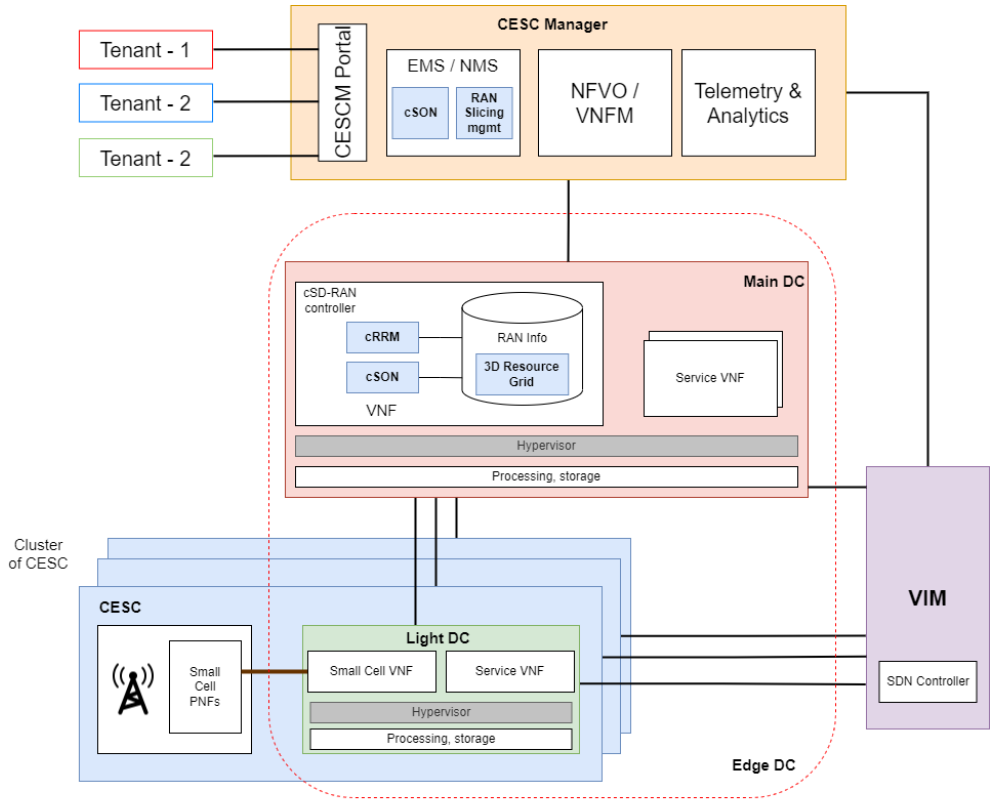
metres [45]. The challenges that characterised the design and the following configuration of the SBS are quite different compared to those related to the macrocell base stations (MBS). Novel approaches and paradigms have been applied in the design of this class of antennas, especially in terms of their physical characteristics. Signal management strategy is clearly based on the behaviour of the target user class. Mobility and coverage are two of the most critical metrics that have been studied in order to develop an efficient signal transmission approach [46] [47]. In addition to these parameters, researchers and engineers are also focused on providing an energy efficient solution that can still guarantee to satisfy 5G requirements even with low levels of power consumption.

Nevertheless, physical design is not the only aspect that can be exploited to provide an additional level of optimisation for RAN components. As seen in the previous sections, edge computing has a very important role in providing very low latency transmission within the user location. In addition to MEC, virtualization techniques have been extensively exploited within 4G and 5G architectures, and RAN components are no exception to this trend. Cloud-enabled small cells (CESCs) , also defined as Small-Cell-as-a-Service (SCaaS), are a paradigm that enables the deployment of VNFs, applications, and services within the access network [48]. Consequently, in order to support the orchestration and the deployment of virtual resources, it is necessary to setup an efficient architecture that can support and enable these features. The 5G Infrastructure Public Private Partnership (5GPPP) has defined a framework commonly named the 5G-Essence project, which is based on the former SESAME project [49]. As explained by the founders of this project, the proposed architecture is based on two relevant elements of the 5G environment: MEC and Small Cells. The reason behind the definition of this solution is based on the fact that many scenarios, such as the micro-operator one, require a flexible and scalable solution which is capable of satisfying new business models. The key function is to provide a platform where multiple network operators have the ability to provide services to their corresponding users by exploiting a set of CESCs. The infrastructure that hosts the small cells is owned and managed by a third party. It is obvious that, politically, this architecture is based on a hierarchical multi-tenancy model, where multiple tenants rely on another tenant.

2.4.1 Architecture description

As the 5G-Essence project is based on the edge computing paradigm, the corresponding architecture is organized in a two-tier cloud system. In order to provide a satisfying level of elasticity and performance, which are neces-

sary to enable a 5G deployment, most of the resources are virtualised based on the most common approaches, such as MEC and network slicing. Even though virtualisation has been widely adopted within 4G and 5G standard infrastructures, the 5G-Essence project exploits the same concepts in order to increase the performance of small cells within a multi-operator scenario. As shown in Figure.2.5, the unique element that is introduced by the developers of this project is the CESC, which is a computational node deployed in a edge location, that provides storage, computing and radio resources within a set of small cells. Multi-tenancy can be enabled very easily by deploying a cluster of CESCs, where each of them can be managed autonomously by a specific local operator. The two-tier architecture is organized as follows:



DC - Data Center	CESC - Cloud Enabled Small Cell
VNF - Virtual Network Function	CESCM - CESC Manager
PNF - Physical Network Function	NFVO - VNF Orchestrator
VIM - Virtualised Infrastructure Manager	VIM - Virtualised Infrastructure Manager
EMS - Element Management System	NMS - Network Management Station

Figure 2.5: 5G-Essence reference high level architecture diagram [49].

- **First tier**: it is the tier located in the edge location which directly enables the CESC functionalities. It is based on a **Light Data Center**, which is usually hosted within the CESCs cluster. It is responsible for enabling basic packet traffic management functionalities and must be able to execute VNFs that are involved in small cell access virtualisation. Furthermore, this tier can also enable any VNFs that require a low amount of computational resources, such as the Machine-to-Machine (M2M) Gateway, which is the core of IoT systems.

- **Second tier**: like in most of the edge systems, edge nodes are supported by a centralized node, which is usually deployed in a further location and it has a larger amount of resources available. Given the nature of the deployment location and the capabilities, it is usually referred as the **Main Data Center**. All tasks that require a substantial amount of resources are executed within this tier. From the perspective of this component, different edge locations can be managed from this location, allowing a global view of the infrastructure.

In addition to these tiers, another crucial component is the CESC Manager. Here, multiple management functionalities are enabled, such as the network virtual function orchestration (NFVO), the Entity Management System (EMS), functions related to the collected metrics from the system, and a portal where tenants can access their allocated resources via API.

2.4.2 Example of service deployment in a Cloud-Enabled Small Cell

Nowadays, edge computing is widely adopted and continuously improved, as many applications rely on the capabilities of this paradigm. However, CESC extends edge capabilities by providing further radio functionalities. In [50], the authors explain how 5G-Essence can be adopted for a very critical class of applications: mission-critical public safety services. Currently, emergency networks are so crucial to society that they do not rely on commercial mobile infrastructure but are deployed as a parallel and isolated infrastructure. This is related to the fact that this class of communication channel requires a very high level of security, but at the same time it must be reliable. Given the nature of the corresponding tasks, low latency must be guaranteed in order to enable a responsive and effective communication channel. Furthermore, mission-critical service users do not have a fixed position since they are usually adopting emergency vehicles. For this reason, it is necessary to increase the overall mobility of the allocated resources within the involved

radio access components, which means that the handover phase from one cell to another is something critical to be managed. Among the mission-critical applications class, one of the most demanding and latency-critical use cases is the Mission Critical Push-To-Talk service (MCPTT). Currently, the available solutions are not very flexible in terms of capacity tuning, missing support of multimedia content delivery in case video or audio is required by the application, or most importantly, there are no possibilities to have a common infrastructure for both mission critical and public traffic. Having a network with the characteristics described above would dramatically improve the quality and responsiveness of the emergency service. In recent years, as previously described, 5G has been a game changer in terms of latency and network capacity. Since MEC is the key enabler of latency-critical applications, the authors of [50] were interested to see if 5G-Essence could be the right solution for the emergency use cases. Since 5G-Essence provides a centralised orchestration of radio resources, it is possible to dynamically allocate resources to a specific area where an emergency occurs. In this case, different tenants from different industries share the same infrastructure thanks to network slicing. However, any tenant that is involved in emergency services has a higher priority access in terms of resources in comparison to other tenants. This enables what was not possible with earlier solutions: providing an elastic approach with which it is possible to exploit public infrastructure deployment while all the mission critical requirements are guaranteed. The validation of this setup was carried out in a real scenario, where public safety services were tested at different levels of emergency, which are triggered by a monitoring system. As expected, depending on the emergency level, access capacity can be increased in order to avoid any issue if more emergency units are focused on a specific area. Furthermore, if the emergency involves also the integrity of the infrastructure, 5G-Essence architecture is capable of extending the coverage of a specific area under exceptional circumstances. Validation was rigorously evaluated, as not only were the basic functionalities of the infrastructure tested, but extended performance evaluation and end-user experience tests were performed.

2.5 Summary

5G environment includes a wide set of novel technologies, which are exploited in order to enable new applications and business models. It is clear that 5G is a step forward compared to previous generations of mobile networks. The gradual passage of TelCo from traditional business models to novel ones, like the micro-operators model, is a clear sign that requirements of the mass

market have drastically changed. Consequently, research community and engineers have to adapt their approaches accordingly. Since virtualisation plays an important role within the 5G environment, a detailed description of its evolution will be provided in the next chapter.

Chapter 3

Virtualisation and containerisation

In the early stages of the World Wide Web (WWW), the "one application per server" rule was the most common approach adopted by service providers. Having one single dedicated server for a specific application guarantees an optimal level of maintainability, as system parameters and configuration depend only on the requirements of a single application [51]. Furthermore, it is clearly difficult that other applications, which are installed in other servers, could actually affect the performances and the security of the service. Despite the advantages that this approach offers, a huge limit was identified while the WWW was changing the world: it was not possible to satisfy the exponential growth of the deployed services with this approach as it was not feasible to have a corresponding number of dedicated servers. At the same time, on the basis of the Moore law, servers were starting to be developed with a higher computational capability. The final result was a large amount of servers, which were not exploited at their maximum capabilities and that, at the same time, were consuming a large amount of electrical power. Another obstacle that was forcing infrastructure architects to keep adopting this approach was the lack of virtualization functionalities available in the OSes: they were not capable of providing the required tools in order to guarantee an efficient level of isolation and reliability between applications deployed on the same machine. Consequently, it was necessary to adopt another different resource allocation strategy, which led engineers to enforce an approach in their infrastructure design approaches: virtualisation. In the next sections of this chapter, a detailed description of how the concept of virtualisation was already defined during the 1960s and how it has evolved during the last 30 years will be provided. Starting from the definition of the concept itself to the state-of-the-art concepts of container and cluster of containers, the most

significant milestones of this evolution will be covered.

3.1 Evolution of virtualisation

The concept of virtualisation was defined for the first time in the 1960s by IBM [52], followed by the formal definition published by Goldberg and Popek during the 1970s [53]. In this specification report, which is still the reference point in modern virtualisation approaches, the authors provide a clear definition of virtual machine (VM) and the corresponding minimum requirements of the host system. A virtual environment can be identified as a VM, if it consists in an efficient and isolated copy of the real physical machine. This means that efficient control of the resources should be implemented. Consequently, in the host machine, a core component, which is often referred to as Virtual machine monitor (VMM), should provide resource allocation and isolation between different virtual environments. An efficient virtualization layer can be guaranteed only if the following characteristics are available:

- Any executed instruction in this environment should be seemingly run as if it was natively launched on the physical machine.
- A subset of the instructions executed by the VM processor must be executed directly by the physical processor whenever an higher grade of efficiency is requested.
- An efficient control of the physical resources must be available to the VMM. Resources allocated to a specific environment should not overlap with those allocated to another virtual environment.

Despite having a clear definition of how virtual hosts should be implemented, in those years, the mass market needs and the substantial decline of hardware production costs were limiting the actual exploitation of virtualisation. Consequently, only in the late 1990s the virtualisation concept was finally adopted in order to optimise the performances and the usage of the available data centre resources. In this way, the previously described limits were finally overcome. In the last 20 years virtualization concept was implemented in different ways, depending on the requirements of the specific use case.

Therefore, two main classes of virtualisation approaches are often identified in the literature [54] [55]:

- **Hardware-level virtualisation**: this class is strictly based on the definition of VM provided by Goldberg and Popek in [53]. Consequently,

in the approaches that can be identified within this class, the virtualisation relies on an additional layer that is responsible for providing multiple virtual clones of the host hardware. For instance, in the case of one of the approaches adopted in this class, full virtualisation, the VMM is installed on the host OS as a generic application and simulates an exact copy of the underlying hardware to allow the guest OS to run completely isolated [56]. Any application designed for the host architecture can be executed on the guest virtual machine, as it can rely on the same instruction set of the physical processor. Despite the clear advantage of being capable of running any OS or application made for the virtualised architecture, the corresponding overhead may be too high in some specific scenarios. In addition to this, the design and implementation of the VMM architecture may be quite complex to develop and maintain [57]. Other commonly adopted approaches of the same class try to overcome the limitations of full virtualisation. For example, as soon as virtualisation became a mandatory requirement for data centres, hardware producers tried to decrease the complexity of the VMM architecture by developing new computational components with a set of features that could natively support the hosting of virtual environments. The two main processor producers, Intel and AMD, have released their own hardware support tools for virtualisation (Intel VT-x, AMD-V). Basically, within the instruction set of a CPU, an additional subset of instructions was implemented to increase the isolation between virtual environments, which in some cases may be crucial for overall performance. Furthermore, another commonly adopted approach, known as paravirtualisation, tries to solve the overhead issues of full virtualisation by providing support functionalities as part of a modified guest OS. In this case, the guest OS is aware that it is currently being executed in a virtual environment. Consequently, it can directly interact with the VMM by using an interface and perform "Hypercalls". Thus, the virtual environment can interact directly with the physical CPU instead of performing a binary translation of every instruction, as is usually done within the full virtualisation approach.

- **Operative System-level virtualisation**: despite other hardware-level virtualisation approaches trying to address overhead and complexity issues, traditional virtual machines are still not optimised for many use cases. Therefore, it was necessary to design a lighter solution that could still provide the virtualisation capabilities originally defined by the Goldberg and Popek model. Instead of replicating an entire system, with OS-level virtualisation, it is possible to create vir-

tual environments that directly depend on the host OS, as shown in Figure.3.1. This means that these virtual environments share the same kernel. Thus, from the point of view of the host OS, these environments are seen as processes with their own dedicated set of resources. This affects the performance of the system, as all the resources that in the traditional hypervised virtualisation are involved in the OS execution are now free to be allocated to the actual applications. In addition to this, any operational activity, such as the deployment or restart of a virtual environment, requires less time compared to a fully virtualised OS [58]. In addition to this, any operational activity, such as deploying or restarting a container, requires less time compared to a fully virtualised OS [58]. Based on these characteristics, it was possible to define the so-called "lightweight virtualisation", commonly known as an alternative to the traditional virtualisation described above [59].

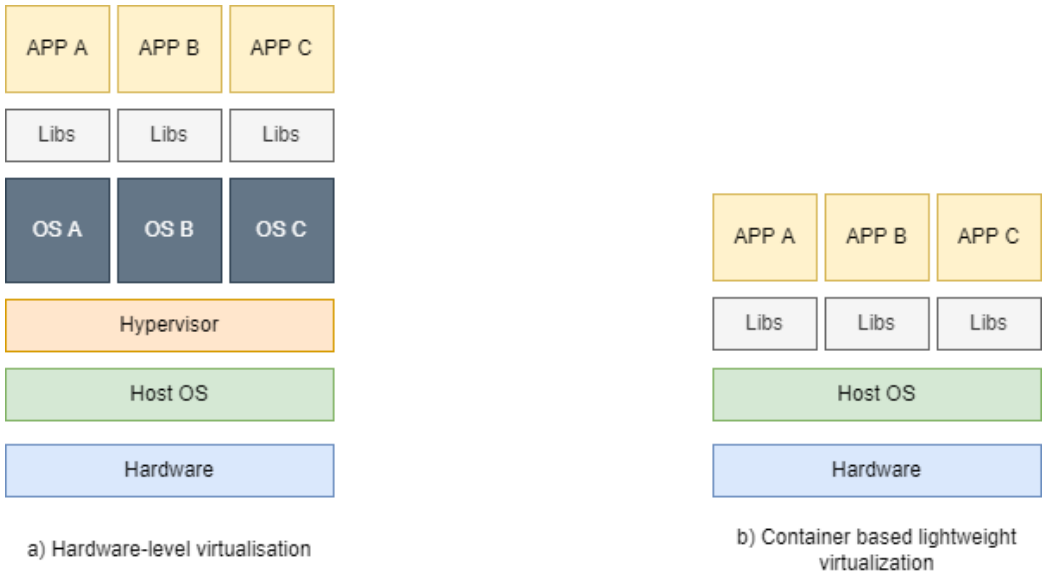


Figure 3.1: High level architecture comparison between hardware-level virtualisation and os-level virtualisation [60].

Although many different approaches are available, in the context of recent mobile networking infrastructures, where virtual environments are dynamically allocated at a very fast pace, lightweight virtualisation is preferred to traditional hardware-level virtualisation in some resource-constrained contexts [61]. For this reason, a more detailed analysis of lightweight virtualisation evolution and state-of-the-art is going to be performed in the next

section.

3.2 Lightweight virtualisation

The first concept design that can be classified as OS-level virtualisation, and thus as lightweight virtualisation, can be found for the first time in Unix V7, 1979. As stated in a recent blog article [62], Unix developers introduced the "chroot system call", a turning point in the evolution of lightweight virtualisation, as it was the starting point of process isolation. Basically, when this call is performed, the root directory of a running process and its corresponding children is moved to a new directory of the filesystem. The same feature was added to another OS, BSD, in 1982. Further steps were accomplished only at the beginning of the 2000s, when FreeBSD developers introduced the concept of "Jail" [63] in order to split hosting provider environments and customer environments for security and integrity reasons. This "clear-cut" separation was performed at the service level as the OS was partitioned into smaller independent systems, which included all the necessary networking functions, such as assigning an IP address. In an article from 2006 [59], an early definition of "lightweight virtualisation" was provided, along with a forecast on the highly probable growth of the adoption of the approach in the next years. Despite being an old article, the concept of "Container" is already identified as one of the main actor in the lightweight virtualization scenario. It is defined as a group of processes that do not emulate any hardware component. During that year, however, the current modern solutions were not available, but one specific project was really similar to them: OpenVZ. Also known as Open Virtuozzo, developed by Virtuozzo and the corresponding project community, it consists in a OS-level virtualization software that enables the user to deploy multiple isolated containers by exploiting a modified version of the Linux Kernel [64]. Each container had its set of resources that included basic elements like memory limits, CPU quotas, filesystem or network configuration. Before presenting the current state-of-the-art of the container concept, it is relevant to describe two main features, namespaces and cgroups, which are available in the Linux Kernel. This is related to the fact that they are often described as the basis of available containerisation approaches [65].

3.2.1 Namespaces and CGroups

Since 1991, Linus Torvalds has been involved in the development of the Linux kernel. The project consists of a Unix-like monolithic kernel that

has been published under the GNU General Public Licence (GPL). Consequently, given the open-source nature of the project, in the last 30 years, many contributors have developed new features and expanded kernel compatibility to a wide set of architectures [66]. On the 3rd of August 2002, the 2.4.19 release of the Linux Kernel was released. Within the new features and fixes of this version, a new functionality was included: namespaces. As described in the Kerrisk's Linux manual [67], a namespace consists of abstracting a set of kernel resources in order to make them available only to a restricted group of processes. Therefore, any change applied to that specific set of resources can only be seen by members of the specific namespace. Any other process that is allocated outside that specific namespace is unaware of the status of these resources. Depending on the target resource, there are different typologies of namespaces:

- **Cgroup**: a cgroup root directory of the target namespace members, which will be described later in this section, is not shared outside that specific namespace.
- **Interprocess communication (IPC)**: any IPC resource, such as POSIX message queues, is dedicated to all processes that are members of a specific namespace. Consequently, any other process cannot communicate with the namespace members within these resources.
- **Network**: a dedicated copy of the network stack kernel module is allocated only to the members of the namespace. Thus, IPv4 and IPv6 protocol stack, network devices, IP routing tables and firewall rules cannot be accessed by any member outside the namespace.
- **Mount**: released as the first ever namespace type in the Linux Kernel, it provides an abstracted isolation of the mount points. This means that filesystems are not shared between members of different namespaces.
- **Process ID (PID)**: a dedicated set of PIDs is assigned to the namespace. Therefore, two processes with different namespaces can actually have the same PID without causing any conflict.
- **Time**: independent system clocks can be assigned to a specific set of processes.
- **User**: user management kernel functionalities are abstracted from the overall OS. This means that it is possible to create a set of security rules and identifiers that can be only used within the specific namespace.

- **Unix Time Sharing (UTS)**: setting system identifiers, such as hostname and domainname, are not going to affect processes outside the namespace.

Originally called "process containers", Control groups (Cgroups) were designed and developed by two Google engineers, Paul Menage and Rohit Seth, in 2006 [68]. This functionality was then merged in the Linux Kernel version 2.6.24 release only 2 years later. CGroups provides a hierarchical approach to limit and monitor a subset of resources within a group of processes. Thus, each kernel component that manages a specific type of resource is responsible for allocating resource quotas to a specific cgroup.

Despite having cumbersome management and deployment approaches, these two kernel functionalities have been a game changer in virtualisation research, as they are the basis for lightweight virtualisation [65].

3.2.2 Linux Containers and Docker

In 2008, the first version of Linux Container Runtime (LXC) was released. LXC is one of the first mostly used OS-level virtualisation solutions that is capable of enabling the concept of container. It is based on the functionalities provided by the Linux Kernel that were described in the previous section: namespaces to enforce isolation between processes and cgroups to define resource quota for each container [69]. Even though many vulnerabilities were found during its development [70], LXC has become a standard in the virtualization approaches and its community has reached a substantial number of contributors. Unfortunately, there are some limitations that are difficult to ignore, especially in some critical scenarios:

- Checkpointing and migration are not included as default features in the vanilla Linux kernel. Consequently, whenever it is necessary to migrate or automate a specific configuration or deployment, there may be some challenges within the process. Third-party tools may be useful in this case, but they are still not reliable to be adopted in a production environment.

- Resource isolation is not straightforward, but it requires substantial effort to configure and test the environment. Even after being sure that enough isolation mechanisms have been deployed, containers can still be affected by the behaviour of other containers, based on the resource quota allocation.

In order to overcome these issues, in 2013 an open-source project named as Docker was established. It is an extension of the LXC approach that

provides a very efficient combination between the already exploited Linux Kernel features and an efficient API server from which the user can easily interact with the container runtime [60]. The architecture of this solution, described in Figure.3.2, is based on the server/client paradigm, as the user can send RESTful API request through a simple command line or the official GUI Client installed with Docker. These requests are then received and managed by the Docker daemon/server, which is going to set up or modify the containers configuration accordingly. A very relevant aspect about this architecture is that an interfaced docker client and a server can be deployed on two different remote hosts. This is a step forward in terms of elasticity compared to the previous available solutions. Whenever a user requests to create a new container, the Docker Daemon is in charge of extracting the required libraries to execute a specific application, which are usually defined in a blueprint, commonly referred to as the Docker Image. These images can be directly pulled from a common repository, also known as the Docker registry, or the user can define the characteristics of an image through a Dockerfile. In addition to basic container management functionalities, support tools for managing networking, resource, storage, user control, and security are also available to the user.

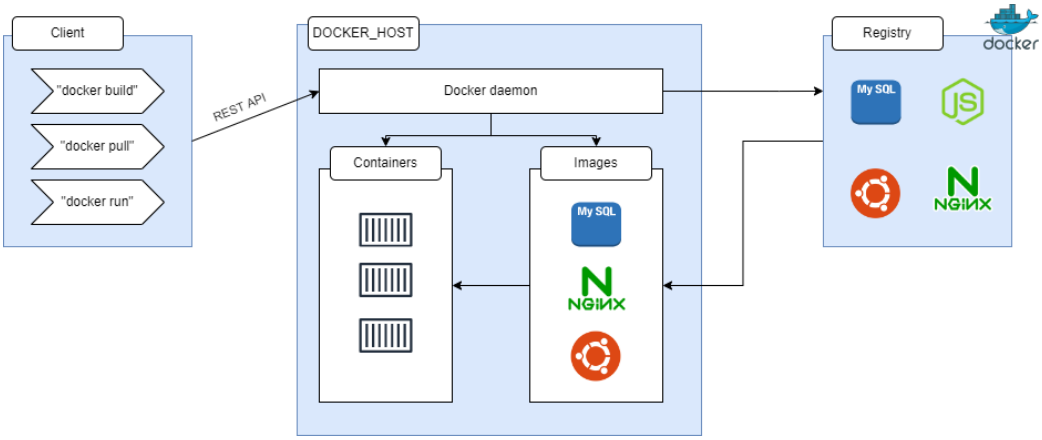


Figure 3.2: High level Docker architecture diagram

The success of Docker container runtime is mainly related to its elastic nature [71]. The necessary time to build and deploy a container is not even comparable to that taken to deploy a complete virtual machine. Thanks to these characteristics, it is possible to develop scalable and dense systems of containers within physical nodes of any data centre, on-premise or cloud, with a very optimal level of resource usage. In conclusion, a good analogy [72] can

summarise what containers represent in the IT infrastructure landscape. If virtual machines can be represented as unique pets that are usually managed in a low amount and they are hard to replace, containers, on the other hand, can be considered chickens, as they are usually deployed in a very high amount and can be replaced very easily.

3.3 Kubernetes

While containers, specifically the Docker container runtime, were gaining more popularity among application developers, cloud computing was also beginning to be seriously considered as a standard in infrastructure design [73]. Based on the Cloud Native Computing Foundation (CNCF), the term "cloud-native" application is used to identify any software that is developed and deployed by exploiting the distributed computed features established by the cloud delivery approach. Crucial features like resiliency, elasticity, scalability and flexibility have characterized this class of application in comparison to the previous development and deployment approaches.

CNCF promotes different projects that are compliant with their ethics and best practises. Depending on the degree of maturity, each project can be identified in a specific category: sandbox, incubating and graduated. Among these categories, the one that includes all projects with a sufficient level of adoption is the "graduated" one [74]. One of the most popular "graduated" projects is Kubernetes (k8s). Originally developed and introduced by Google in 2014, k8s consists in an open source container orchestrator. The main objective of this project is to extend the functionality of any supported container runtime by providing a reliable and elastic system to deploy and manage a large number of containers [75]. In a market where microservice-based applications have nowadays an important role in cloud computing, being capable of supporting highly efficient automation and monitoring approaches is essential for achieving the current demanding requirements. In one of the most widely adopted relative manuals [76], the success of k8s can be identified by the following main characteristics:

- **Self-healing and scaling functions**: as reliability is one of the critical requirements in modern systems, a system that autonomously reacts to failures or provisioning issues has a dramatic impact on service delivery and the final user experience. System entities are promptly redeployed whenever a failure is occurring or system capabilities are increased if under-provisioning is forecast.

- **Declarative configuration**: in line with the portability capability

of the Docker container runtime, k8s further increase it by making every single entity of the system a declarative configuration object. In this way, the developer can explicitly define the desired status of a component. Consequently it is possible to reuse the same configuration as a template in very different scenarios and use cases.

- **API and third party cloud providers support**: given the "cloud-native" nature of this approach, the k8s development team has always been focused on guaranteeing that service providers could easily deploy their micro-services on any main cloud infrastructure provider.

Still, in order to understand how k8s can be exploit in specific scenarios, in the next subsections a brief discussion on which are the main components that characterize this approach will be performed. Additionally, some details about the most common use cases will also be presented.

3.3.1 Architecture

In addition to the characteristics described above, another reason for the success of kubernetes lies in its architecture design. In a manual focused on the management of kubernetes-based systems [77], it is highlighted that the design of k8s architecture is based on the philosophy of Unix modularity. Each component has a specific role and can be deployed and configured independently. Consequently, each component does not affect the status of other components, which means that, in the event that a single component is not operating properly, the probability of a general system failure is very low. If this seems a straightforward approach to improve the reliability of deployed applications, on the other hand, the system may result in complex implementation, as more understanding is required for managing a set of modules. From an external point of view, all kubernetes entities are collected in a core logical entity, which is commonly called cluster. Any user can interact with the cluster through an application programming interface (API), which may be considered as a single point of failure of the entire cluster as, in addition to the actual users, it is used by internal components. It is possible to communicate with this API by performing HTTP requests and JSON-format configurations. Its reference component can be identified in the API Server, where API objects are defined in order to provide all the necessary functionalities of a container orchestrator. The following API objects are the basis of any micro-service deployment as they provide core functionalities for the application:

- **Pods**: commonly classified as the atomic unit of a K8s cluster where applications are actually hosted. It consists of one or more containers

that share most of the allocated resources for that specific pod. One of the main functionality is checking the health status of each container: in case a container fails, the pod is in charge of re-deploying it.

- **ReplicaSets**: in order to achieve an optimal level of reliability, it is often necessary to deploy several replicas of the same application. Thus, this object guarantees that a minimum number of containers are deployed and running for a specific application. In addition to this, if the system experience additional load, it is possible to increase the number of deployed containers by modifying the parameters of this object.

- **Deployment**: even though this object could be mistaken for a ReplicaSet, it provides essential functionality for application developers. Whenever is necessary to deploy a new version of an already deployed application, it is usually necessary to substitute the containers that run an older version of the corresponding application image with a new set that run the new version. If the update is performed directly on the ReplicaSet, it would be necessary to delete the one that points the old image with a new one. This may cause downtime, which in some use cases is not welcomed. For this reason, in this specific scenario, the deployment object performs a rolling update where the old ReplicaSet is substituted with the new one only when the new containers are healthy and operating.

Depending on the specific role, there are objects, such as services or Ingress that provide advanced network functionalities such as HTTP load balancing, proxying or firewalling. Basic tools for managing storage resources or configuration files are included within a set of objects that include, for instance, persistent volumes, ConfigMaps, or secrets. In order to guarantee an optimal level of security, dedicated objects for deploying user access and resource permissions are provided. One of the most relevant for this thesis is the **namespace**. It is a logical entity with which it is possible to group K8s objects into subsets. Thanks to specific role-based access control (RBAC) rules, further access restrictions can be applied to any resource of a specific namespace. For further understanding, a namespace can be represented as a directory of objects, as their existence is strictly related to the namespace one: if a namespace is removed, all the objects under it are also removed.

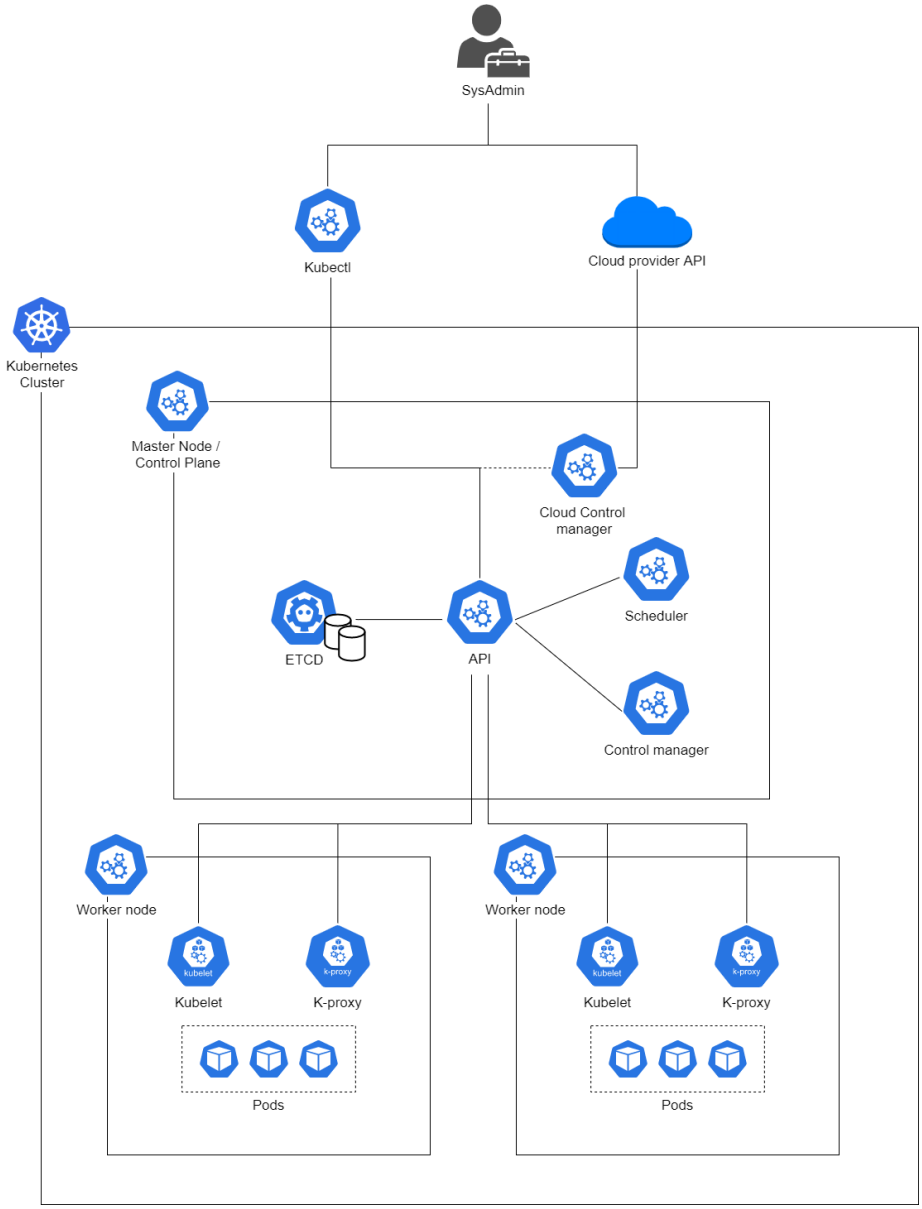


Figure 3.3: High level Kubernetes architecture diagram [78]

As shown in Figure.3.3, a cluster contains a set of nodes which can be classified into two categories. Each category has its own role and the corresponding components:

- **Master node**: it is the most important node in a k8s cluster as it contains all the necessary core components for making the API fully

operating.

  - **etcd**: as discussed previously, one of the key features of k8s is the ability to self-heal and scale. Etcd is a key-value storing system that contains all the information about the desired components status. Consequently, it is possible to employ an approach named "compare and swap", where other core components always check if the actual system status is the same as the one stored in the etcd server.
  - **API server**: identified as the main interface where both internal components and users interact with each other.
  - **Scheduler**: similarly to the other master node components, it is mainly involved in the deployment and management of API objects. It has the capability of identifying which is the best node where objects should be deployed. Its efficiency is based on cooperation with the etcd and API servers.
  - **Controller manager**: despite the previously described components seem to be capable of successfully managing a k8s cluster, the control manager is the actual gearing that drives everything. Thanks to a system of efficient control loops, the other components know of to act whenever a status change occurs.

- **Worker node**: in these category of nodes the actual applications are deployed. In the very rare case where the cluster is made up of a single node, the master node also acts as a worker node. The following components are necessary to execute the required applications and, at the same time, to interact with the components of the master node:

  - **Kubelet**: core component of a worker node, it is responsible for joining the allocated resources of the node into the entire K8s cluster. Furthermore, it is directly interfaced with the master node through the API Server to communicate the health state of the node objects. In case one of the pod containers fails, Kubelet will act directly by redeploying it, without involving the API server. In this way reliability is still guaranteed without increasing the API server traffic.
  - **kube-proxy**: it implements most of the networking related K8s objects, such as the service object. Any established network session between the pod and an external or internal node is filtered based on specific network rules.

3.3.2 Use cases

During its development and evolution, the popularity of kubernetes has grown exponentially within the mass market. This can be proven by looking at the official K8s case studies page (`https://kubernetes.io/case-studies/`). Here, different companies from totally different market sectors have successfully adopted kubernetes for their internal tasks and they have successfully experienced a positive impact. For example, the European Organisation for Nuclear Research (CERN) collects a huge amount of data on a daily basis. Therefore, they require an efficient infrastructure that can satisfy their strict storage and data processing needs. Currently, they rely on a hybrid solution where cloud resources could offer additional computational resources during exceptional peaks, such as during big conferences. In this case, Kubernetes provided an elastic approach for autonomously increasing the required resources in a few minutes, and at the same time less resources are required for the virtual environment in comparison to the previous adopted solution.

A different reason motivated Adidas in their choice of adopting Kubernetes as their main environment for hosting e-commerce sites: many benefits were found in increasing the efficiency of their continuous delivery model, as with Kubernetes they could finally design their solution "starting from the developer point of view". Release frequency went from 4-6 weeks to 3-4 times a day, opening new business opportunities and being more competitive towards other e-commerce stakeholders.

Finally, another relevant example, especially in accordance with the analysis of this thesis, is the Nokia use case. By exploiting the K8s capabilities, it was possible to develop infrastructure-agnostic behaviour software for any component of the mobile network infrastructure. The scalability and reliability of this solution have allowed Nokia to develop a very competitive solution for 5G infrastructure deployments.

3.4 Summary

Virtualisation is now an essential paradigm in modern infrastructures. Most of the daily life services rely on a specific virtualisation approach, as most security and performance requirements can be easily satisfied. Due to lightweight virtualisation, a further step has been taken in terms of performance and maintainability. Despite the evident advantages brought by virtualization, there could be some cases where security between users can raise some concerns. In the next chapter, a detailed analysis on a specific Kubernetes case will be performed.

Chapter 4

Multi-Tenancy in Kubernetes

As cloud computing is one of the most adopted design paradigm in modern infrastructures, the term "tenancy" has become relevant in this context. This is mainly due to the nature of cloud computing, where the same physical resources are shared between different stakeholders. In this chapter, a brief analysis of what tenancy means in the kubernetes universe is performed. Details about different grades of tenancy are presented along with the available solutions, which are systematically evaluated in Chapter 5.

4.1 Hard and Soft Multi-Tenancy

In a 2018 paper [79], an Oracle engineer performed a detailed analysis on the relevance of Multi-Tenancy in the current application development and deployment scenario. In general, as explained by the author, a tenant is basically the customer of a cloud service provider. It shares any entity that can be included in the Everything-as-a-Service (XaaS) model [80] with a wide set of users. The service provider is in charge of developing a solution that can guarantee a satisfactory level of isolation between tenants, depending on the requirements of the use cases. This is enforced by the strict privacy and data processing regulations that have been defined during the last years. Once the definition of multi-tenancy was clearly explained, the author of [79] compares the container-based virtualisation, previously described in Chapter 3, with the novel serverless computing approach. This new approach completely abstracts any detail related to the infrastructure from the actual application deployment. Consequently, developers are no longer concerned with configuring and deploying virtual environments with all the relative details. For this reason, the final assumption of the author is that multi-tenancy is no more relevant in the state-of-the-art solutions as the serverless approach implicitly

includes already everything required to isolate different tenants.

Despite this analysis, after four years since its publication, it is possible to observe that container-based computing is still very relevant and the corresponding community has evolved very fast. As multi-tenancy is not implicit in container-based virtualisation, as in serverless computing, additional effort has been spent on the topic. Specifically, in the Kubernetes environment two classes of multi-tenancy have been commonly identified by the community [81] based on the security requirements:

- **Soft Multi-Tenancy**: as suggested by the adjective, in some scenarios it is not necessary to enforce a high level of isolation between tenants. This is a feasible solution when tenants trust each other, for example, when all of them are part of the same organisation but are involved in different projects. The main scope of this approach is to split available resources between different project teams in order to avoid any conflict. Therefore, the approach does not focus on system security, as it does not provide an efficient tool to prevent attacks between tenants. In the end, soft multi-tenancy is only a support tool for any organisation that has multiple teams working on the same infrastructure. In a K8s environment, soft multi-tenancy can be realised by exploiting the available objects in the vanilla version of the container orchestrator. Specifically, all objects related to resource control, such as namespaces, RBAC rules, labels, and selectors, have enough capabilities to satisfy the needs described previously.

- **Hard Multi-Tenancy**: the constraints that motivate infrastructure designers to further increase the isolation between tenants are mainly related to the deployment scenario. In most of the cloud-native solutions, a third party organisation provides physical resources to its users. Ideally, these users should not be aware of the presence of other tenants, especially if there is no business relationship between them. Therefore, according to current data processing regulations, such as the European General Data Protection Regulation (GDPR) [82], any information owned by a tenant must not be accessible or readable to other tenants. Here, tenants do not trust each other and boundaries must be enforced against any kind of security breach. Despite having many API objects for role access and security, reaching an optimal level of isolation between tenants in Kubernetes is often complex and is not yet in compliance with requirements [83] [84].

It is clear that vanilla K8s has many limits in terms of tenant isolation, as the available resource control objects do not provide the required level of

protection from any data leak. Nevertheless, the following section is focused on how the developer and engineer community is currently trying to overcome these limitations.

4.2 Hard Multi-Tenancy Approaches

In March 2019, a group of developers, employed in different IT companies, established a working group to increase Kubernetes multi-tenancy capabilities (`https://github.com/kubernetes-sigs/multi-tenancy`). In these three years, as additional developers have joined the working group, many ideas and new projects related to the topic have been developed and published. During KubeCon 2021, this working group has hosted a speech about the difference between the "Multi-cluster" and "Multi-Tenant" concepts in a Kubernetes environment [85]. During the discussion, it was noted that these two concepts are not mutually exclusive, as there may be some scenarios where multi-tenancy is implemented by designing a multi-cluster system. In the end, the participants of the talk agreed that the approach and the corresponding level of isolation between tenants highly depend on the requirements of the use case. On the official Kubernetes blog, the developers of the working group have posted their main results [86]: depending on the level of isolation between tenants, three main models have been identified, which can be included in the XaaS paradigm. In the next subsections, a brief description of each model will be provided.

4.2.1 Hierarchical namespaces Controller (HNC) with strict RBAC rules - Namespaces-as-a-Service

As seen previously, vanilla kubernetes offers several API objects to logically split the available resources of a single K8s cluster. This model is based on assigning a namespace to each tenant from which it can declare any object. On the other hand, to properly isolate any allocated resource within that specific namespace, it is necessary to configure a set of control objects:

- **Role based access control rules**: each tenant must have access only to a specific set of resources. Thus, it is necessary to deploy a set of access rules to all the resources that are deployed inside the namespace.
- **Network policies**: a critical aspect of sharing resources is networking. By default, any node in a K8s cluster can reach other nodes. This is not acceptable when tenants do not trust each other. Thus, strict

network policies are necessary to avoid the establishment of network traffic between objects of different namespaces.

- **Resource quotas**: the performances of a service implemented by a tenant should not be affected in anyway by other tenants. Consequently, resource quotas must be enforced in order to guarantee that only the allocated resources can be fairly exploited by the single tenant.
- **Labels and selectors**: finally labels and selectors can be exploited to assign a worker node to a specific namespace. Having a dedicated node for a specific tenant decreases the probability of an attack being carried out within the resources allocated to a node.

Within the multi-tenancy working group, an extension of the namespace objects has been developed: Hierarchical Namespace Controller (HNC).

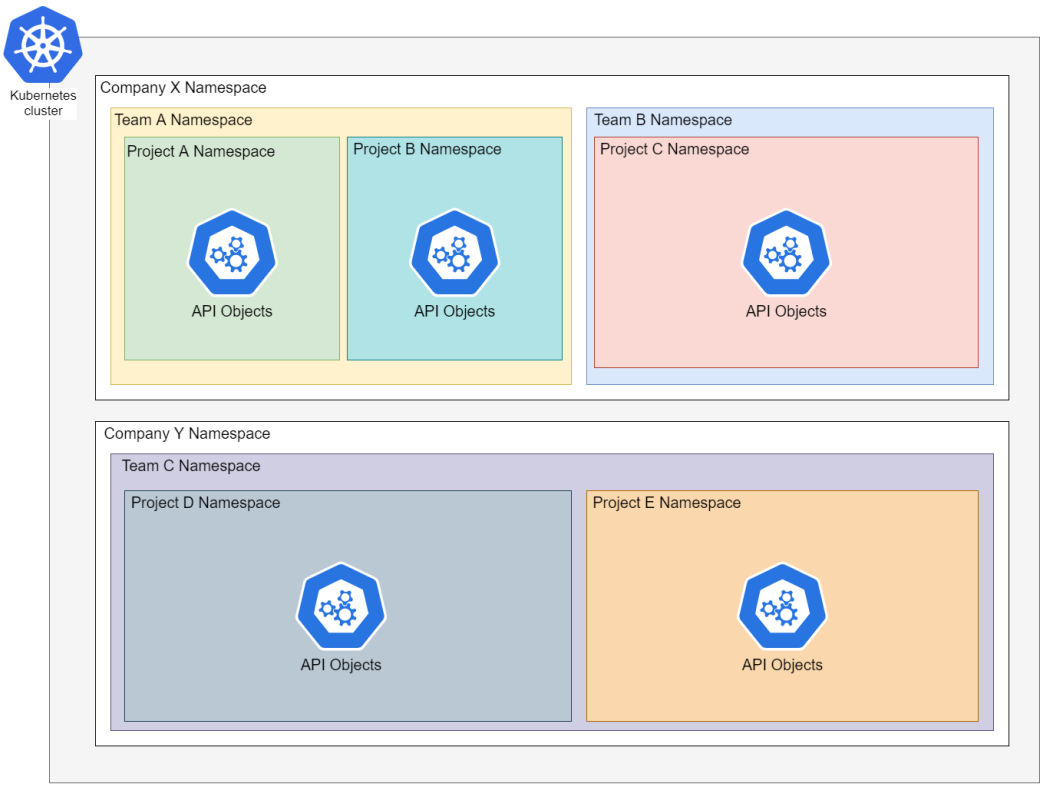


Figure 4.1: Example of HNC deployment scenario in a single cluster - Cluster point of view

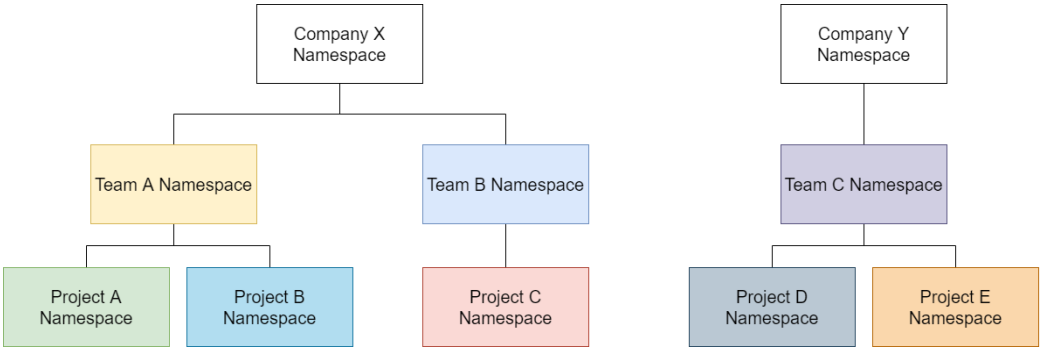


Figure 4.2: Example of HNC deployment scenario in a single cluster - Hierarchical tree diagram

The main goal of this project is to simplify the management of namespaces, especially when the complexity of the organisation cannot be handled by default namespaces. Basically, thanks to this additional custom resource definition (CRD), it is possible to create sub-namespaces within a namespace. This approach does not provide any additional security level for the tenant, as it is still necessary to enforce the control objects described above. This is related to the fact that tenants still share the same control plane and API server, which may be considered as a possible attack vector. Furthermore, if each tenant tries to deploy cluster-wide resources, there may be race conditions and conflicts between them. Still, in some cases where different teams own a set of different projects, it may be useful to have a hierarchical model to be exploited for organisational purposes.

4.2.2 Multi-cluster deployment - Cluster-as-a-service

The most straightforward and most secure model for implementing multi-tenancy is, for sure, the cluster-as-a-service approach. Here, each tenant owns a dedicated cluster, of which resources are not shared with any other tenant. Consequently, complete isolation is guaranteed, since the API server and control plane are not shared with anyone else. Additionally, any cluster-wide resource has no impact on other clusters, which implies that each tenant has more freedom to deploy specific cluster configurations. Although the previous approach has significant advantages, especially in terms of isolation, there could be some limitations that may be critical in some use cases. For example, if the available resources are limited, it may not be possible to deploy a complete cluster for each tenant. Additionally, the owner of the actual physical infrastructure is in charge of managing a large number of

clusters, which may require an additional support tool for the management of them: projects like Cluster API (CAPI) or Kind provide an efficient interface for managing group of clusters.

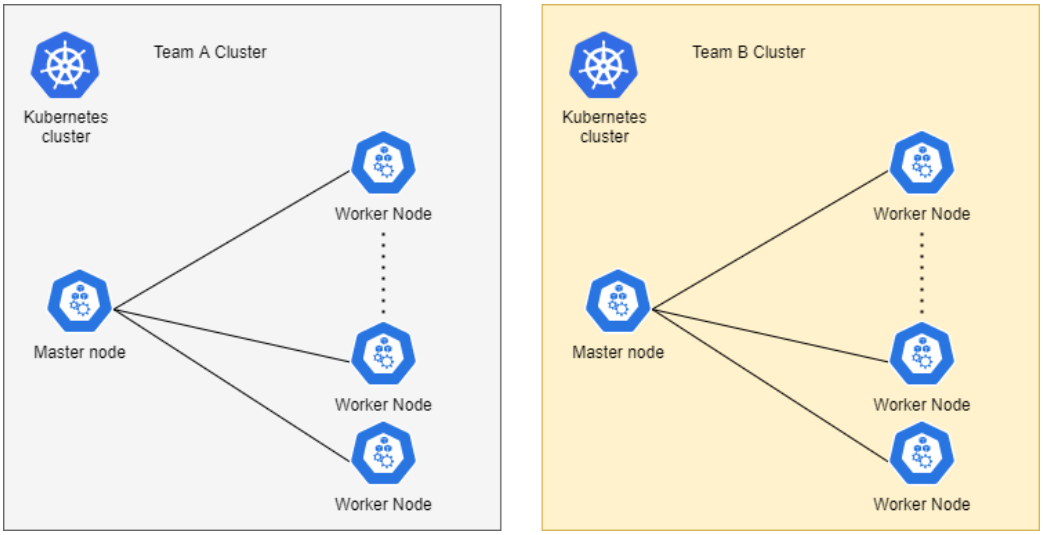


Figure 4.3: Example of Multi Cluster Deployment

4.2.3 Virtual Clusters / Cluster API Provider nested (CAPN) - Control-Planes-as-a-Service

As seen previously, one of the limitations of the HNC approach is the fact that tenants share the same control plane. Multi-tenancy working group have developed a novel solution that tries to overcome this limit: cluster api provider nested, also known as virtual cluster. A very similar project was established by Loft Labs, an American development house focused on developing a kubernetes platform for deploying services. The main design differences are extensively described in an issue thread on the GitHub project repository: `https://github.com/loft-sh/vcluster/issues/5`. Since the Loft Labs project has shown to have more support, documentation, and consequently more production readiness, analysis will be performed on it.

Virtual Cluster is an extension of the Kubernetes namespace entity, as it provides the capability of deploying a tenant-dedicated control plane inside the namespace itself. It would be quite complicated to realise this design with the help of CRDs. Consequently, further engineering was required: each virtual cluster control plane has a dedicated API server, a data store, and, finally, a controller manager. Consequently, the tenant can exploit the virtual

control plane in the same way as he would have done with a traditional host, for instance, by using Kubectl to make API requests to the virtual cluster API server. In this way, the tenant is not directly interfaced with the host API server but with a virtual instance of it.

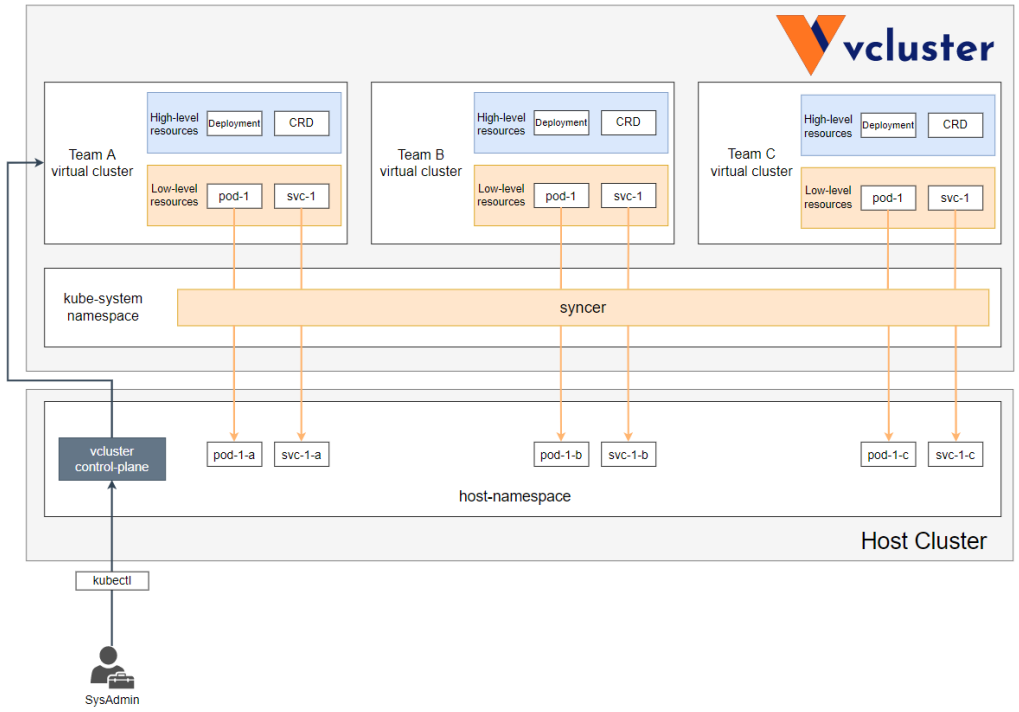


Figure 4.4: Example of Virtual Cluster Deployment

The only core component that is not present in a virtual control plane is the scheduler, which is replaced by an additional layer called a "syncer". This additional component plays the crucial role of syncing the virtual cluster with the actual host cluster through the host control plane. Thus, a virtual cluster does not have any actual physical nodes allocated but is directly interfaced with physical resources through the syncer. In conclusion, this approach can be identified as a hybrid solution that tries to exploit the advantages of both HNC and multi-cluster solutions: it is possible to have strong isolation between tenants, similarly to what has been seen in the multi-cluster solution, but, at the same time, the performances are kept on an optimal level as everything is still deployed in a single Kubernetes cluster.

4.3 Summary

Even though there are clear development paths, multi-tenancy implementation still lacks a reference approach, as each available solution has advantages and drawbacks. Thus, the choice of the approach is highly related to the specific scenario and thus uniquely depends on the actual requirements. In the next sections, a performance evaluation of the analysed multi-tenancy methods will be performed. Specifically, the evaluation will be based on the context of a 5G small cell micro-operator local deployment.

Chapter 5

Evaluation of the solutions

In a scenario where multiple micro-operators are interested in deploying their services in a local area, cloud-enabled small cells and the corresponding mobile edge computing capabilities have the potential to offer many business opportunities. The advantages are based on the fact that with this business model it is possible to achieve an optimal level of QoS with restricted operational costs. Third-party mobile infrastructure providers can dynamically allocate their available resources to each micro-operator. In the specific case of the 5G-Essence project, the cloud-enabled small cell framework was successfully tested in different business scenarios [50] [87] [88] [89]. Although deploying services within the radio access network guarantees very low latency, other requirements should not be underestimated. As it is very likely that the micro-operators share the same physical resources, it is crucial to isolate them from each other. Thus, in order to be compliant with the strict security and privacy laws, it is necessary to take on account the previously described hard multi-tenancy concept. Specifically, if Kubernetes is adopted as the main container orchestration for deploying services, the available approaches can be potentially exploited to guarantee a satisfactory level of isolation between tenants. However, since 5G applications have very demanding requirements, it is also very important to assess the impact these approaches have on the performance of the deployed service. Consequently, in this chapter, three applications, one for each 5G use case (URLLC, eMBB, mMTC), will be deployed simultaneously with each presented hard multi-tenancy approach. An evaluation of network performance will be developed for each approach.

5.1 Test Environment

In one of the 5G Essence whitepapers [90], where it is possible to consult a report on the design, the deployment and the testing of the system prototype, an Intel Next Unit of Computing (NUC) was employed as computational unit of the light data centre. This unit consists of a very small chassis, which has been widely used in contexts where energy efficiency and space limitations were critical metrics [91] [92]. Therefore, as in the 5G Essence system, this data centre is usually located in an edge location and is directly interfaced with small cells (Figure 2.5), Intel NUCs are a valid choice. For this reason, the test environment, where the kubernetes hard multi-tenancy approaches have been deployed, has very similar computational capabilities in comparison with a medium-class Intel NUC. The environment is based on an OpenStack virtual instance with the following characteristics:

- 8 VCPU based on the Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz;
- 16 GBs of RAM;
- A volume of 100 GBs mounted in the root directory of the OS;
- A custom image of Ubuntu LTS 20.04.

As it was necessary to deploy kubernetes clusters for every scenario analysed, a cluster API tool was exploited: kind (`https://kind.sigs.k8s.io/`). This support tool provides basic functionalities in order to deploy and manage K8s clusters with the required configuration of nodes and parameters. Once the cluster was set up, the application deployment was handled with the help of Helm (`https://helm.sh/`). This additional tool increases the portability of Kubernetes-based applications, as it is possible to install a specific application very easily with just one command. Each application relies on a specific "Chart", which is a collection of yaml files. Within these files, it is possible to find the application deployment template with the required configuration and K8s object API definitions. Each hard multi-tenancy approach was deployed individually by launching a bash script that includes all the necessary commands. In terms of networking, the OpenStack host machine is located in the same premises as the user equipment from which applications have been tested. Consequently, the average latency experienced while pinging the virtual instance of the user equipment was 3 ms. This increases the realism of the test environment, as the experienced latency is very similar to that usually present in a local 5G application deployments.

5.2 Tested applications

The choice of applications for the evaluation of hard multi-tenancy approaches was based on their characteristics. Each of them should represent a specific 5G use case class, which means that only a subset of metrics are critical for quality of service.

5.2.1 Janus WebRTC Gateway - eMBB use case

For the first class of application, enhanced multi broadband, it was necessary to pick an application that, as stated in [93], should exploit at the same time a very high throughput rate and low latency. Thus, as described in a related state-of-the-art overview [40], augmented reality (AR) is one of the use cases that can be identified as the eMBB case. Specifically, a very common protocol was identified in the context of AR multimedia deployments: WebRTC. This protocol is defined as a set of standards that enable real-time communication between web browsers. The success of this protocol is based on the fact that it is very simple to develop any kind of web application, as it can be directly enabled with the support of HTML5 and JavaScript, which is the modern standard for web developers. For these reasons, in the context of the evaluation activity of this thesis, an open source WebRTC gateway was adopted for the tests: Janus WebRTC. Developed by the Italian softwer house Meetecho, it consists of a general purpose WebRTC server, where each feature is based on plugins. Usually, the WebRTC protocol establishes a peer-to-peer connection with two browsers, and audio/video transmission is directly exchanged and managed by them. However, in the case of Janus WebRTC gateway, since it can also act as a WebRTC media server, in addition to signalling management, it is also in charge of processing the media data according to the plugin configuration. For the purposes of this thesis, the "VideoRoom plugin" was used, where both audio and video streams are delivered from the streamer browser to the viewer browser via the Janus WebRTC Gateway server. In the official Docker hub image register, a Janus Docker image is available. From this image, it was possible to deploy two Kubernetes pods that could run Janus containers. In this way, it was possible to simulate a high availability configuration where the server is replicated in case one of them fails. In addition to the WebRTC gateway servers, an nginx load balancer was deployed to redirect traffic to the healthy node.

5.2.2 EMQX IoT Broker - mMTC use case

The second class of applications is massive machine-type communications (mMTC), characterised by an extreme capacity to manage a large number of connected devices within a network. These devices usually communicate with each other by sending very short packets, which is the main communication scheme that can be found in IoT and machine-to-machine scenarios. Therefore, for this class of applications, an IoT broker was identified as a valid choice.

One of the most widely adopted protocols for this purpose is Message Queue Telemetry Transport (MQTT), which is based on the publish-subscribe communication paradigm [94]. The MQTT Broker is a standalone server that plays the crucial role of delivering any message sent to the target subscription topic. All devices subscribed to that specific topic will receive the message. The importance of the MQTT Broker is mainly related to its tasks: managing a large number of devices with their corresponding connection states, and the produced traffic must be performed optimally in order to deploy an efficient IoT network. In the case of 5G Small Cell deployments, this new class of antennas has been identified as valid enablers for this class of applications, as they are expected to provide additional device capacity [95] [96]. Among the available solutions, the EMQX Community Edition was chosen for this use case. It is defined as a cloud-native and open-source solution as an helm chart for deploying the MQTT Broker server in a K8s cluster is available. Similarly to what was done for the eMBB case, a high-availability configuration was deployed, which consisted of two EMQX broker servers and a nginx load balancer.

5.2.3 QuakeJS - URLLC use case

The last case analysed is ultra-reliable low-latency communication, where latency and reliability are the most critical metrics for evaluating the performance of the service. Among the applications that are part of this class, online gaming has been identified as a latency-critical application. The user experience is highly affected by the lag caused by latency, which can be dramatically reduced by deploying gaming servers within an edge location [97] [98]. Thus, a portable web browser game has been identified as a valid candidate for this class: quakeJS. This porting of one of the most famous first person shooter gamer can be deployed on a Kubernetes cluster, as, also in this case, an helm chart is available for easing the deployment. Two containers are deployed within the target point: one delivers the required game files to the user, while the second hosts the actual game server for the online session.

Consequently, if the user establishes a connection for the first time with the exposed endpoint, the required content of the game will be downloaded, and then the user will be able to join the server game and start playing.

5.3 Deployment configurations

Once the target applications were identified, it was necessary to configure each scenario accordingly. Starting from the most straightforward, the multi-cluster approach is based on three standalone clusters (Figure.5.1), one for each deployed application. Thanks to the Cluster API Kind, it was possible to deploy three clusters with different node configurations within a single script. In the cases of EMQX and Janus deployments, in order to guarantee high availability, 4 worker nodes and 1 control plane were deployed for each application. In this way, replicas of the same service could be deployed on different dedicated nodes. However, the QuakeJS deployment required only two worker nodes and a single master node, as no high availability scenario was deployed. Each cluster provides a dedicated control plane and monitoring platform for tenants, which provides an optimal level of isolation between tenants, as they do not share any component of the cluster. With the second approach, the hierarchical namespace controller, a higher degree of complexity can be seen in the deployment configuration (Figure.5.2). Since Janus and QuakeJS applications can be grouped under the multimedia applications class, a specific namespace was deployed for this class of services, which is owned and managed by a fictional multimedia operator. Each application in this category has its own sub-namespace with the corresponding resources. Similarly, the EMQX Broker service has been deployed under the IoT Operator's namespace. Since it was necessary to isolate the nodes allocated for each application, in addition to the namespace mechanism and the RBAC rules, each cluster worker node was labelled with a specific application label. Therefore, each API object was deployed to a specific node according to the node selector value. In this case, tenants do not share worker nodes but interact with the cluster from the same Server API.

The last approach is based on an even more complex configuration, since an additional layer, the syncer, interfaces the virtual cluster objects with the one actually deployed in the K8s cluster host (Figure.5.3). Similarly to the first approach, each application has its own dedicated virtual cluster. Since the syncer already provides an efficient virtualisation of the actual host Kubernetes resources, it is not necessary to instruct directly the scheduler to assign API objects to a specific node.

Thus, it is possible to proceed with the same deployment approach adopted

in the multi-cluster scenario. Despite the fact that this approach is based on a single K8s cluster, the tenants do not share the same control plane. Consequently, it is possible to achieve the required level of isolation between tenants, even if all of their resources are deployed in a single cluster.

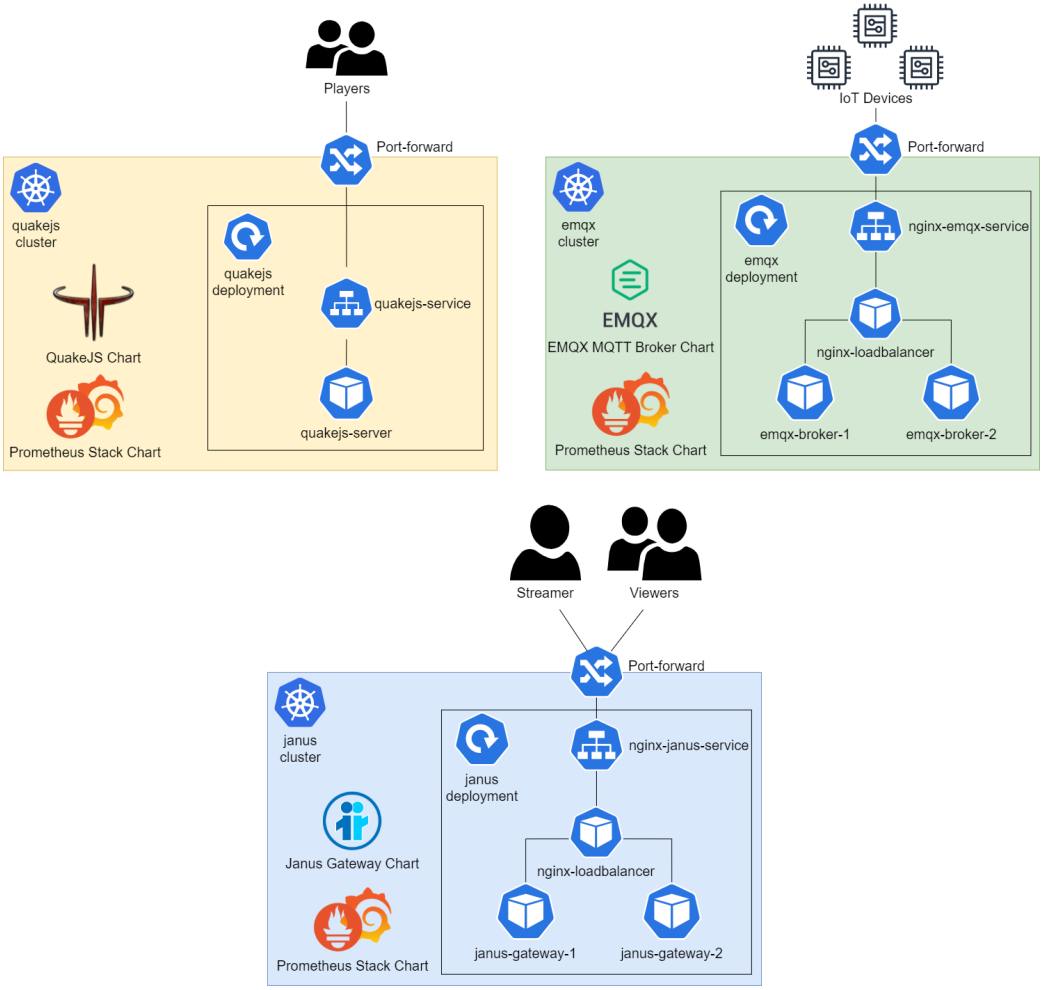


Figure 5.1: Multi Cluster deployment configuration diagram

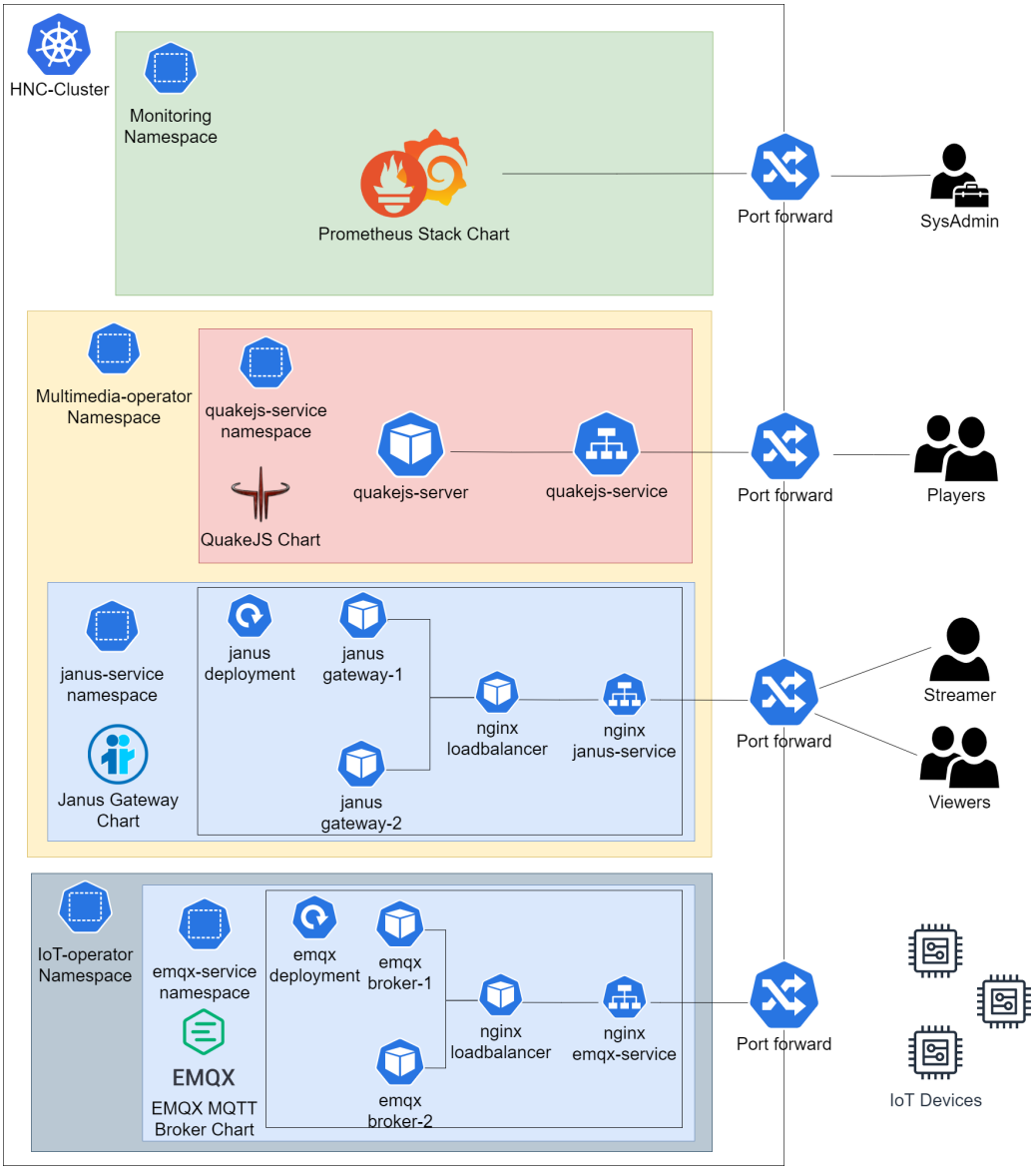


Figure 5.2: Hierarchical Name Space Controller deployment configuration diagram

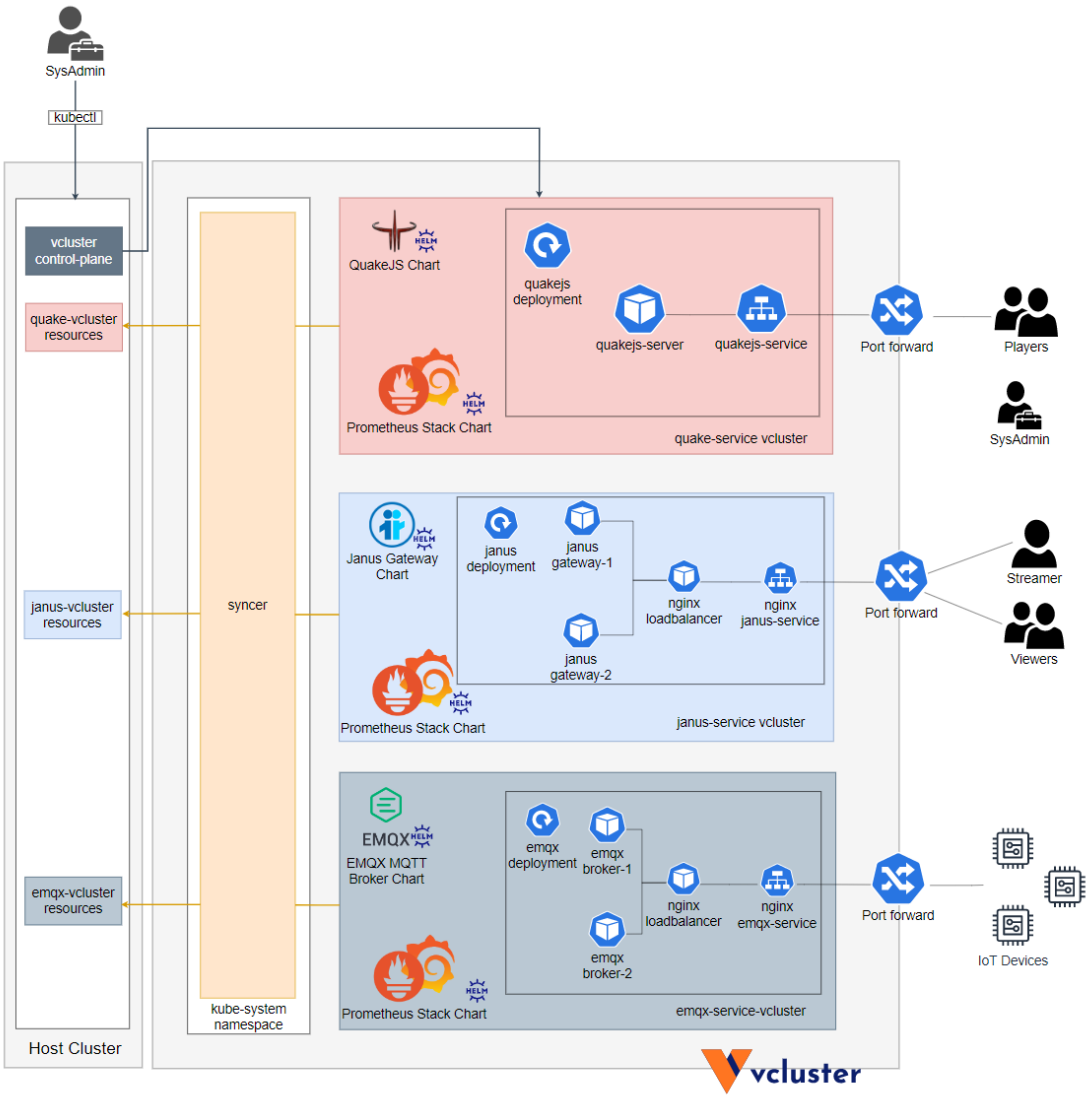


Figure 5.3: Virtual cluster deployment configuration diagram

5.4 Methodology

After the environment was properly set up, it was possible to proceed with the extraction of the required performance data from each application. The main goal is to compare how the selected services perform within each hard multi-tenancy approach. Since the chosen applications have different characteristics

and a very different nature, it was relevant to select a specific set of metrics for each of them.

5.4.1 Metrics definition

The main reference point in terms of standards for mobile networks is 3GPP. Whenever new technology standards are developed, performance requirements are defined. Therefore, hardware and software producers can develop their products on the basis of these common requirements. 3GPP technical specification documents are really specific, as they cover many use cases providing all the necessary details. As the environment test does not include a complete 5G infrastructure, it is not feasible to compare the actual requirements values with those extracted during the testing phase of this thesis. Thus, an evaluation on how much the hard multi-tenancy impacts the performances of the applications was performed.

- In the specific case of **Janus WebRTC gateway** deployment, it is quite complex to target a specific set of metrics, as the WebRTC protocol can potentially be adopted for many scenarios [99]. However, in the context of video and audio transmissions, among the key parameters defined by 3GPP [100], there are two crucial metrics:
  - **Round-trip time (RTT) - milliseconds (ms)**: based on the WebRTC API documentation [101], the observed metric "currentRoundTripTime" is the time required by the Session Traversal Utilities for NAT protocol (STUN) to discover the path characteristics between the two devices. This means that whenever a session is established between two devices, the STUN protocol evaluates which path is the most optimal based on the available ones. In the specific case, as the Janus WebRTC gateway acts as a WebRTC Media Server, the observed round-trip time is based on the STUN behaviour between the Janus WebRTC Gateway and the viewer/streamer.
  - **Throughput - bits per second (bps)**: in general multimedia content requires extremely high levels of throughput. In the case of conference calls, audio and video quality is crucial in order to have an optimal level of QoE between users and avoid disruptions in terms of communication efficiency.
- **EMQX IoT Broker** is involved in a very different scenario. Since the main objective of this class of applications is to provide efficient

connectivity to a substantial number of devices, 3GPP states that the usual parameters, such as the round trip time, could be affected by some influence quantities [102].

  - **Message delivery time - milliseconds (ms), based on the number of connected devices**: message delivery time is the the required by the MQTT broker to deliver a message from a publisher to a subscriber. Since in mMTC use cases, the number of connected devices is a critical parameter, an analysis of how this metric is affected was performed.
  - **Message throughput - messages per second (message/s), based on the number of connected devices**: this metric shows how many messages the MQTT Broker is capable of handling in a specific time interval. Similarly to the previous metric analysed, the number of connected devices was identified as an influence quantity.

- In interactive applications, such as **QuakeJS**, the user experience is mainly affected by how the system responds to any user input [103] [104]. Furthermore, when more than one user interacts with a virtual 3D environment, it is also important to understand how quickly the input of other players is received by the user. Thus, also in this case, 3GPP defines characteristic parameters that are influenced by other quantities [102].

  - **Round-trip time (RTT) - milliseconds (ms), based on the number of connected users**, also known as "Ping" in this specific context, is crucial in terms of QoE. If this value is too high, the player is more likely to experience a very poor gaming session. Since this may linked to the quantity of users connected to the gaming servers, it would be interesting to see if this value is affected by this parameter, in addition to the chosen hard multitenancy approach itself.

5.4.2 Testing tools and methods

Each chosen application is characterised by its specific transmission protocol and communication approach. Therefore, it was necessary to extract the target metrics using the dedicated tools for each application. Additionally, to extract meaningful data, a specific testing strategy was defined for each case.

Starting from the Janus WebRTC gateway case, two devices were involved during the test: one was streaming its own screen content, while the second was receiving and viewing the received payload. The two devices were connected to the same local network and consequently experienced the same RTT when reaching the Janus WebRTC gateway endpoint. For each hard multi-tenancy deployment approach, three streaming sessions of ten minutes were performed. The quality of the streaming content delivered during the testing session is based on the current average quality experienced by most of the available mobile handheld devices: 1920x1080 and 30 frames per second (FPS). Although 5G can very easily handle UHD video streaming [105] with a 4K or 8K resolution, in this testing environment it was not possible to deliver higher quality content due to bandwidth limitations of the network. The required data were collected from the built-in WebRTC Developer Tools available on any Chrome-based browser, available at "edge://webrtc-internals" or "chrome://webrtc-internals". This integrated tool relies on the WebRTC API request "getUserMedia()", which, in addition to the actual video and audio payload, also returns a set of useful streaming metrics.

EMQX IoT Broker was tested using two open source benchmark tools. The first one, called the "MQTT benchmarking tool", is developed by Alexandr Krylovskiy (`https://github.com/krylovsk/mqtt-benchmark`) and consists of a simple tool that tests MQTT broker capabilities. With this tool, it is possible to generate virtual clients and send messages with them through the target MQTT broker. When sending messages, the tool collects data about message delivery time and message throughput. In the case of the thesis test activity, only the message delivery time data was extracted with this tool. The second tool that was used for the evaluation is the "MQTT Stresser", which is developed by inovex GmbH (`https://github.com/inovex/mqtt-stresser`). Likewise, the first tool, virtual clients can be generated in order to stress-test the target MQTT broker. However, with this tool, detailed data about the message is collected. Consequently, the evaluation of the deployment approaches was based on the message throughput data collected with the support of this tool. As it was necessary to understand if the two metrics were affected by the number of connected devices to the MQTT Broker, different scenarios were tested: 10, 25 and 50 devices connected to the broker. The messages were characterised by a Quality of Service (QoS) equal to 1, which in the context of the MQTT protocol means that for every message sent, the publisher must receive an acknowledgement once the MQTT broker completes the delivery.

For the last case, QuakeJS, the approach was similar to the one adopted previously with the MQTT broker: for each deployment approach, different gaming sessions were performed with different amount of users connected to

the game server at the same time. Specifically, scenarios were tested from a single user to three connected users at the same time. The required data were extracted from the built-in command console of the QuakeJS client.

Once all raw data was available, plots were created using R, which is a very common programming language, specifically used for data analysis.

5.5 Results

In the research community, no comparison is available between the presented approaches in terms of network performance. However, in the master thesis [84] and in the conference paper [12], in a scenario where it is necessary to provide hard multi-tenancy within a Kubernetes Cluster, network isolation approaches have been tested in terms of security and performance. Specifically, in [84] a detailed evaluation was performed on a solution based on multiple namespaces, deployed on the same cluster and isolated with a set of RBAC rules and a sidecar container that applies iptables rules to the tenant's network namespace. In [12], hard multi-tenancy was enforced by exploiting network policies and container network interface (CNI) plugins. In both cases, the ability to isolate between tenants did not result in significant overhead in network performance. On the other hand, it is reasonable to expect the tested approaches to behave differently. For instance, it is possible to hypothesise that the virtual cluster approach may suffer some overhead, since there is an additional virtualisation layer between the virtual clusters and the host physical cluster.

5.5.1 Janus WebRTC Gateway - eMBB use case

Once the three WebRTC streaming sessions were performed, the data available in the Appendix Sections A.1.1 and A.1.2, respectively, throughput and RTT related data, were extracted using the Chrome built-in WebRTC analysis tool.

Starting from the round-trip time, in WebRTC based applications this value is mainly affected by the distance between the devices that are exchanging media. However, as previously described, the media payload is actually sent to the Janus WebRTC Gateway, which acts as a WebRTC Media Server. Thus, in this case, the deployed service in the Kubernetes Cluster is also involved in forwarding media content from the streamer to the viewer. Consequently, this means that the collected data are based on the distance between the cluster, where the Janus WebRTC gateway is deployed, and the connected users. From Figure.5.4, it is possible to observe that in most cases,

regardless of the approach used, the streamer and viewer round-trip time is really similar to the one experienced while pinging the host where the Kubernetes cluster is deployed (3 ms). Furthermore, it is possible to observe that during the multi cluster approach testing, a less stable RTT was experienced in specific testing sessions. As explained during the metrics definition, this is not related directly with the deployment approach but with the current network status. This is confirmed by the fact that other testing sessions did not show unstable RTT values, even if a multi cluster approach was adopted.

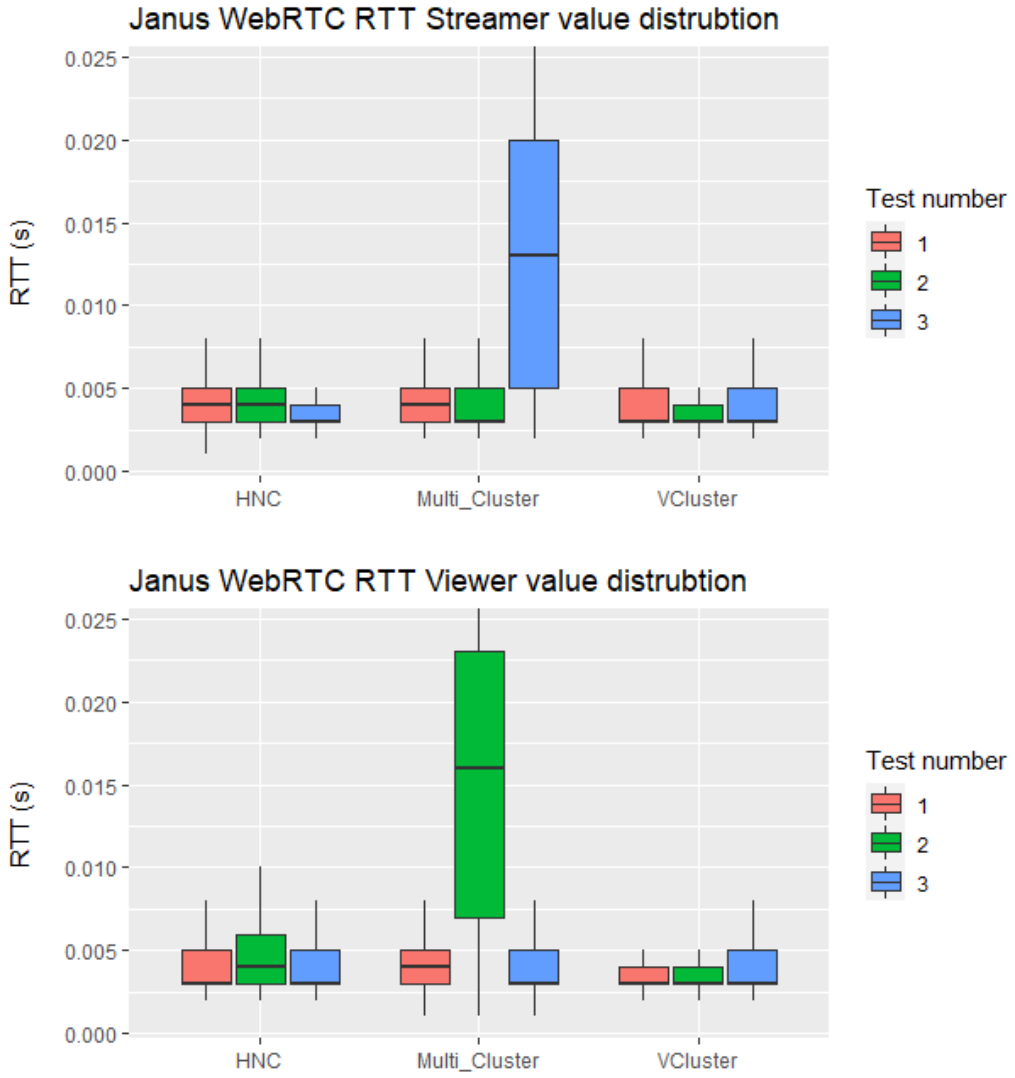


Figure 5.4: Experienced RTT distribution from the streamer and viewer point of view

In terms of throughput, the analysis focused on the upload bitrate of the streamer and the download bitrate of the viewer, which were both based on streaming media content. From this analysis, it was possible to understand whether the available bandwidth was optimally exploited by both users and if the Janus WebRTC gateway caused some throughput loss.

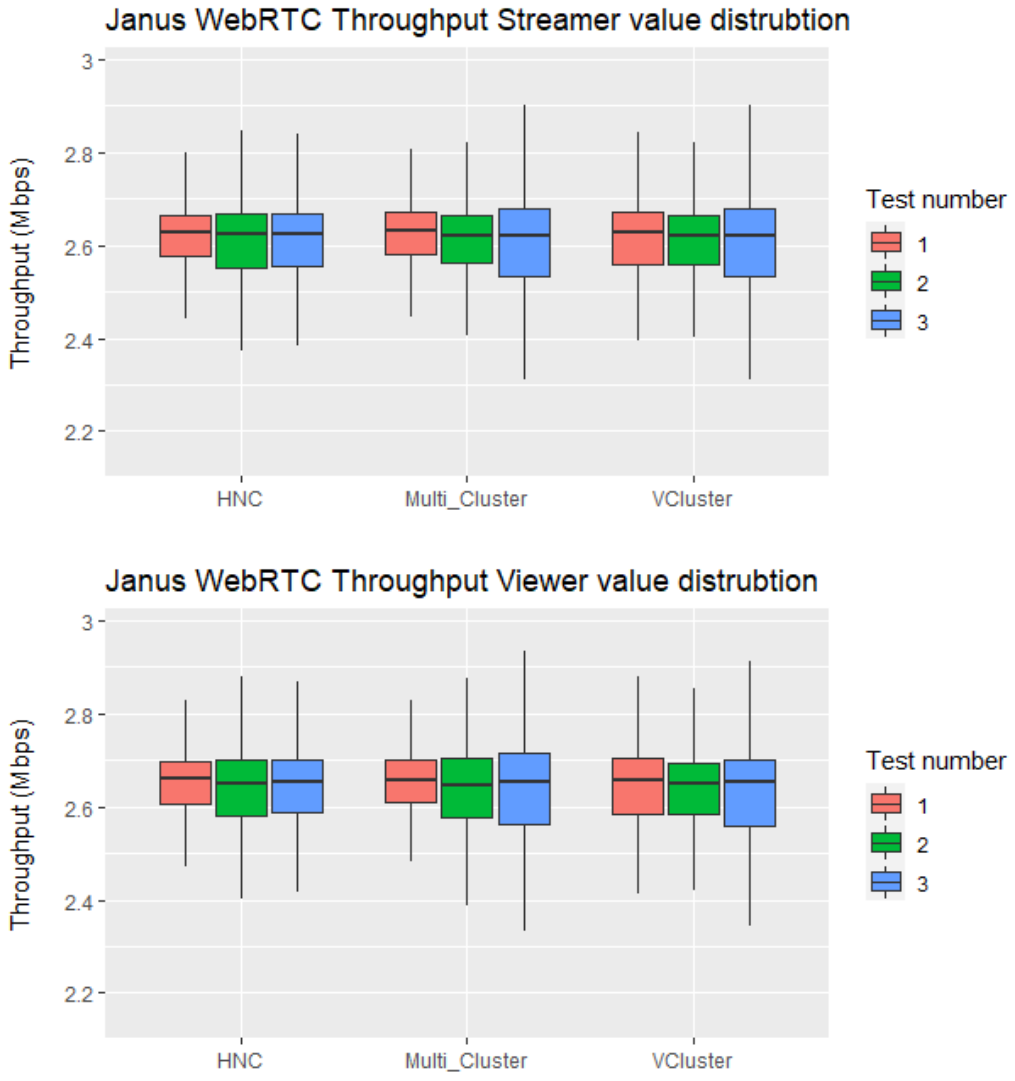


Figure 5.5: Experienced Throughput distribution from the streamer and viewer point of view

From Figure.5.5, it is possible to observe the experienced throughput of both the streamer and the viewer during the streaming session, specifically

the streamer upload throughput and the viewer download throughput. Janus VideoRoom Plugin was configured to allocate a maximum bandwidth of 4 Mbits/s for any device connected to the Janus WebRTC Gateway. This was sufficient to stream video content with the quality characteristics described above. In the WebRTC protocol, the bandwidth is managed by exploiting the capabilities of a mechanism known as bandwidth estimation (BWE) [106], which tries to anticipate the behaviour of the network based on the heuristic model. The bandwidth is then allocated accordingly. From the available data of the streaming traffic, it is possible to see that the streaming sessions did not experience any difference in performance, regardless of the adopted deployment approach. On average, a throughput of 2,65 Mbits/s was experienced in every case. Overall stability was also confirmed by the standard deviation of the collected values, which is between 0.15 to 0.20 Mbits/s.

By looking at both RTT and throughput collected data, it is finally possible to observe that the experienced spikes in the multi cluster testing sessions did not have any impact on the experienced throughput. Consequently, the quality of the content delivered was not affected.

5.5.2 EMQX IoT Broker - mMTC use case

Thanks to the previously described MQTT broker benchmark tools, it was possible to extract the data available in the Appendix Sections A.2.1 and A.2.2, which in this case respectively refer to the experienced Message Delivery Time and Message Throughput.

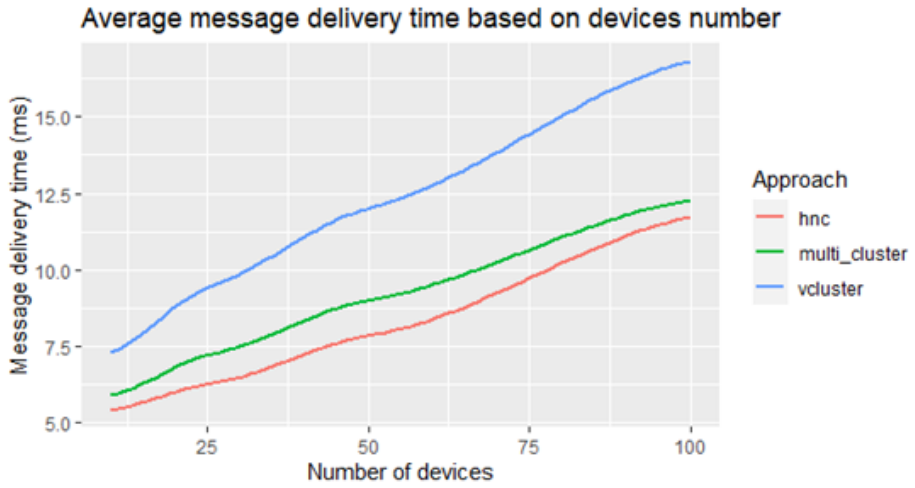


Figure 5.6: Average message delivery time

In Figure.5.6, it is possible to observe how the message delivery time is on average related to the number of devices connected to the MQTT broker. Based on the assumption that a single MQTT Broker server starts to suffer performance degradation after a certain amount of simultaneous connections [107] [108], the delivery time is directly proportional to the number of connected devices, also in the case of this thesis testing activity. This is related to the fact that the MQTT broker has to manage more connections in the same instant, which obviously has an impact on the time necessary to deliver a message from the publisher to the subscribers. Despite the fact that a similar behaviour was observed with all the approaches, in terms of absolute values, the HNC and Multi-Cluster approaches experienced a lower level of delivery time compared to the Virtual Cluster approach: with the same number of connected devices, the Virtual Cluster approach experienced about 25-30% of higher message delivery time compared to the other approaches. Specifically, even with a low number of devices, while multi cluster and HNC experienced a delivery time below 6 ms, the virtual cluster approach resulted in a delivery time of nearly 7.5 ms. With the highest number of connected devices, the same differences where experienced as the broker deployed with multi cluster or HNC was in average delivering messages in 12.5ms, meanwhile with virtual cluster in 16.25ms. However, since the standard deviation in all deployment cases on average was equal to 0.75 ms, overall stability is experienced, regardless of the adopted approach.

Table 5.1: Publishing message throughput (Messages/s)

Number devices	HNC	Multi-Cluster	VCluster
10	300	209	166
25	263	222	169
50	258	222	169
100	256	221	167

Table 5.2: Receiving message throughput (Messages/s)

Number devices	HNC	Multi-Cluster	VCluster
10	2802	2696	2281
25	2790	2578	2139
50	2613	2871	2171
100	2851	2752	2253

If the delivery time of the message was affected by the number of connected devices, the message throughput was not affected. This is related to

the fact that the broker tries to exploit the maximum available bandwidth of messages sent in an instant, regardless of the input number of messages and connected devices when launching a testing session with the benchmark tools. However, since the main objective is to understand whether the deployment approach affects behaviour, in Tables 5.1 and 5.2, the experienced message throughput of both the publisher and subscribers is compared. In the case of the published messages throughput in the case of HNC and Multi Cluster approach an average of 235-240 messages per second was experienced, meanwhile, in the case of Virtual Clusters, 167 messages per second was the average delivery rate. Similarly, for the received message throughput, HNC and Multi Cluster experienced an average of 2650 messages per second, while with Virtual Clusters 2200 messages per second. In both cases, the message throughput experienced by the virtual cluster approach is about 25-30% lower than the other approaches. Consequently, it is possible to observe that the same behaviour experienced during the message delivery time analysis was also experienced with the message throughput analysis, as the magnitude difference between the approaches is similar.

To better understand the experienced behaviour, packet traffic was captured using TShark on the cluster host. Specifically, two sessions of message delivery between 10 devices were further analysed: the first session was performed within a cluster based on the multi-cluster approach, meanwhile, the second session was performed within a virtual cluster deployment. Once the traffic was captured, WireShark was exploited to analyse the behaviour of the packets and their corresponding flows. Thanks to the filtering feature provided by the WireShark GUI, it was possible to perform an analysis only on the MQTT packets. From the "Protocol Hierarchy Statistics" it is possible to observe that in both cases most of the traffic, about 60% is made of TCP packets, since the MQTT protocol is based on that transport protocol. Within these TCP packets, 50% of them encapsulated the MQTT protocol packets. On the TCP level, window size, segment length, and round-trip time did not show any difference between the two message delivery sessions. Finally, in both cases, 70% of the packets had 40 to 79 bytes length. By looking at the packet traffic analysis, it is possible to observe that the transport protocol is not the root of the different behaviour of virtual cluster approach. Since the deployed application is the same for all approaches, the root cause of the observed behaviour can only be traced to the internal cluster pod-service communication.

5.5.3 QuakeJS - URLLC use case

In the last analysed application class, the traffic involved in the synchronisation of the game between users is based on the User Datagram Protocol (UDP). UDP is characterised by fast transmission as, in comparison to the TCP protocol, there are no acknowledgement or retransmission mechanisms. Consequently, packet management may be characterised by duplicated, out of order, or missing packets. Even though UDP is not reliable, it fits the analysed use case, since QuakeJS is a very fast paced game. However, as seen previously [109], in case of a session characterised by a packet of different size, the UDP protocol may result in a higher RTT compared to TCP. By looking at the Appendix Section A.3.1, the same phenomenon was also encountered during the testing session of this application. Based on the results available in the Appendix Section A.3.1, in Figure.5.7 it is possible to see that on average a higher latency was experienced compared to the one observed during the Janus WebRTC tests.

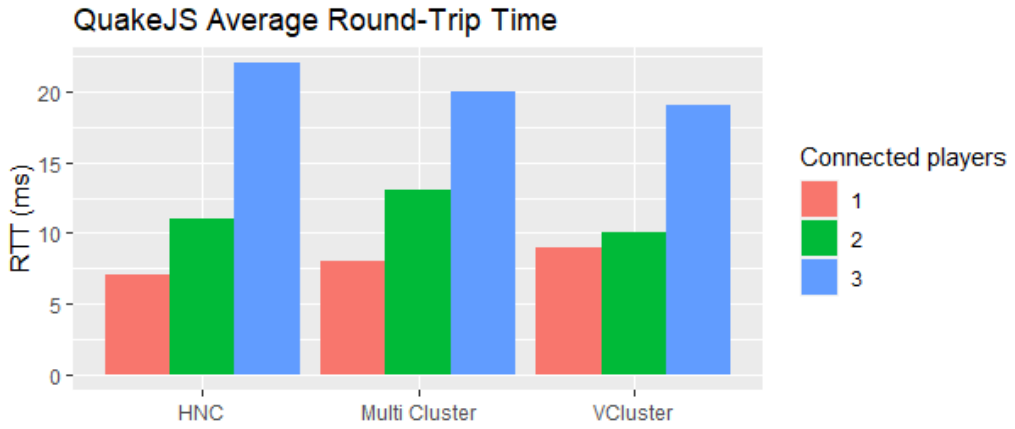


Figure 5.7: Average RTT based on number of connected players

Specifically, in the scenario where only one user was connected to the server, all approaches experienced an RTT of 7 ms on average. By adding more users to the server, in all three approaches, the RTT value fluctuated between 10 and 40 ms. Higher values were reached when 3 players were simultaneously connected to the server. As explained in a fan base documentation [110], the QuakeJS network protocol is based mainly on the actions performed by users, which means that the characteristic of the traffic is very dynamic and highly dependent on the characteristics of each user connec-

tion and hardware. Thus, at least for this case, it is possible to conclude that the adopted Kubernetes multi-tenancy approach had no influence on the performance of the online gaming sessions.

5.6 Evaluation

Since the traffic analysis did not show any obvious differences, the degradation experienced while testing the EMQX broker, within the Virtual Cluster deployment approach, may be explained by looking at the architecture of the approach itself.

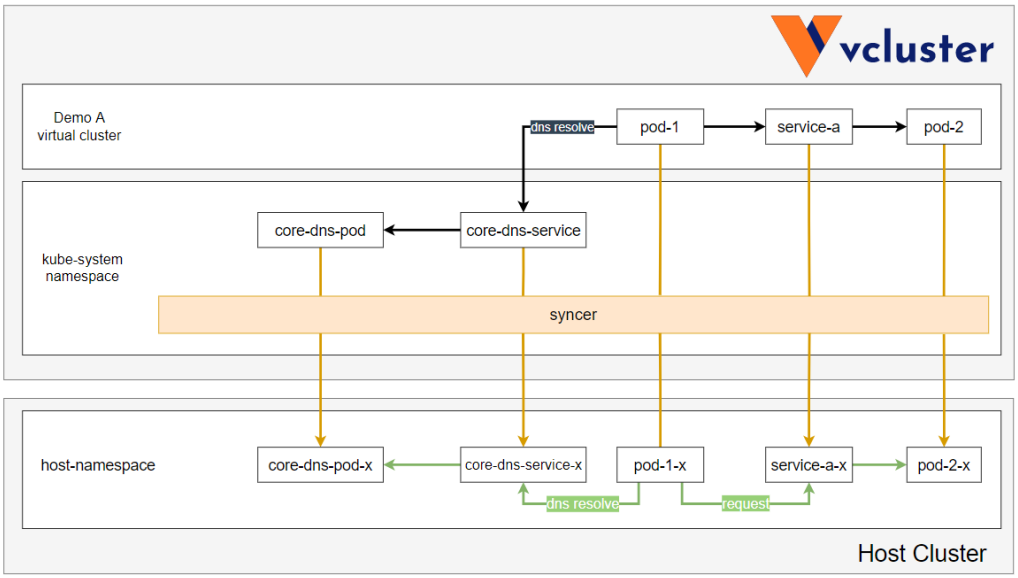


Figure 5.8: Virtual Cluster Networking management [111]

In the case of the VCluster approach, as shown in Figure.5.8, a standalone deployment of the DNS service is placed in the virtual cluster. Consequently, the syncer interfaces the DNS service deployed in the virtual cluster with the one deployed in the host cluster from which the actual traffic is handled. In addition to this, by default, virtual resources like services, endpoints, and the nodes themselves are synced with the corresponding one in the actual host cluster. Consequently, pod-to-service and pod-to-pod traffic is also synced between virtual cluster and host cluster resources, as stated in the official Virtual Cluster documentation [111]. For these reasons, in some cases, additional overhead may be experienced in comparison to the other approaches.

Still, to predict which cases may be affected similarly, it may be useful to analyse which transport protocol is exploited by the tested applications. In the case of the Janus WebRTC gateway, the peer-to-peer session is established by the Interactive Connectivty Establishment (ICE) protocol, which relies on the UDP transport protocol. UDP is used not only for the actual media delivery but also for the signalling between peers. Similarly, as explained previously, QuakeJS is based on the Quake 3 network protocol. Snapshots of the current state of the game session are exchanged very quickly among connected users. This application protocol is based on the UDP transport protocol, since it is necessary to have a very fast exchange of packets even if high packet loss may be experienced. On the other hand, the MQTT protocol relies on the TCP transport protocol, as it is necessary to have a packet loss detection and recovery mechanism, especially when the QoS message delivery is greater than one. Based on the very different nature of the three use cases, it is very complex to make clear assumptions on which hard multi-tenancy approach is absolute best choice. In a scenario where the UDP protocol was exploited as the main transport protocol, like in the eMBB and URLLC cases, no differences were experienced between the three analysed approaches in terms of network performance. Meanwhile, in the case where the TCP protocol was in charge of packet transport, as seen in the mMTC case, the virtual cluster resulted in a worse overall network performance compared to the other two approaches, which shared similar behaviour. In the end, it is clear that application and transport protocol have an important impact on network performance, since there are scenarios where the network performance is highly affected by the application itself and the selected approach has a limited or no influence on the actual user experience.

Chapter 6

Discussion

Test and evaluation activity was useful to understand how the hard multi-tenancy approach behaves while adopted for the deployment of real use-case applications. From the collected results, it is clear that hard multi-tenancy approaches network performance highly depends on the nature of the deployed application. Each approach responded in a different way, depending on the specific behaviour of the application and the corresponding traffic characteristics. Specifically, factors like application and transport protocol, available resources, cluster configuration, user equipment, and network status all have a clear impact on the analysed test sessions. Still, interesting results were observed while testing a specific class of application, which led to thinking that in some cases the choice of a specific hard multi-tenancy approach may have an impact on the network performance. For these reasons, it is quite complex to evaluate which is the best solution solely on this aspect, as within a local micro-operator service deployment, each tenant may deploy applications that are very different in terms of network behaviour or requirements. Furthermore, other aspects like security and resource usage of each deployment method may have a higher priority from an infrastructure provider point of view, as, in many cases, they have to comply with very strict requirements in terms of available hardware or security policies.

Hence, in a design evaluation, network performances could have high priority, only if in a specific local deployment area, all the micro-operators deploy applications of the same class, or at least with similar characteristics and requirements.

6.1 Future work

Since this thesis is focused on the analysis of network performance, it would be interesting in the future to also evaluate the resource usage performance and the isolation degree of each hard multi-tenancy approach.

Evaluating the impact of each approach in terms of CPU, memory, and I/O operations is very relevant, as in 5G local edge deployments, hardware availability may have some limitations or constraints. This is also driven by the fact that the computational units deployed within the RAN cannot physically occupy a large amount of space. Furthermore, during the test activity of this thesis, different hardware resource usage rates were observed while testing each approach. This suggests that it is highly possible that each approach is characterised by different hardware usage.

As discussed previously, security and isolation are very crucial aspects of an hard multi-tenancy scenario, since tenants must be completely isolated between each other. As a starting point for a possible security evaluation activity, the previously presented working group, focused on developing multi-tenancy features for Kubernetes, is currently involved in the development of a useful tool that can evaluate the degree of isolation between cluster tenants: "Multi-Tenancy" benchmarks (`https://github.com/kubernetes-sigs/multi-tenancy/tree/master/benchmarks`).

As this work is focused on the performance evaluation of a local Kubernetes deployment hosted within a virtual OpenStack instance, another relevant research path could be testing multi-tenancy approaches also within the actual 5G Infrastructure. Specifically, it would be interesting to verify if multi-tenancy approaches could be exploited within the Open Source NFV Management and Orchestration (MANO) platform. Finally, as during the Janus WebRTC tests it was not possible to stream multimedia content with a higher resolution than 1920x1080 due to bandwidth limitation, it would be interesting to see how the hard multi-tenancy deployment approaches behave with higher quality multimedia deliveries.

6.2 Conclusion

5G local edge deployments within small cells are characterised by a large amount of requirements that cover different aspects of the system. Since security and isolation between tenants is mandatory in order to comply with the strict policy regulations, hard multi-tenancy mechanisms are highly necessary in order to guarantee the success of the solution in the mass market. However, since there are different classes of applications, it is necessary to

carefully evaluate the choice, as the performance of each approach highly depends on the type of application. Hence, based on the state-of-the-art, there is no standard or straightforward rule that can drive the infrastructure provided to perform a specific choice. In the end, the overall scenario is still complex, and detailed evaluation must be performed whenever it is necessary to design and deliver a micro-operator service platform.

References

[1] T. Dunnewijk and S. Hultén, "A brief history of mobile communication in europe", *Telematics and Informatics*, vol. 24, no. 3, pp. 164–179, 2007, ISSN: 0736-5853. DOI: 10.1016/j.tele.2007.01.013.

[2] S. O'Dea, "4G and 5G services usage in the United Kingdom (UK) in 2021", 2021. Available at https://www.statista.com/statistics/387245/market-share-of-4g-services-in-the-uk/#statisticContainer.

[3] Global System for Mobile Communications Association (GSMA), "The Mobile Economy 2021", 2021. Available at https://www.gsma.com/mobileeconomy/wp-content/uploads/2021/07/GSMA_MobileEconomy2021_3.pdf.

[4] S. Sharma, M. Deivakani, K. Reddy-Srinivasa, A. Gnanasekar, and A. Gira, "Key enabling technologies of 5g wireless mobile communication", *Journal of Physics: Conference Series*, vol. 1817, p. 012 003, Mar. 2021. DOI: 10.1088/1742-6596/1817/1/012003.

[5] O. Arouk and N. Nikaein, "Kube5g: A cloud-native 5g service platform", in *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, 2020, pp. 1–6. DOI: 10.1109/GLOBECOM42002.2020.9348073.

[6] F. A. Wiranata, W. Shalannanda, R. Mulyawan, and T. Adiono, "Automation of virtualized 5g infrastructure using mosaic 5g operator over kubernetes supporting network slicing", in *2020 14th International Conference on Telecommunication Systems, Services, and Applications (TSSA)*, 2020, pp. 1–5. DOI: 10.1109/TSSA51342.2020.9310895.

[7] E. J. Oughton and Z. Frias, "The cost, coverage and rollout implications of 5g infrastructure in britain", *Telecommunications Policy*, vol. 42, no. 8, pp. 636–652, 2018, ISSN: 0308-5961. DOI: 10.1016/j.telpol.2017.07.009.

[8] S. Kumagai, T. Kobayashi, T. Oyama, *et al.*, "Experimental trials of 5g ultra high-density distributed antenna systems", in *2019 IEEE 90th Vehicular Technology Conference (VTC2019-Fall)*, 2019, pp. 1–5. DOI: 10.1109/VTCFall.2019.8891604.

[9] J. B. Moreira, H. Mamede, V. Pereira, and B. Sousa, "Next generation of microservices for the 5g service-based architecture", *International Journal of Network Management*, vol. 30, no. 6, 2020. DOI: 10.1002/nem.2132.

[10] Cloud native computing foundation - CNCF, "Annual survey 2021", 2021. Available at https://www.cncf.io/wp-content/uploads/2022/02/CNCF-AR_FINAL-edits-15.2.21.pdf.

[11] P. Ahokangas, M. Matinmikko-Blue, S. Yrjola, *et al.*, "Business models for local 5g micro operators", in *2018 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*, 2018, pp. 1–8. DOI: 10.1109/DySPAN.2018.8610462.

[12] G. Budigiri, C. Baumann, J. T. Mühlberg, E. Truyen, and W. Joosen, "Network policies in kubernetes: Performance evaluation and security analysis", in *2021 Joint European Conference on Networks and Communications 6G Summit (EuCNC/6G Summit)*, 2021, pp. 407–412. DOI: 10.1109/EuCNC/6GSummit51104.2021.9482526.

[13] R. Janssen, "Categorizing container escape methodologies in multi-tenant environments", University of Amsterdam, MSc Security and Network Engineering Research Project, 2018. Available at https://rp.os3.nl/2017-2018/p80/report.pdf.

[14] C. Bussler, "Multi-tenancy: A concept whose time has come and (almost) gone", in *Proceedings of the 14th International Conference on Web Information Systems and Technologies, WEBIST 2018, Seville, Spain, September 18-20, 2018*, SciTePress, 2018, pp. 316–323. DOI: 10.5220/0006963303160323.

[15] A. Beltre, P. Saha, and M. Govindaraju, "Kubesphere: An approach to multi-tenant fair scheduling for kubernetes clusters", in *2019 IEEE Cloud Summit*, 2019, pp. 14–20. DOI: 10.1109/CloudSummit47114.2019.00009.

[16] B. C. Şenel, M. Mouchet, J. Cappos, O. Fourmaux, T. Friedman, and R. McGeer, "Edgenet: A multi-tenant and multi-provider edge cloud", ser. EdgeSys '21, Online, United Kingdom: Association for Computing Machinery, 2021, pp. 49–54, ISBN: 9781450382915. DOI: 10.1145/3434770.3459737.

[17] S. O. Oladejo and O. E. Falowo, "5g network slicing: A multi-tenancy scenario", in *2017 Global Wireless Summit (GWS)*, 2017, pp. 88–92. DOI: 10.1109/GWS.2017.8300476.

[18] K. Samdanis, X. Costa-Pérez, and V. Sciancalepore, "From network sharing to multi-tenancy: The 5g network slice broker", *CoRR*, vol. abs/1605.01201, 2016. arXiv: 1605.01201.

[19] 3GPP, "Technical Specification Group Services and System Aspects - Release 15 Description", 3rd Generation Partnership Project (3GPP), Technical Report (TR) 21.915, Sep. 2019, Version 15.0.0. Available at https://www.3gpp.org/ftp//Specs/archive/21_series/21.915/21915-f00.zip.

[20] G. Barb and M. Otesteanu, "4g/5g: A comparative study and overview on what to expect from 5g", in *2020 43rd International Conference on Telecommunications and Signal Processing (TSP)*, 2020, pp. 37–40. DOI: 10.1109/TSP49548.2020.9163402.

[21] L. Yang, T. A. Anderson, R. Gopal, and R. Dantu, *Forwarding and control element separation (forces) framework*, RFC 3746, Apr. 2004. DOI: 10.17487/RFC3746.

[22] Huawei Technologies Co., LTD., "5g network architecture - a high-level perspective", Tech. Rep. 11, 2016, pp. 1–16. Available at https://carrier.huawei.com/~/media/CNBG/Downloads/Program/5g_nework_architecture_whitepaper_en.pdf.

[23] D. Witkowski, *Bridging the Gap - 21st Century Wireless Telecommunications Handbook (2nd Edition, 2019)*. Dec. 2019, ISBN: 978-1675085080.

[24] P. Popovski, K. F. Trillingsgaard, O. Simeone, and G. Durisi, "5g wireless network slicing for embb, urllc, and mmtc: A communication-theoretic view", *IEEE Access*, vol. 6, pp. 55 765–55 779, 2018. DOI: 10.1109/ACCESS.2018.2872781.

[25] A. Elnashar and M. A. El-saidny, "Iot evolution towards a super-connected world", in *Practical Guide to LTE-A, VoLTE and IoT: Paving the way towards 5G*. 2018, pp. 310–381. DOI: 10.1002/9781119063407.ch7.

[26] D. Feng, L. Lai, J. Luo, Y. Zhong, C. Zheng, and K. Ying, "Ultra-reliable and low-latency communications: Applications, opportunities and challenges", *Science China Information Sciences*, vol. 64, pp. 1–12, 2021. DOI: 10.1007/s11432-020-2852-1.

[27] B. Kim, "Ict-based business communication with customers in the 4th industrial revolution era", *Business Communication Research and Practice*, vol. 2, pp. 55–61, Jul. 2019. DOI: 10.22682/bcrp.2019.2.2.55.

[28] S. El Hassani, A. Haidine, and H. Jebbar, "Road to 5g: Key enabling technologies", *Journal of Communications*, vol. 14, pp. 1034–1048, Sep. 2019. DOI: 10.12720/jcm.14.11.1034-1048.

[29] 3GPP, "New Radio; User Equipment (UE) radio transmission and reception", 3rd Generation Partnership Project (3GPP), Technical specification (TS) 38.101-1, Jan. 2018. Available at https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3283.

[30] P. Rost, C. J. Bernardos, A. D. Domenico, *et al.*, "Cloud technologies for flexible 5g radio access networks", *IEEE Communications Magazine*, vol. 52, no. 5, pp. 68–76, 2014. DOI: 10.1109/MCOM.2014.6898939.

[31] Global System for Mobile Communications Association (GSMA), *Spectrum in competition policy - appendix 1 of competition policy in the digital age: Case studies from asia and sub-saharan africa*, KubeCon Europe 2021, Dec. 2016. Available at https://www.gsma.com/publicpolicy/wp-content/uploads/2016/12/4.CPITDA_Case_Studies_Asia__Sub-SaharanAfrica_Appendix1-2.pdf.

[32] M. Matinmikko, M. Latva-Aho, P. Ahokangas, S. Yrjölä, and T. Koivumäki, "Micro operators to boost local service delivery in 5g", *Wirel. Pers. Commun.*, vol. 95, no. 1, pp. 69–82, Jul. 2017, ISSN: 0929-6212. DOI: 10.1007/s11277-017-4427-5.

[33] I. Badmus, A. Laghrissi, M. Matinmikko-Blue, and A. Pouttu, "End-to-end network slice architecture and distribution across 5g micro-operator leveraging multi-domain and multi-tenancy", *EURASIP Journal on Wireless Communications and Networking*, vol. 2021, Apr. 2021. DOI: 10.1186/s13638-021-01959-7.

[34] R. Prasad, A. Jukan, D. Katsaros, and Y. Goeleven, "Architectural requirements for cloud computing systems: An enterprise cloud approach", *J. Grid Comput.*, vol. 9, pp. 3–26, Mar. 2011. DOI: 10.1007/s10723-010-9171-y.

[35] T. Boillat and C. Legner, "From on-premise software to cloud services: The impact of cloud computing on enterprise software vendors' business models", *Journal of Theoretical and Applied Electronic Com-*

merce Research, vol. 8, no. 3, pp. 39–58, 2013, ISSN: 0718-1876. DOI: 10.4067/S0718-18762013000300004.

[36] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing—a key technology towards 5g", Tech. Rep. 11, 2015, pp. 1–16. Available at https://www.etsi.org/images/files/etsiwhitepapers/etsi_wp11_mec_a_key_technology_towards_5g.pdf.

[37] A. Ahmed and E. Ahmed, "A survey on mobile edge computing", in *2016 10th International Conference on Intelligent Systems and Control (ISCO)*, 2016, pp. 1–8. DOI: 10.1109/ISCO.2016.7727082.

[38] Q.-V. Pham, F. Fang, V. N. Ha, *et al.*, "A survey of multi-access edge computing in 5g and beyond: Fundamentals, technology integration, and state-of-the-art", *IEEE Access*, vol. 8, pp. 116 974–117 017, 2020. DOI: 10.1109/ACCESS.2020.3001277.

[39] Z. Zou, Y. Jin, P. Nevalainen, Y. Huan, J. Heikkonen, and T. Westerlund, "Edge and fog computing enabled ai for iot-an overview", in *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, 2019, pp. 51–56. DOI: 10.1109/AICAS.2019.8771621.

[40] X. Qiao, P. Ren, S. Dustdar, and J. Chen, "A new era for web ar with mobile edge computing", *IEEE Internet Computing*, vol. 22, no. 4, pp. 46–55, 2018. DOI: 10.1109/MIC.2018.043051464.

[41] X. Sun and N. Ansari, "Edgeiot: Mobile edge computing for the internet of things", *IEEE Communications Magazine*, vol. 54, pp. 22–29, Dec. 2016. DOI: 10.1109/MCOM.2016.1600492CM.

[42] L. M. Contreras and C. J. Bernardos, "Overview of architectural alternatives for the integration of etsi mec environments from different administrative domains", *Electronics*, vol. 9, no. 9, 2020, ISSN: 2079-9292. DOI: 10.3390/electronics9091392.

[43] F. Bonomi, R. Milito, J. Zhu, and S. 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*, ser. MCC '12, Helsinki, Finland: Association for Computing Machinery, 2012, pp. 13–16, ISBN: 9781450315197. DOI: 10.1145/2342509.2342513.

[44] C. Chang, A. Hadachi, J. Mass, and S. N. Srirama, "Mobile fog computing", in *Fog Computing*. John Wiley & Sons, Ltd, 2020, ch. 1, pp. 1–41, ISBN: 9781119551713. DOI: 10.1002/9781119551713.ch1.

[45] D. Muirhead, M. A. Imran, and K. Arshad, "A survey of the challenges, opportunities and use of multiple antennas in current and future 5g small cell base stations", *IEEE Access*, vol. 4, pp. 2952–2964, 2016. DOI: 10.1109/ACCESS.2016.2569483.

[46] P. Mogensen, K. Pajukoski, E. Tiirola, *et al.*, "5g small cell optimized radio design", in *2013 IEEE Globecom Workshops (GC Wkshps)*, 2013, pp. 111–116. DOI: 10.1109/GLOCOMW.2013.6824971.

[47] J. Chen, X. Ge, and Q. Ni, "Coverage and handoff analysis of 5g fractal small cell networks", *IEEE Transactions on Wireless Communications*, vol. 18, no. 2, pp. 1263–1276, 2019. DOI: 10.1109/TWC.2018.2890662.

[48] J. Pérez-Romero, J. Sanchez-Gonzalez, O. Sallent, and A. Whitehead, "On introducing knowledge discovery capabilities in cloud-enabled small cells", Aug. 2017, pp. 680–692, ISBN: 978-3-319-65171-2. DOI: 10.1007/978-3-319-65172-9_57.

[49] M. Spada, A. Kostopolous, I. P. Chochilouros, *et al.*, "5g essence project: A general view and the project pilots results", 2019. Available at http://www.5g-essence-h2020.eu/Portals/0/5G%5C%20ESSENCE_results-Whitepaper_final_v1.0.pdf?ver=2019-12-18-112253-267.

[50] M. R. Spada, J. Pérez-Romero, A. Sanchoyerto, R. Solozabal, M. A. Kourtis, and V. Riccobene, "Management of mission critical public safety applications: The 5g essence project", in *2019 European Conference on Networks and Communications (EuCNC)*, 2019, pp. 155–160. DOI: 10.1109/EuCNC.2019.8802026.

[51] M. D. Hanson, "The client/server architecture", in *Server Management*, Auerbach Publications, 2000, pp. 17–28, ISBN: 9780429114861.

[52] R. J. Creasy, "The origin of the vm/370 time-sharing system", *IBM Journal of Research and Development*, vol. 25, no. 5, pp. 483–490, 1981. DOI: 10.1147/rd.255.0483.

[53] G. J. Popek and R. P. Goldberg, "Formal requirements for virtualizable third generation architectures", *Commun. ACM*, vol. 17, no. 7, pp. 412–421, Jul. 1974, ISSN: 0001-0782. DOI: 10.1145/361011.361073.

[54] Y. Yu, "Os-level virtualization and its applications", Ph.D. dissertation, 2007, ISBN: 9780549914075. DOI: 10.5555/1571423.

[55] A. Moga, T. Sivanthi, and C. Franke, "Os-level virtualization for industrial automation systems: Are we there yet?", in *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, ser. SAC '16, Pisa, Italy: Association for Computing Machinery, 2016, pp. 1838–1843, ISBN: 9781450337397. DOI: 10.1145/2851613.2851737.

[56] R. Rose, "Survey of system virtualization techniques", Oregon State University, MSc Electrical and Computer Engineering Master Thesis, 2004. Available at https://ir.library.oregonstate.edu/concern/graduate_projects/t148fh24b?locale=en.

[57] E. Bugnion, S. Devine, M. Rosenblum, J. Sugerman, and E. Y. Wang, "Bringing virtualization to the x86 architecture with the original vmware workstation", *ACM Trans. Comput. Syst.*, vol. 30, no. 4, Nov. 2012, ISSN: 0734-2071. DOI: 10.1145/2382553.2382554.

[58] A. M. Joy, "Performance comparison between linux containers and virtual machines", in *2015 International Conference on Advances in Computer Engineering and Applications*, 2015, pp. 342–346. DOI: 10.1109/ICACEA.2015.7164727.

[59] S. J. Vaughan-Nichols, "New approach to virtualization is a lightweight", *Computer*, vol. 39, no. 11, pp. 12–14, 2006. DOI: 10.1109/MC.2006.393.

[60] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes", *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, 2014. DOI: 10.1109/MCC.2014.51.

[61] H. Baba, M. Matsumoto, and K. Noritake, "Lightweight virtualized evolved packet core architecture for future mobile communication", in *2015 IEEE Wireless Communications and Networking Conference (WCNC)*, 2015, pp. 1811–1816. DOI: 10.1109/WCNC.2015.7127743.

[62] R. Osnat, "A brief history of containers: From the 1970s till now", *Acquasec*, Jan. 2020. Available at https://blog.aquasec.com/a-brief-history-of-containers-from-1970s-chroot-to-docker-2016.

[63] The FreeBSD Project Developer Team, "Chapter 15. jails", in *FreeBSD Handbook*. 2022, ch. 15. Available at https://docs.freebsd.org/en/books/handbook/jails/.

[64] M. Furman, *OpenVZ essentials*. Packt Publishing Ltd, 2014, ISBN: 9781782167327.

[65] V. Struhár, M. Behnam, M. Ashjaei, and A. V. Papadopoulos, "Real-Time Containers: A Survey", in *2nd Workshop on Fog Computing and the IoT (Fog-IoT 2020)*, A. Cervin and Y. Yang, Eds., ser. OpenAccess Series in Informatics (OASIcs), vol. 80, Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2020, 7:1–7:9, ISBN: 978-3-95977-144-3. DOI: 10.4230/OASIcs.Fog-IoT.2020.7.

[66] D. P. Bovet and M. Cesati, *Understanding the Linux Kernel: from I/O ports to process management*. O'Reilly Media Inc., 2005, ISBN: 0596554915, 9780596554910.

[67] M. Kerrisk, *Namespaces(7) - linux manual page*, 2021. Available at https://man7.org/linux/man-pages/man7/namespaces.7.html.

[68] R. Seth, *Containers (cgroups): Introduction*, 2006. Available at https://lwn.net/Articles/199643/.

[69] S. Senthil Kumaran, *Practical LXC and LXD: linux containers for virtualization and orchestration*. Springer, 2017, ISBN: 148423023X.

[70] J. Hertz, "Abusing privileged and unprivileged linux containers", *Whitepaper, NCC Group*, vol. 48, 2016. Available at https://www.nccgroup.com/globalassets/our-research/us/whitepapers/2016/june/container_whitepaper.pdf.

[71] B. Bashari Rad, H. Bhatti, and M. Ahmadi, "An introduction to docker and analysis of its performance", *IJCSNS International Journal of Computer Science and Network Security*, vol. 173, p. 8, Mar. 2017. Available at https://www.researchgate.net/publication/318816158_An_Introduction_to_Docker_and_Analysis_of_its_Performance.

[72] R. McKendrick, "The pets and cattle analogy demonstrates how serverless fits into the software infrastructure landscape", Feb. 2018. Available at https://hub.packtpub.com/pets-cattle-analogy-demonstrates-how-serverless-fits-software-infrastructure-landscape/.

[73] A. Nag, "Cloud computing: A paradigm shift in it infrastructure", *CSI Communication*, vol. 38, p. 8, Jan. 2015. Available at https://www.researchgate.net/publication/270958592_Cloud_Computing_A_Paradigm_Shift_in_IT_Infrastructure.

[74] Cloud native computing foundation - CNCF, *Graduated and incubating projects*, 2022. Available at https://www.cncf.io/projects/.

[75] M. Luksa, *Kubernetes in action*. Simon and Schuster, Dec. 2017, ISBN: 9781617293726.

[76] B. Burns, J. Beda, and K. Hightower, *Kubernetes: Up and running*. Dpunkt Heidelberg, Germany, 2018, ISBN: 1491935677.

[77] B. Burns and C. Tracey, *Managing Kubernetes: operating Kubernetes clusters in the real world*. O'Reilly Media, 2018, ISBN: 149203391X.

[78] The Kubernetes Authors, The Linux Foundation, *Kubernetes components - kubernetes official documentation*, 2022. Available at `https://kubernetes.io/docs/concepts/overview/components/`.

[79] C. Bussler, "Multi-tenancy: A concept whose time has come and (almost) gone.", in *WEBIST*, 2018, pp. 316–323. DOI: `10.5220/0006963303160323`.

[80] Y. Duan, G. Fu, N. Zhou, X. Sun, N. C. Narendra, and B. Hu, "Everything as a service (xaas) on the cloud: Origins, current and future trends", in *2015 IEEE 8th International Conference on Cloud Computing*, 2015, pp. 621–628. DOI: `10.1109/CLOUD.2015.88`.

[81] J. Cowan, M. Chandrasekaran, and A. Parmar. "Eks best practices guides - tenant isolation". (2022), Available at `https://aws.github.io/aws-eks-best-practices/security/docs/multitenancy/`.

[82] "2018 reform of eu data protection rules", European Commission. (May 25, 2018), Available at `https://ec.europa.eu/commission/sites/beta-political/files/data-protection-factsheet-changes_en.pdf` (visited on 06/17/2019).

[83] S. A. Kjeserud, V. Rahm, S. Y. Sanden, and E. Tobiassen, "Security within a multi-tenant kubernetes cluster", Norwegian University of Science, Technology (NTNU), BSc Thesis in Digital Infrastructure, and Cyber Security, 2021. Available at `https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/2781186/no.ntnu1/:inspera/:78301194/:82504109.pdf?sequence=1`.

[84] X. Nguyen *et al.*, "Network isolation for kubernetes hard multi-tenancy", Aalto University, MSc Thesis in Security and Cloud Computing (SECCLO), 2020. Available at `https://aaltodoc.aalto.fi/handle/123456789/46078`.

[85] T. Drew, R. Bezdicek, A. Ludwin, and J. Bugwadia, *Multi-tenancy vs. multi-cluster: When should you use what?*, KubeCon Europe 2021, May 2021. Available at `https://kccnceu2021.sched.com/event/iE66`.

[86] T. Drew, R. Bezdicek, A. Ludwin, and J. Bugwadia, *Three tenancy models for kubernetes*, Kubernetes official blog, Apr. 2021. Available at `https://kubernetes.io/blog/2021/04/15/three-tenancy-models-for-kubernetes/`.

[87] J. Fernandez Hidalgo, M. Catalán Cid, P. Sayyad Khodashenas, *et al.*, "A 5g framework for the next generation in-flight entertainment and communications (ng-ifec)", Jun. 2018. Available at `https://www.researchgate.net/publication/327423918_A_5G_Framework_for_the_Next_Generation_In-Flight_Entertainment_and_Communications_NG-IFEC`.

[88] A. Kostopoulos, I. P. Chochliouros, E. Sfakianakis, D. Munaretto, and C. Keuker, "Deploying a 5g architecture for crowd events", in *2019 IEEE International Conference on Communications Workshops (ICC Workshops)*, 2019, pp. 1–6. DOI: `10.1109/ICCW.2019.8757152`.

[89] M.-A. Kourtis, B. Blanco, J. Pérez-Romero, *et al.*, "A cloud-enabled small cell architecture in 5g networks for broadcast/multicast services", *IEEE Transactions on Broadcasting*, vol. 65, no. 2, pp. 414–424, 2019. DOI: `10.1109/TBC.2019.2901394`.

[90] O. Segou and A. Flizikowski, "Prototypes of network services and integration planning for use case 2", Mar. 2019. Available at `https://www.5g-essence-h2020.eu/Portals/0/5G-ESSENCE_Deliverable%206.2_v1.0_Final.pdf?ver=2020-05-29-134509-817`.

[91] L. Ferranti, L. Bonati, S. D'Oro, and T. Melodia, "Skycell: A prototyping platform for 5g aerial base stations", in *2020 IEEE 21st International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*, 2020, pp. 329–334. DOI: `10.1109/WoWMoM49955.2020.00062`.

[92] J. M. Kumar, R. Mahajan, D. Prabhu, and D. Ghose, "Cost effective road accident prevention system", in *2016 2nd International Conference on Contemporary Computing and Informatics (IC3I)*, 2016, pp. 353–357. DOI: `10.1109/IC3I.2016.7917988`.

[93] T. Taleb, Z. Nadir, H. Flinck, and J. Song, "Extremely interactive and low-latency services in 5g and beyond mobile systems", *IEEE Communications Standards Magazine*, vol. 5, no. 2, pp. 114–119, 2021. DOI: `10.1109/MCOMSTD.001.2000053`.

[94] A. Banks, E. Briggs, K. Borgendale, and R. Gupta, *Mqtt version 5.0*, Mar. 2019. Available at `https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html`.

[95] F. Al-Turjman, E. Ever, and H. Zahmatkesh, "Small cells in the forthcoming 5g/iot: Traffic modelling and deployment overview", *IEEE Communications Surveys Tutorials*, vol. 21, no. 1, pp. 28–65, 2019. DOI: `10.1109/COMST.2018.2864779`.

[96] S. Borkar and H. Pande, "Application of 5g next generation network to internet of things", in *2016 International Conference on Internet of Things and Applications (IOTA)*, 2016, pp. 443–447. DOI: 10.1109/IOTA.2016.7562769.

[97] J. N. Syed, S. K. Sharma, M. N. Patwary, and M. Asaduzzaman, "Enhanced URLLC-Enabled Edge Computing Framework for Device-Level Innovation in 6G", Feb. 2021. DOI: 10.36227/techrxiv.13325336.v2.

[98] Y. Jararweh, A. Doulat, O. AlQudah, E. Ahmed, M. Al-Ayyoub, and E. Benkhelifa, "The future of mobile cloud computing: Integrating cloudlets and mobile edge computing", in *2016 23rd International Conference on Telecommunications (ICT)*, 2016, pp. 1–5. DOI: 10.1109/ICT.2016.7500486.

[99] N. Blum, S. Lachapelle, and H. Alvestrand, "Webrtc - realtime communication for the open web platform: What was once a way to bring audio and video to the web has expanded into more use cases we could ever imagine.", *Queue*, vol. 19, no. 1, pp. 77–93, Feb. 2021, ISSN: 1542-7730. DOI: 10.1145/3454122.3457587.

[100] 3GPP, "Service requirements for Video, Imaging and Audio for Professional Applications (VIAPA)", 3rd Generation Partnership Project (3GPP), Technical Specifications (TS) 22.263, Jun. 2021, Version 17.4.0. Available at https://www.3gpp.org/ftp/Specs/archive/22_series/22.263/22263-h40.zip.

[101] H. Alvestrand, V. Singh, and H. Boström, "Identifiers for WebRTC's statistics API", W3C, Candidate Recommendation, Jun. 2022. Available at https://www.w3.org/TR/2022/CRD-webrtc-stats-20220614/.

[102] 3GPP, "Service requirements for the 5G system", 3rd Generation Partnership Project (3GPP), Technical Specifications (TS) 22.261, Mar. 2022, Version 18.6.0. Available at https://www.3gpp.org/ftp/Specs/archive/22_series/22.261/22261-i60.zip.

[103] J. Saldana, J. Fernández-Navajas, J. Ruiz-Mas, E. Viruete Navarro, and L. Casadesus, "The effect of router buffer size on subjective gaming quality estimators based on delay and jitter", in *2012 IEEE Consumer Communications and Networking Conference (CCNC)*, 2012, pp. 482–486. DOI: 10.1109/CCNC.2012.6181008.

[104] O. S. Peñaherrera-Pulla, C. Baena, S. Fortes, E. Baena, and R. Barco, "Measuring key quality indicators in cloud gaming: Framework and assessment over wireless networks", *Sensors*, vol. 21, no. 4, 2021, ISSN: 1424-8220. DOI: 10.3390/s21041387.

[105] J. Nightingale, P. Salva-Garcia, J. M. A. Calero, and Q. Wang, "5g-qoe: Qoe modelling for ultra-hd video streaming in 5g networks", *IEEE Transactions on Broadcasting*, vol. 64, no. 2, pp. 621–634, 2018. DOI: 10.1109/TBC.2018.2816786.

[106] A. Bergkvist, D. Burnett, A. Narayanan, B. Aboba, and T. Brandstetter, *Webrtc 1.0: Real-time communication between browsers*, W3C, Sep. 2018. Available at https://www.w3.org/TR/2018/CR-webrtc-20180927/.

[107] I. H. Jung, J. M. Lee, and K. Hwang, "Mqtt protocol extension for real time location based service", *Webology*, vol. 19, no. 1, 2022. Available at https://www.webology.org/data-cms/articles/20220123013255pmWEB19314.pdf.

[108] D. Soni and A. Makwana, "A survey on mqtt: A protocol of internet of things (iot)", in *International Conference On Telecommunication, Power Analysis And Computing Techniques (ICTPACT-2017)*, vol. 20, 2017, pp. 173–177. Available at https://www.researchgate.net/publication/316018571_A_SURVEY_ON_MQTT_A_PROTOCOL_OF_INTERNET_OF_THINGSIOT.

[109] F. T. AL-Dhief, N. Sabri, N. A. Latiff, *et al.*, "Performance comparison between tcp and udp protocols in different simulation scenarios", *International Journal of Engineering & Technology*, vol. 7, no. 4.36, pp. 172–176, 2018. Available at https://www.researchgate.net/publication/329698255_Performance_Comparison_between_TCP_and_UDP_Protocols_in_Different_Simulation_Scenarios.

[110] J. Fedoryński, *Quake 3 network protocol*, 2020. Available at https://www.jfedor.org/quake3/.

[111] Loft Labs, Inc., *Virtual cluster - network & dns*, 2022. Available at https://www.vcluster.com/docs/architecture/networking.

Appendix A

Results from the evaluation phase

A.1 Janus WebRTC Gateway

A.1.1 Throughput

Table A.1: Streamer throughput in Mbps within Multi Cluster deployment approach

Multi Cluster - Streamer throughput in Mbps			
Test number	Test #1	Test #2	Test #3
Mean	2,62	2,59	2,59
Standard Deviation	0,12	0,18	0,21
Minimum	2,14	1,38	1,55
25th Percentile	2,58	2,56	2,53
50th Percentile	2,63	2,62	2,62
75th Percentile	2,67	2,66	2,68
Maximum	3,04	3,64	3,52

Table A.2: Viewer throughput in Mbps within Multi Cluster deployment approach

Multi Cluster - Viewer throughput in Mbps			
Test number	Test #1	Test #2	Test #3
Mean	2,65	2,62	2,62
Standard Deviation	0,12	0,19	0,21
Minimum	2,17	1,30	1,53
25th Percentile	2,61	2,58	2,56
50th Percentile	2,66	2,65	2,65
75th Percentile	2,70	2,70	2,72
Maximum	3,08	3,67	3,55

Table A.3: Streamer throughput in Mbps within HNC deployment approach

HNC - Streamer throughput in Mbps			
Test number	Test #1	Test #2	Test #3
Mean	2,61	2,59	2,60
Standard Deviation	0,16	0,18	0,17
Minimum	0,92	1,63	1,55
25th Percentile	2,58	2,55	2,55
50th Percentile	2,63	2,62	2,62
75th Percentile	2,67	2,67	2,67
Maximum	3,37	3,57	3,75

Table A.4: Viewer throughput in Mbps within HNC deployment approach

HNC - Viewer throughput in Mbps			
Test number	Test #1	Test #2	Test #3
Mean	2,64	2,62	2,63
Standard Deviation	0,16	0,18	0,17
Minimum	0,93	1,67	1,58
25th Percentile	2,60	2,58	2,59
50th Percentile	2,66	2,65	2,65
75th Percentile	2,70	2,70	2,70
Maximum	3,44	3,58	3,79

Table A.5: Streamer throughput in Mbps within Virtual Cluster deployment approach

Virtual Cluster - Streamer throughput in Mbps			
Test number	Test #1	Test #2	Test #3
Mean	2,59	2,59	2,59
Standard Deviation	0,27	0,17	0,21
Minimum	0,57	1,46	1,55
25th Percentile	2,56	2,56	2,53
50th Percentile	2,63	2,62	2,62
75th Percentile	2,67	2,66	2,68
Maximum	3,88	3,78	3,52

Table A.6: Viewer throughput in Mbps within Virtual Cluster deployment approach

Virtual Cluster - Viewer throughput in Mbps			
Test number	Test #1	Test #2	Test #3
Mean	2,62	2,62	2,60
Standard Deviation	0,28	0,18	0,17
Minimum	0,59	1,56	0,36
25th Percentile	2,59	2,58	2,56
50th Percentile	2,66	2,65	2,66
75th Percentile	2,70	2,69	2,70
Maximum	3,95	3,82	3,92

A.1.2 Round-trip time

Table A.7: Streamer RTT in ms within Multi Cluster deployment approach

Multi Cluster - Streamer RTT in ms			
Test number	Test #1	Test #2	Test #3
Mean	4,63	5,34	13,47
Standard Deviation	3,12	8,68	9,33
Minimum	2,00	2,00	2,00
25th Percentile	3,00	3,00	5,00
50th Percentile	4,00	3,00	13,00
75th Percentile	5,00	5,00	20,00
Maximum	21,00	92,00	38,00

Table A.8: Viewer RTT in ms within Multi Cluster deployment approach

Multi Cluster - Viewer RTT in ms			
Test number	Test #1	Test #2	Test #3
Mean	4,59	16,00	3,98
Standard Deviation	3,68	11,00	2,29
Minimum	1,00	1,00	1,00
25th Percentile	3,00	7,00	3,00
50th Percentile	4,00	16,00	3,00
75th Percentile	5,00	23,00	5,00
Maximum	36,00	118,00	17,00

Table A.9: Streamer RTT in ms within HNC deployment approach

HNC - Streamer RTT in ms			
Test number	Test #1	Test #2	Test #3
Mean	4,59	4,90	4,04
Standard Deviation	3,25	3,88	2,56
Minimum	1,00	2,00	2,00
25th Percentile	3,00	3,00	3,00
50th Percentile	4,00	4,00	3,00
75th Percentile	5,00	5,00	4,00
Maximum	21,00	42,00	18,00

Table A.10: Viewer RTT in ms within HNC deployment approach

HNC - Viewer RTT in ms			
Test number	Test #1	Test #2	Test #3
Mean	4,54	4,88	4,60
Standard Deviation	2,92	3,38	3,94
Minimum	2,00	2,00	2,00
25th Percentile	3,00	3,00	3,00
50th Percentile	3,00	4,00	3,00
75th Percentile	5,00	6,00	5,00
Maximum	19,00	23,00	39,00

Table A.11: Streamer RTT in ms within Virtual Cluster deployment approach

Virtual Cluster - Streamer RTT in ms			
Test number	Test #1	Test #2	Test #3
Mean	4,11	4,11	4,24
Standard Deviation	2,94	4,61	3,11
Minimum	2,00	2,00	2,00
25th Percentile	3,00	3,00	3,00
50th Percentile	3,00	3,00	3,00
75th Percentile	5,00	4,00	5,00
Maximum	35,00	65,00	30,00

Table A.12: Viewer RTT in ms within Virtual Cluster deployment approach

Virtual Cluster - Streamer RTT in ms			
Test number	Test #1	Test #2	Test #3
Mean	4,47	4,33	4,55
Standard Deviation	4,44	5,14	3,79
Minimum	1,00	2,00	2,00
25th Percentile	3,00	3,00	3,00
50th Percentile	3,00	3,00	3,00
75th Percentile	4,00	4,00	5,00
Maximum	42,00	57,00	34,00

A.2 EMQX MQTT Broker

A.2.1 Message Delivery Time

Table A.13: Message delivery time within Multi Cluster approach

10 Devices (ms)			
Test number	Test #1	Test #2	Test #3
Minimum	1,03	1,13	0,72
Mean	5,81	6,01	5,84
Standard deviation	0,19	0,17	0,08
Maximum	14,87	15,52	22,94

25 Devices (ms)			
Test number	Test #1	Test #2	Test #3
Minimum	1,39	1,05	0,60
Mean	6,92	7,21	7,45
Standard deviation	0,12	0,19	0,21
Maximum	19,59	22,10	20,45

50 Devices (ms)			
Test number	Test #1	Test #2	Test #3
Minimum	0,50	0,00	0,98
Mean	8,28	9,08	9,06
Standard deviation	0,30	0,19	0,22
Maximum	27,54	23,22	23,52

100 Devices (ms)			
Test number	Test #1	Test #2	Test #3
Minimum	0,00	1,59	0,00
Mean	10,42	13,19	13,22
Standard deviation	2,00	0,37	0,53
Maximum	32,59	39,16	50,73

Table A.14: Message delivery time within HNC approach

10 Devices (ms)			
Test number	Test #1	Test #2	Test #3
Minimum	0,00	1,10	1,00
Mean	5,51	5,41	5,36
Standard deviation	0,39	0,27	0,12
Maximum	121,48	18,15	18,15

25 Devices (ms)			
Test number	Test #1	Test #2	Test #3
Minimum	0,39	2,34	0,00
Mean	6,41	6,43	5,91
Standard deviation	0,12	0,13	0,15
Maximum	19,20	18,93	19,16

50 Devices (ms)			
Test number	Test #1	Test #2	Test #3
Minimum	0,51	0,00	0,00
Mean	7,38	8,09	8,04
Standard deviation	0,50	0,20	0,20
Maximum	34,71	25,95	21,99

100 Devices (ms)			
Test number	Test #1	Test #2	Test #3
Minimum	0,00	0,00	0,00
Mean	11,34	11,33	12,44
Standard deviation	0,55	0,40	0,62
Maximum	39,19	32,25	43,25

Table A.15: Message delivery time within Virtual Cluster approach

10 Devices (ms)			
Test number	Test #1	Test #2	Test #3
Minimum	3,82	4,36	4,48
Mean	7,10	7,24	7,55
Standard deviation	0,07	0,07	0,09
Maximum	18,66	30,24	17,17

25 Devices (ms)			
Test number	Test #1	Test #2	Test #3
Minimum	4,42	4,74	3,99
Mean	9,50	9,38	9,33
Standard deviation	0,14	0,13	0,13
Maximum	25,49	17,97	23,17

50 Devices (ms)			
Test number	Test #1	Test #2	Test #3
Minimum	3,64	4,22	0,00
Mean	11,74	12,30	11,96
Standard deviation	0,18	0,28	0,29
Maximum	34,86	36,19	30,72

100 Devices (ms)			
Test number	Test #1	Test #2	Test #3
Minimum	0,00	0,00	0,00
Mean	15,35	17,89	17,23
Standard deviation	3,34	0,54	0,59
Maximum	54,10	43,57	70,66

A.2.2 Message Throughput

Table A.16: Message throughput (message/second) within Multi Cluster deployment approach

Multi Cluster - Message Throughput (Message/second) - 10 devices						
Test number	Test #1		Test #2		Test #3	
Publishing/Receiving Throughput	Publishing	Receiving	Publishing	Receiving	Publishing	Receiving
Minimum	213	1939	195	1958	180	2254
Mean	225	2332	206	2343	196	3414
Maximum	228	3095	218	3221	208	4554

Multi Cluster - Message Throughput (Message/second) - 25 devices						
Test number	Test #1		Test #2		Test #3	
Publishing/Receiving Throughput	Publishing	Receiving	Publishing	Receiving	Publishing	Receiving
Minimum	199	2230	170	977	195	2089
Mean	224	2926	228	2108	215	2700
Maximum	260	4518	241	4358	230	5145

Multi Cluster - Message Throughput (Message/second) - 50 devices						
Test number	Test #1		Test #2		Test #3	
Publishing/Receiving Throughput	Publishing	Receiving	Publishing	Receiving	Publishing	Receiving
Minimum	199	1669	183	2032	189	1889
Mean	219	2591	218	2958	229	3065
Maximum	249	5420	259	11389	259	5746

Multi Cluster - Message Throughput (Message/second) - 100 devices						
Test number	Test #1		Test #2		Test #3	
Publishing/Receiving Throughput	Publishing	Receiving	Publishing	Receiving	Publishing	Receiving
Minimum	182	1604	191	1614	181	1177
Mean	219	2612	225	3045	221	2601
Maximum	248	7612	267	5945	240	4954

Table A.17: Message throughput (message/second) within HNC deployment approach

HNC - Message Throughput (Message/second) - 10 devices						
Test number	Test #1		Test #2		Test #3	
Publishing/Receiving Throughput	Publishing	Receiving	Publishing	Receiving	Publishing	Receiving
Minimum	255	2822	286	2011	259	2450
Mean	276	3310	331	2261	293	2835
Maximum	288	5781	337	7351	298	4879

HNC - Message Throughput (Message/second) - 25 devices						
Test number	Test #1		Test #2		Test #3	
Publishing/Receiving Throughput	Publishing	Receiving	Publishing	Receiving	Publishing	Receiving
Minimum	242	2444	231	1833	244	2174
Mean	261	3174	261	2449	269	2748
Maximum	280	5282	288	4892	286	3983

HNC - Message Throughput (Message/second) - 50 devices						
Test number	Test #1		Test #2		Test #3	
Publishing/Receiving Throughput	Publishing	Receiving	Publishing	Receiving	Publishing	Receiving
Minimum	244	1354	235	1564	152	1020
Mean	266	2257	262	2670	248	2913
Maximum	299	6561	289	5905	284	6089

HNC - Message Throughput (Message/second) - 100 devices						
Test number	Test #1		Test #2		Test #3	
Publishing/Receiving Throughput	Publishing	Receiving	Publishing	Receiving	Publishing	Receiving
Minimum	204	1901	217	2003	206	1796
Mean	256	2948	262	3019	250	2588
Maximum	286	7691	294	6311	300	8009

Table A.18: Message throughput (message/second) within Virtual Cluster deployment approach

Virtual cluster - Message Throughput (Message/second) - 10 devices						
Test number	Test #1		Test #2		Test #3	
Publishing/Receiving Throughput	Publishing	Receiving	Publishing	Receiving	Publishing	Receiving
Minimum	144	1959	162	1940	155	1589
Mean	158	2234	175	2398	167	2211
Maximum	165	2915	185	3117	181	3675

Virtual Cluster - Message Throughput (Message/second) - 25 devices						
Test number	Test #1		Test #2		Test #3	
Publishing/Receiving Throughput	Publishing	Receiving	Publishing	Receiving	Publishing	Receiving
Minimum	140	1837	165	1415	156	1791
Mean	159	2386	177	1885	173	2146
Maximum	180	3375	203	2818	199	3830

Virtual Cluster - Message Throughput (Message/second) - 50 devices						
Test number	Test #1		Test #2		Test #3	
Publishing/Receiving Throughput	Publishing	Receiving	Publishing	Receiving	Publishing	Receiving
Minimum	140	1837	165	1415	156	1791
Mean	159	2386	177	1885	173	2146
Maximum	180	3375	203	2818	199	3830

Virtual Cluster - Message Throughput (Message/second) - 100 devices						
Test number	Test #1		Test #2		Test #3	
Publishing/Receiving Throughput	Publishing	Receiving	Publishing	Receiving	Publishing	Receiving
Minimum	116	1368	135	1122	145	1637
Mean	169	2236	166	2140	168	2383
Maximum	187	4532	180	3993	193	4359

A.3 QuakeJS

A.3.1 Round-trip time

Table A.19: Round-trip time (message/second) within Virtual Cluster deployment approach

	Virtual Cluster - Round-trip time in ms		
Number of connected players	1 Player	2 Players	3 Players
Minimum RTT	7	7	5
Average RTT	9	10	19
Maximum RTT	11	15	39

Table A.20: Round-trip time (message/second) within HNC deployment approach

	HNC - Round-trip time in ms		
Number of connected players	1 Player	2 Players	3 Players
Minimum RTT	6	8	7
Average RTT	7	11	22
Maximum RTT	14	17	46

Table A.21: Round-trip time (message/second) within Multi Cluster deployment approach

	Multi Cluster - Round-trip time in ms		
Number of connected players	1 Player	2 Players	3 Players
Minimum RTT	4	8	9
Average RTT	8	13	20
Maximum RTT	10	23	51