ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΣΧΕΔΙΑΣΜΟΣ ΚΑΙ ΑΝΑΠΤΥΞΗ ΤΕΧΝΙΚΩΝ ΚΛΙΜΑΚΩΣΗΣ ΣΥΝΑΡΤΗΣΕΩΝ ΤΗΛΕΠΙΚΟΙΝΩΝΙΑΚΟΥ ΔΙΚΤΥΟΥ ΠΕΜΠΤΗΣ ΓΕΝΙΑΣ ΑΝΑΛΟΓΙΚΑ ΜΕ ΤΟΝ ΦΟΡΤΟ

Μεταπτυχιακή Διπλωματική Εργασία

Βασίλειος Ζαλοκώστας-Δίπλας

Επιβλέπων: Αθανάσιος Κοράκης

Βόλος 2023

UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

DESIGN AND IMPLEMENTATION OF SCALING MECHANISMS FOR THE 5G CORE NETWORK FUNCTIONS DEPENDING ON THE NETWORK LOAD

Master Thesis

Vasileios Zalokostas-Diplas

Supervisor: Athanasios Korakis

Volos 2023

Εγκρίνεται από την Επιτροπή Εξέτασης:

Επιβλέπων	**Αθανάσιος Κοράκης** Αναπληρωτής Καθηγητής, Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Πανεπιστήμιο Θεσσαλίας
Μέλος	**Δημήτριος Μπαργιώτας** Αναπληρωτής Καθηγητής, Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Πανεπιστήμιο Θεσσαλίας
Μέλος	**Αντώνιος Αργυρίου** Αναπληρωτής Καθηγητής, Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Πανεπιστήμιο Θεσσαλίας

Ημερομηνία έγκρισης: 28-02-2023

ΕΥΧΑΡΙΣΤΙΕΣ ή ΣΧΟΛΙΑ

Ευχαριστώ τον κ. Αθανάσιο Κοράκη που κατεύθυνε σε αυτό τον τομέα της διπλωματικής μου αυτής εργασίας. Επιπρόσθετα, ευχαριστώ πολύ τον Νίκο Μακρή για την καθοδήγηση του για την περάτωση της εργασίας αυτής.

ΥΠΕΥΘΥΝΗ ΔΗΛΩΣΗ ΠΕΡΙ ΑΚΑΔΗΜΑΪΚΗΣ ΔΕΟΝΤΟΛΟΓΙΑΣ ΚΑΙ ΠΝΕΥΜΑΤΙΚΩΝ ΔΙΚΑΙΩΜΑΤΩΝ

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ρητά ότι η παρούσα διπλωματική εργασία, καθώς και τα ηλεκτρονικά αρχεία και πηγαίοι κώδικες που αναπτύχθηκαν ή τροποποιήθηκαν στα πλαίσια αυτής της εργασίας, αποτελεί αποκλειστικά προϊόν προσωπικής μου εργασίας, δεν προσβάλλει κάθε μορφής δικαιώματα διανοητικής ιδιοκτησίας, προσωπικότητας και προσωπικών δεδομένων τρίτων, δεν περιέχει έργα/εισφορές τρίτων για τα οποία απαιτείται άδεια των δημιουργών/δικαιούχων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον και πληρούν τους κανόνες της επιστημονικής παράθεσης. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο, αρχεία ή/και πηγές άλλων συγγραφέων, αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή. Αναλαμβάνω πλήρως, ατομικά και προσωπικά, όλες τις νομικές και διοικητικές συνέπειες που δύναται να προκύψουν στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής.

Ο Δηλών

Βασίλειος Ζαλοκώστας-Δίπλας

Ημερομηνία

27/02/23

ΠΕΡΙΛΗΨΗ

Με την αλματώδη πρόοδο της τεχνολογίας και την ευρεία διάδοση και χρήση του διαδικτύου στην σημερινή εποχή, οι απαιτήσεις από τους υπολογιστικούς πόρους και τους πόρους του δικτύου έχουν κορυφωθεί. Τα δίκτυα πέμπτης γενιάς (5G) παρέχουν πολύ υψηλές ταχύτητες, ιδιαίτερα χαμηλούς χρόνους απόκρισης, μεγαλύτερη σταθερότητα και ασφάλεια, σε συνάρτηση με το 4G, προσφέρουν αμέτρητες νέες δυνατότητες στον καθημερινό άνθρωπο αλλά και στις επιχειρήσεις. Επίσης, αποτελούν πρωταρχικό ρόλο για την δημιουργία ενός νέου κόσμου εφαρμογών από έξυπνες πόλεις, αυτοκίνητα χωρίς οδηγό, έξυπνη υγειονομική περίθαλψη μέχρι τηλεχειριζόμενη ρομποτική και εφαρμογές εικονικής πραγματικότητας.

Σκοπός της εργασίας αυτής είναι η υλοποίηση

Η εργασία αυτή έχει σαν στόχο την δημιουργία ενός μηχανισμού με στόχο την κλιμάκωση του Δικτύου 5G σε πραγματικό χρόνο σύμφωνα με τον φόρτο που επιβαρύνεται το δίκτυο. Όταν παρατηρείται περισσότερη κίνηση η οποία εισέρχεται μέσω του δικτύου, περισσότεροι 5G Core Network κλώνοι χρησιμοποιούνται που θα διαχειρίζονται τις ανάλογες εργασίες ενώ όταν παρατηρείται λιγότερη τότε οι κλώνοι αυτοί παραμένουν ανενεργοί. Έτσι, οι υπολογιστικοί πόροι χρησιμοποιούνται αποδοτικότερα καθώς οι εργασίες διαμοιράζονται με βάση τον φόρτο του δικτύου σε αυτούς.

ABSTRACT

With the leaps and bounds of technology and the widespread use of the internet in today's era, the demands on computing and network resources have peaked. The fifth generation (5G) networks provide very high speeds, especially low response times, greater stability and security, in relation to 4G, they offer countless new possibilities to the everyday person as well as to businesses. They are also instrumental in creating a new world of applications from smart cities, driverless cars, smart healthcare to remote-controlled robotics and virtual reality applications.

This master thesis focuses on the implementation of a mechanism aimed at scaling the 5G Network in real time according to the network load. When more traffic is observed entering through the network, more 5G Core Network clones are used in order to manage the corresponding tasks, while when less is observed then these clones are idle. In such a manner, more effective use of the available computing resources can be made as tasks are distributed based on the network load that they need to carry out.

Table of Contents

List of Figures

Chapter 1: Introduction

1.1 Background

Over the years, technology ,and thus the internet, has radically changed our world and everyday lives. Additionally, technology for seniors has created incredible tools and services that provide useful information at our fingertips. Companies are moving much faster on their digital transformation journey as they are trying to achieve agility, profitability, sustainability. In the modern era, cloud computing [1] is the number one technology standard to ensure all the above. Cloud computing offers on-demand IT services, including analytics, databases, networking, servers and storage via the internet. It provides companies multiple benefits with the most important ones being the cost-saving as the costs are being reduced, the scalability with the scale ups and downs and the agility as businesses can be responsive to the market changes with ease. There is no doubt that Cloud Computing is becoming the most prominent technological innovation on the 21st century as it is gaining more popularity than ever.

Another area of science that has recently experienced tremendous expansion is the 5G Networks [2]. 5G technology is created to facilitate the interaction of all the machines, objects, devices. It offers high peak data speeds, low latency, more availability, reliability and security that its ancestor (4G). It accomplishes high-speed connectivity and guarantees reliability no matter the location.

Both of these subjects have a favorable impact on the state of the world today. There is some work still to be done before these technologies can be used together to their full potential. But eventually, they will be able to help improve many aspects of our lives.

1.2 Motivation

In the modern era, 5G and cloud computing are picking up increasingly uniquity. Kubernetes, which provides service discovery and load balancing, can expose a Docker container using the DNS name or their own IP address, is the most prevalent cloud orchestrator. This framework gives us the opportunity to automate the scaling procedure, meaning to make the necessary scale ups or down without human interaction with the machines.

Every company today that truly aims to succeed goes without saying that it needs to be able to facilitate growth. In case the expansion of a company outstrips the infrastructure behind it, genuine issues can emerge. So, to maintain a strategic distance from any issues that might hold back a potential success story, it's imperative that all companies fully consider the implications of their growth procedures, especially when network capabilities are concerned. Network scalability is essential for venture victory, since it permits a company to upscale, or downsize when it needs to. So, autoscaling is a very crucial feature for cloud computing science.

All things considered, the combination of cloud computing and 5G technology was the motivation behind this master thesis. To **implement a technique** which will connect and utilize an environment in order to **scale a 5G deployment** and prove that not only the network traffic that passes through the Mobile Core Network multiply. Also, energy is being conserved as the necessary tasks that needs to be carries out as distributed accordingly to every compute resource available.

1.3 Thesis Structure

This thesis is divided in 5 chapters which are being descripted below:

- **Chapter 1**: A summary of the technologies regarding cloud computing and 5G network is being made explaining their importance to individuals as a combined unit. Also, the inspiration is being discussed along with the problem statement.
- **Chapter 2**: A reference to the technical background and the tools utilized, is made so that the research behind this work becomes more explicit.
- **Chapter 3**: How this master thesis is being implemented for the target outcome. Additionally, the process for developing the system architecture and the environment set-up.
- **Chapter 4**: Results and graphs assessing the effectiveness of our implementation.
- **Chapter 5**: Conclusion and Future Work.

Chapter 2: Infrastructure and Experimental Tools

2.1 Chapter Introduction

An abstract of the infrastructure we used to implement this master thesis and the tools needed to carry it out. The technologies that had been used are being descripted and analyzed in depth as much in Kubernetes environment as in the Open Air Interface open-source software.

2.2 NITOS Testbed

For the implementation of this master thesis, we used the NITOS Testbed Laboratory [3] by utilizing 2 of each nodes. NITOS is a testbed created by the NITLab team and is located in Volos, Thessaly, Greece. It comprises of latest generation technology computer resources that highlight remote interfacing and permit experimentation with heterogenous remote (Wi-Fi, WiMAX, LTE, 5G) and wired telecommunications. NITOS is accessible remotely and open to the research community 24/7. It is deployed on the outside of the UTH campus building as shown in the image[1] below with each number to represent a single node.

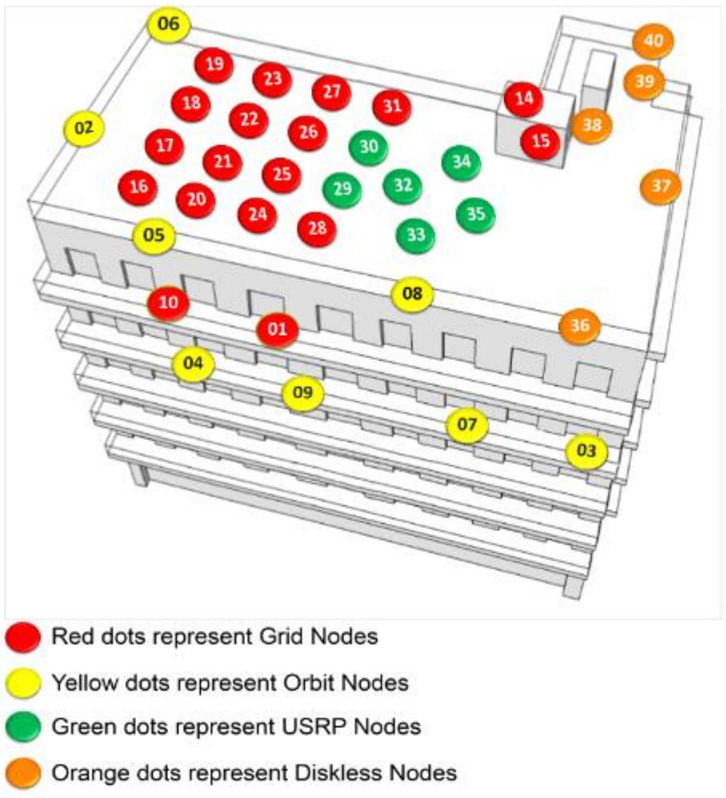

06
19
23
27
31
14
40
39
38
02
18
22
26
30
34
15
17
21
25
29
32
35
37
16
20
24
28
33
05
08
10
01
36
04
09
07
03
🟥 Red dots represent Grid Nodes
🟨 Yellow dots represent Orbit Nodes
🟩 Green dots represent USRP Nodes
🟧 Orange dots represent Diskless Nodes

Figure 1. NITOS Testbed Exterior Implementation

NITOS consists of 2 separate servers that handle the necessary tasks and has a client-server architecture. The first server is a webserver while the second is a console-server. The webserver is responsible for allocating the NITOS resources by the user's command and is in charge of the management of the user access requests. The console-server hosts various network functions and services such as DHCP, NTP etc. With the help of these 2 servers that operate non-stop, NITOS Testbed provides a solution to anyone that want to utilize, experiment, and test their own implementations in a large-scale environment.

[1] http://nitlab.inf.uth.gr/NITlab_old/index.php/testbed

2.3 Docker

2.3.1 Overview

Docker [4] is very popular amongst virtualization technologies and is utilized as the perfect environment for building, delivering and developing applications.
It empowers the programmers to encapsulate their applications and insulate them from the infrastructure in an environment named container. This way software can be delivered faster and more reliable. With Docker, applications and infrastructure can be handled similarly by taking advantage of its technologies. So, the developers are able to altogether decrease the procrastination creating an application and shipping it to the users.

2.3.2 Containers

Docker containers [5] are the main components of the Docker Engine. It is a standardized unit which can be created on the fly to deploy a specific application or environment. Docker containers are known for the isolation that they provide between processes using virtualization techniques. These virtualization advantages gives the opportunity to numerous application components to share the resources of a single instance of the host operating system in the same way that a hypervisor empowers different Virtual Machines to share CPU and memory. Cgroups are a typical example of these isolation and virtualization capabilities. Cgroups (Control Groups) are utilized for apportioning resources among processes and by using namespaces that restrict a processes access and visibility into all the other system resources.
Consequently, containers and Virtual Machines can be compared as they share various similar features, but includes some extra benefits:

1. Smaller size: In contrast to containers, Virtual Machines are burdened with the weight of an entire OS instance. Containers only run the necessary OS dependencies and processes. That’s why containers are measured in megabytes and the included hardware is more effectively utilized.

2. Improved productiveness: Containerized applications can be created once and run in any location. Additionally, VMs are more time consuming in monitoring, shipping and restarting application in comparison with containers.
3. Greater resources efficiency: With containers, developers can run numerous duplicates of an application on the same infrastructure which decrease cloud spending.

2.3.3 Docker Architecture

Docker implements a client-server architecture. Docker client is the medium between Docker Client and the Docker daemon. Docker daemon can be characterized as the heart of the whole system because is in charge of making, deploying and shipping the containers. Docker clients communicate with the daemon through a Restful API which is based on UNIX as depicted in the image[2] below.

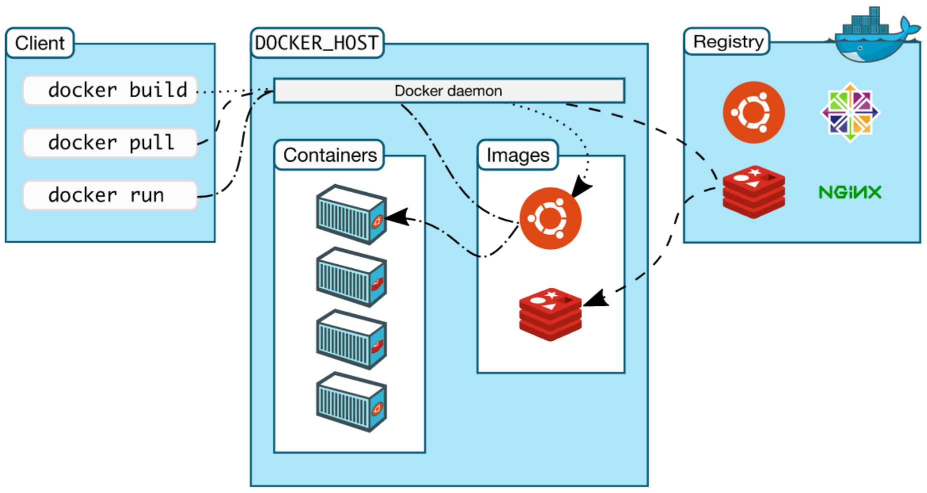


Figure 2. Docker Architecture

There are some more Docker components which needs to be analyzed so the Docker concept can be more comprehensible to the reader. These are the Docker Images and the Docker Registry.

Every Docker Image has read-only restriction which includes directions on how to develop and run containers in this particular environment. Docker gives the opportunity

[2] https://docs.docker.com/get-started/overview/

to developers to team up applications and treat them as a unit. Also, preconfigured server environments can be initiated which the developers can utilize for private or public (with other Docker developers) use. Docker image is a collection of files that include all the necessary software to run such as installations, application code and code dependencies. These images can all be saved in a storage, which is available in totally different labeled forms, called Docker registry. Developers can pull or push images from a registry in order to have them locally on their pc or to upload them privately or publicly to the Docker registry. In this master thesis, DockerHub was utilized to store and pull the Docker images used which is essentially a facilitated repository for uploading, downloading and using container images.

2.4 Kubernetes

2.4.1 Overview

Kubernetes (K8s) [6] is an open-source container orchestration system that is used for automating software deployment, scaling and management. It is an extensible and portable platform that oversee workloads and services. Kubernetes includes built-in commands for deploying applications in the cloud, rolling out updates, scaling up and down deployments according to the developers needs and operates along with virtualization software that packages up apps into containers. Docker, which is the most popular virtualization standard, is commonly used as a container runtime that Kubernetes orchestrates.

2.4.2 Kubernetes Architecture

Kubernetes gives the opportunity to developers to create, deploy and manage applications using a set of basic tools called primitives. It follows the primary/replica architecture [7] and its components can be separated into those that control an individual node (Kubelets or Worker Nodes) and those that are portion of the control plane (Master Nodes) as shown in the picture[3] below.

[3] https://www.toptal.com/kubernetes/what-is-kubernetes

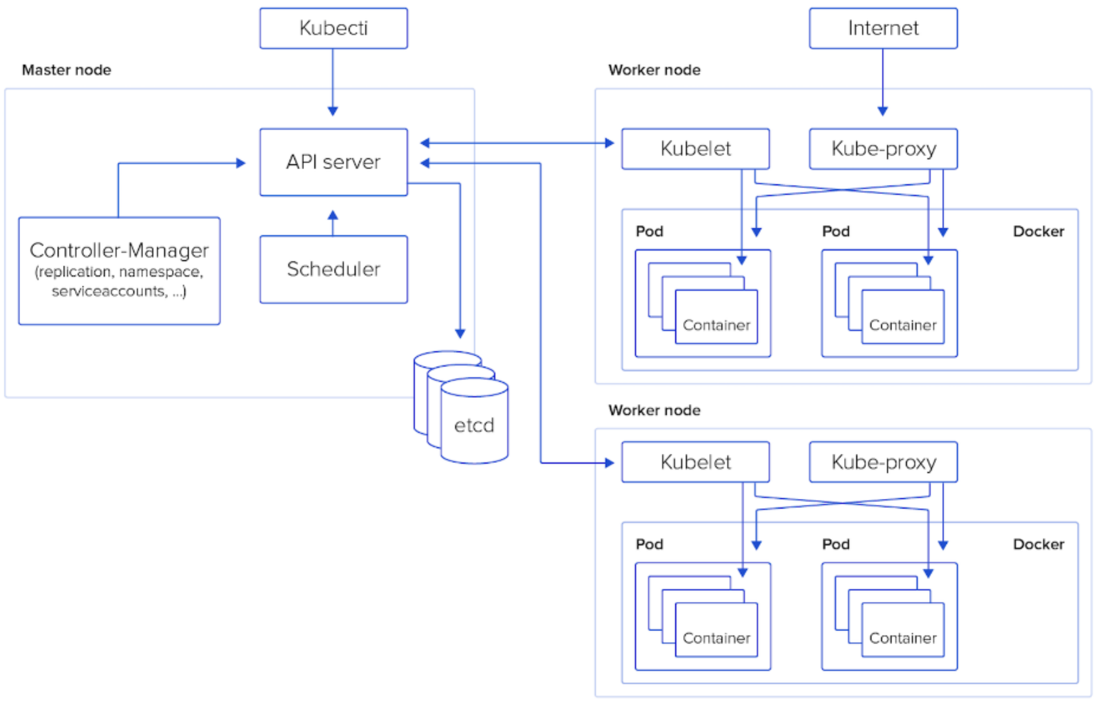


Figure 3. Kubernetes Architecture

2.4.2.1 Kubernetes Master Nodes

Kubernetes Master Node [8] is a node used as a control panel for the whole cluster and is in charge of ensuring that the Kubernetes cluster a achieves a desired state, characterized by the developers in a declarative manner. It consists of the following components:

1. **Kubernetes API Server:** is responsible for handling internal and external traffic and API calls that handle admission controls, authentication and authorization. It is the only component that interacts with the etcd database and works as the front end of the cluster's shared state.
2. **Etcd:** is the database system that's utilized for saving the cluster state, networking information and other diligent information. It stores data within the shape of key-value pairs and is consistent and highly available.
3. **Kubernetes Scheduler[4]:** is in charge of assigning which worker node will host which pod and service, using various scheduling algorithms. Firstly, it checks for availability and then based on the type of request, it determines the node available for this particular job.

[4] https://kubernetes.io/docs/concepts/scheduling-eviction/kube-scheduler/

4. **Kubernetes Controller Manager:** runs distinctive control processes. It constantly communicates with the Kubernetes API Server to ensure that everything is up and running. If the state of a component does not match the desired one, the Controller Manager contacts the necessary controller to match the desired state.

2.4.2.2 Kubernetes Worker Nodes

The Kubernetes Worker Nodes [8] is where the applications and the services are deployed and running. These nodes consist of the following components:

1. **Kubelet:** is an agent that ensures that the current state of the containers running in a Pod is the same as the desired state. It receives pod specifications from the Kubernetes Master Node and works to carry out the requested task, regarding the Kubernetes Worker Node.
2. **Kube-Proxy:** is a network proxy that maintains the necessary network rules of the cluster and manages the network requirements of the node by keeping track using an iptable.
3. **Container Runtime Engine**: is responsive for creating and running containers on the node. Docker was used in this master thesis, as a Container Runtime Engine.

2.4.3 Kubernetes Objects

As mentioned above, Kubernetes platform provides control over resources that have to do with storage and compute mechanisms. These resources are regarded as Objects and Kubernetes contain 8 Key Objects that are:

1. **Pods:** consist of one or more containers that co-exist in an environment that share network and storage resources and is created by the host system. It’s pod has a unique IP so there are no conflicts regarding the network ports and each Pod communicate with the others through a Pod

Network. In this master thesis, Flannel[5] Pod Network was deployed for the Pod communication**.**

2. **Replica Sets:** is a group of instances that are used for preserving a stable data set of running replica Pods at any given time. Its purpose is to manage the availability of identical Pods.
3. **Services:** is defined by a deployed group of pods in a Kubernetes cluster. A service provides specific functions (web services, image processing, etc.) a name and a unique cluster IP address to the group of pods that it is assigned to.
4. **Volume:** is a directory that contains ephemeral (data that gets deleted in a restart event) or persistent (data exists in the whole lifetime of a Pod) information accessible by containers inside a Pod.
5. **Namespaces:** are virtual clusters inside the Kubernetes cluster. They are used in order to organize and group when different individuals or projects coexist in the cluster.
6. **ConfigMaps and Secrets:** are used in order to store data that can't be accessed by everyone in a cluster. ConfigMaps are meant for plaint text data saving while secrets are for data that only the application needs to access and anyone else.
7. **Stateful Sets:** are used to manage every Kubernetes deployment and scale Pods and ensures the arrangement of Pod tasks and the distrinctinveness of them. Also, Stateful workloads require the state to be saved in a restart event even if the state of the application is distributed across various replicas, thing that Stateful Sets performs.
8. **Daemon Sets:** the task of scheduling the location in which the Pods will run is done via the Daemon Sets through the Kubernetes Scheduler algorithm.

[5] https://github.com/flannel-io/flannel

2.4.5 Metrics

Cloud monitoring is additionally an imperative field with respect to the operation of a cloud system. It is the method of assessing the health of the cloud-based IT infrastructures by checking the availability, the performance, the operational costs and the security behind every running application. In this master thesis, we utilized Prometheus and Grafana as cloud-monitoring tools that fetched and visualized the necessary metrics.

2.4.5.1 Prometheus

Prometheus [9] is the most popular monitoring tool across the Kubernetes platform. It is an open-source system utilized for monitoring purposes that also create notification for the developers based on time. It is used to collect and store data and metrics from different services to a unique identifier and a time stamp. Prometheus allows developers to easily query across its query metrics and provides data sets that can be easily manipulated for visualization.

Prometheus offers a unique feature called exporters, which compared to the similar monitoring technologies, exporters don't run on the host agent. Exporters are software that changes the format from third-party monitoring apps to a Prometheus one. It comes with a query language called PromQL that can be used to fetch and visualize metrics and its services run on a "/metric" endpoint. PromQL is used on a time-series database that stores all the data locally. Also, a component called "Alert Manager" can be configured that sends notifications to different notification systems like email by changing the alerting rules in the Prometheus server. The high-level architecture can be seen in the image[6] below:

[6] https://www.containiq.com/post/prometheus-metrics

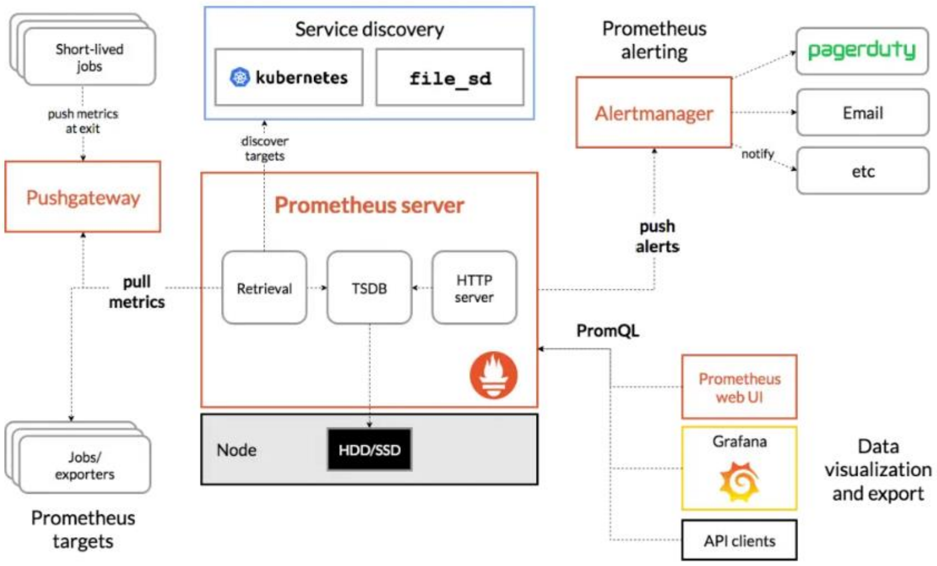


Figure 4. Prometheus Architecture

2.4.5.2 Grafana

Grafana [10] is also an open source tool utilized across Kubernetes that allows users to visualize their data via charts and graphs that are bound together into one dashboard for easier interpretation and understanding. It became a very popular tool as it is very easy to implement and supports many data sources like Prometheus, Mysql etc. Grafana offers a wide variety of dashboards and log analytics and also, queries and alerts from every information stored in the cloud infrastructure. Below, is depicted its user interface:

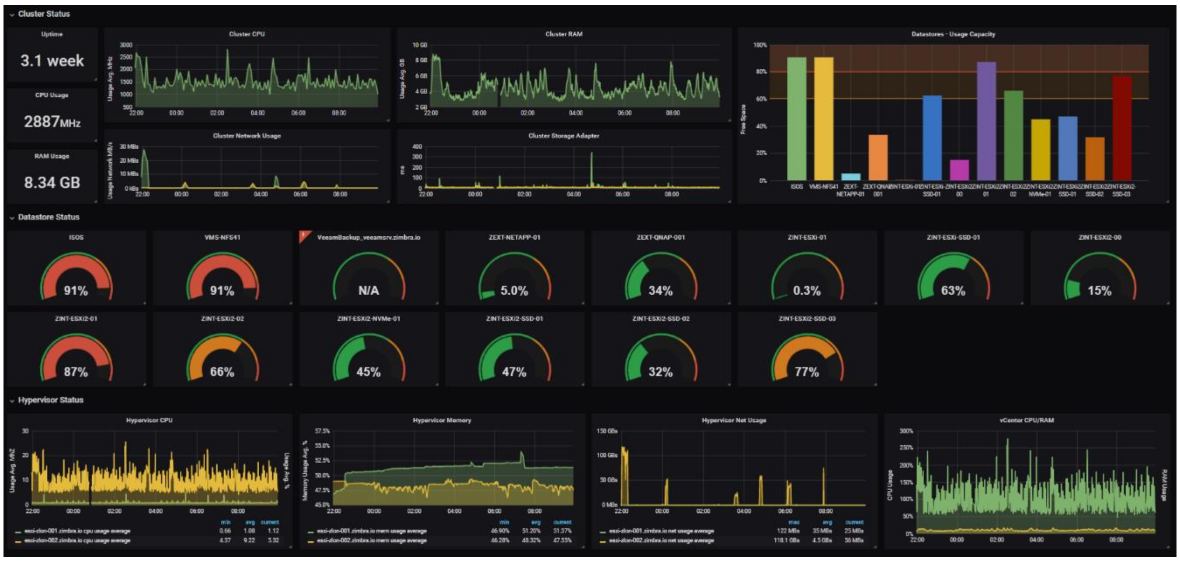


Figure 5. Grafana Dashboard User Interface

All in all, Prometheus and Grafana are two monitoring tolls that provide all the necessary information a developer can utilize to his advantage in order to maintain applications on the cloud and monitor them to keep track of their operations. In this master thesis, we implemented Prometheus as a data source for Grafana which we used to visualize our own queries.

2.5 5G Mobile Networks

2.5.1 Overview

5G [11] is the fifth generation mobile network and it is the successor of 4G. 5G transforms the entire telecommunication network, presenting numerous benefits for every user. 5G technology is meant to deliver higher multi-Gbps speeds, low latency, maximum reliability and security. Its cellular network can be divided into 2 main components: the Radio Access Network (RAN) and the Mobile Core Network. The RAN oversees the radio spectrum, making sure that everything is working fine and the QoS levels are met. On the other hand, the Core network's purpose is to connect the RAN to the IP-based internet. The following image[7] describes a basic architecture of how 5G operates:

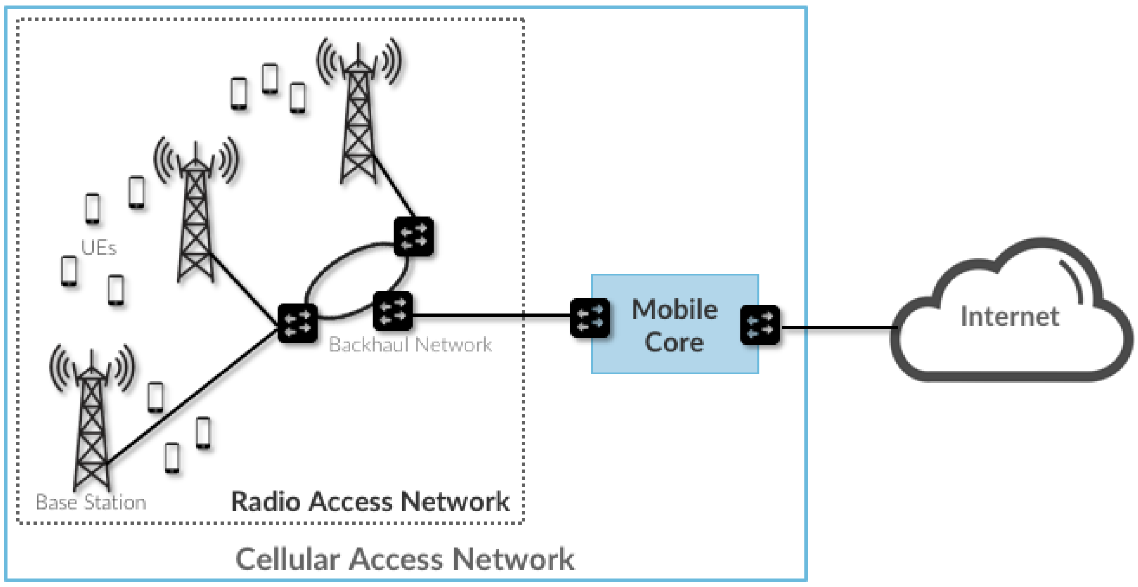


Figure 6. 5G Main Architecture

[7] https://5g.systemsapproach.org/arch.html

2.5.2 5G Radio Access Network

The 5G Radio Access Network is a major component of the 5G architecture that uses 5G radio FDD frequencies to provide wireless connectivity from the UE (User Equipment) to base stations. Below, is being described its operation with the UEs and the Mobile Core:

1. Upon detecting that a UE has not been active for a certain amount of time, each Base Station is responsible for waking up the UE that this particular subscriber has.
2. Every Base Station is used as a medium between the UE and the Core component that it communicates and transfers signals between these 2. Also, the user is tracked and authenticated.
3. For each active UE, tunnels (e.g. Voice or Streaming Multimedia) are being established with the Mobile Core component.
4. The base station is used for the communication of the Core and the UE by using a protocol named GTP.
5. Every BS is responsible for making true UE handovers with other Base Stations.
6. Whether or not a UE handover is involved, the relevant BS can make a multi-point transmission from BS to BS

We simulated the Radio Access Network using a tool called RF Simulator. The RF Simulator is an OpenAirInterface software package that replaces the radio heads such as a USRP device. It allows connecting the OAI UserEquipment (like a smartphone or a tablet) and the OAI gNodeb (a 3GPP-compliant implementation of a 5G base station) through a network interface. We implemented 3 gNodeBs with 3 UEs connected to each one, making 3 replicas of the same 5G network topology.

2.5.1 5G Mobile Core Network

Mobile Core is the heart of a 5G topology. It gives the UEs access to an external network while each user location. It ensures that every user is authenticated and the services provided work as they should do. It's the fact that oversee every subscriber and his activity that makes the Mobile Core so complex.

The 5G Mobile Core, utilizes a micro-service architecture because there has been a significant level disaggregation as there are a set of functional blocks. That said, the key components of NG-Core can be described as microservices running autonomously, as shown in the image below:

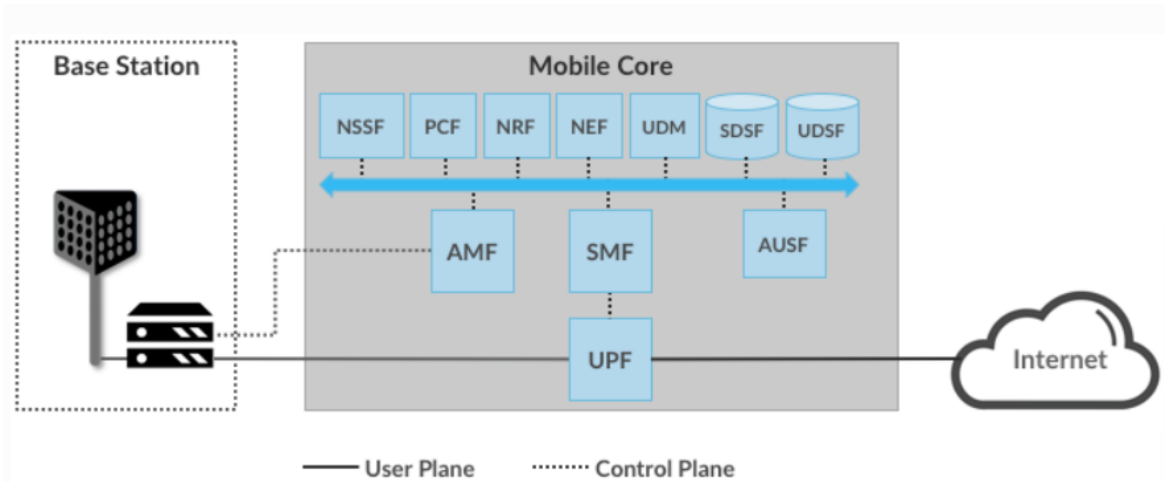


Figure 7. OAI 5G Mobile Core Components

The NG-Core can be organized into 3 groups of functions, with the first one that runs in the Control Plane and has a counterpart in the EPC and consists of the following components:

- AMF (Core Access and Mobility Management Function): in charge of services including user authentication, authorization, connection and mobility services.
- SMF (Session Management Function): oversees each UE session and selection of associated UP function.
- PCF (Policy Control Function): controls every policy that governs the CP function.
- UDM (Unified Data Management): in charge of user information and credentials.
- AUSF (Authentication Server Function): which is an authentication server.

The second group also runs on the Control Plane but it does not have a direct counterpart in the EPC and is made up from the components described below:

- SDSF (Structured Data Storage Network Function): can be characterized as an SQL database storing structured data.
- UDSF (Unstructured Data Storage Network Function): just like SDSF but with unstructured data.

- NEF (Network Exposure Function): implemented for exposing several function to select capabilities to external services.
- NRF (NF Repository Function): is a discovery service.
- NSSF (Network Slicing Selector Function): utilized in order to select a Network Slice Function for a particular UE.

Finally, the last group consists of a single piece of software called UPF (User Plane Function). It is a medium of communication between the Internet and the RAN as it forwards traffic. Also, it is in charge of keeping the policies to a standard level and monitoring the traffic and ensuring QoS standards. A representation of the NG-Core architecture with its main components is displayed in Figure 7.

In the experiments that we conducted, we implemented 3 replicas of an NG-Core network along with the corresponding RAN network, but we didn’t deploy all of these components as some of them were not necessary for the testing procedure. As it is being explained in 3.2.3, we deployed a Core network with: Mysql (Subscriber Database), AMF, SMF, UPF, NRF.

2.6 OpenAirInterface

2.6.1 Overview

OpenAirInterface (OAI) is created by EURECOM ,which is located in France. It aims to optimize mobile telecommunication mainly regarding 4G and 5G. It is an open source project and is built upon mobile software components and can be implemented, modified and tested by anyone who decides to utilize it.

2.6.2 OAI Projects

OpenAirInterface being the go-to solution for enterprises that want to implement mobile telecommunication networks, offer a variety of open source projects which are:

1. **5G RAN:** The OAI 5G RAN project has the goal of developing a 3GPP compatible 5G gNB Radio Access Network stack as an open source software for the OAI

community. One of the 2 basic component that every 5G Network utilized is called RAN. It is a medium of communication between the UEs and to the Core. Radio waves has been used in order to achieve this, transmitted packets wirelessly to the Core component destined to the Internet. It corresponds to a distributed collection of base stations, as shown in the image below, called gNodeBs.

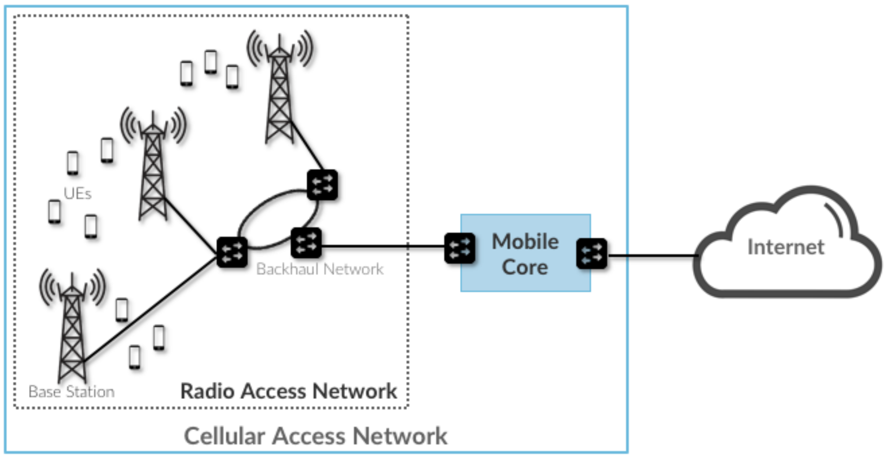


Figure 8. RAN, Core and Internet communication

2. **5G Core Network:** the second basic component is called Core Network and it follows the 3GPP standards created by EURECOM. It provides internet connectivity to every UE as shown in the image above. It ensures user authentication and SLA subscriptions. Also, it keeps track of the user mobility. The OAI 5G RAN and OAI 5G Core Network was implemented in this master thesis, in order to utilize a 5G topology as realistic as possible and use it in order to perform some tests and experiments.
3. **MOSAIC5G:** OAI's MOSAIC5G project aims to transform radio access networks into powerful network-service delivery platforms. Such a platform allows developers to investigate modern use-cases of interest to various industries. In this project, a set of extendable control and orchestration frameworks are being developed on top of the OAI 5G Core Network and OAI 5G RAN in order to offer observability and programmability of the underlying infrastructure.
4. **CI/CD:** Continuous Integration/Continuous Deployment (CI/CD) mission is to ensure high quality resources available to the community by maintaining and

improving CI/CD pipelines, developing new methods and implementing new tests to the existing projects.

Chapter 3: Development and Implementation

3.1 Chapter Introduction

In this chapter, we examine the way we implemented and developed this master thesis. How the tools analyzed in the chapter above, was set up and used to our advantage. This chapter is separated in 4 sub-chapter: the Environment Set-Up explaining how this cloud environment was created, the Scale Policy mentioning the methodology in which we scale the replicas and how the traffic was distributed across them and the System Architecture that sums up the design of the whole system we implemented.

3.2 Cloud Environment

3.2.1 Kubernetes

As we mentioned above, we used the Kubernetes orchestrator that allows scaling of the deployed network functions. However, in order to set up a Kubernetes cluster we used NITOS Testbed utilizing two of its nodes, one as a Kubernetes Master and the other as a Kubernetes Worker. After installing Docker on both of the nodes, we initialize the cluster with the help of "kubeadm" tool by running the following command:

```
$ sudo kubeadm init –pod-network-cidr=10.244.0.0/16
```

This command is used for the implementation of the Kubernetes master node and is executed only on one of the 2 NITOS nodes. The "—pod-network-cidr" flag is responsible for the IP address range specification and allocation of CIDRs for every node.

The following box contains the output of this command indicating that some more steps need to be done for the Kubernetes cluster utilization.

```
Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config

Then you can join any number of worker nodes by running the following on each as root:
kubeadm join 192.168.100.6:6443 –token 06tl4c.oqn35jyhcibz0r0m –discovery-token-ca-
cert-hash sha256:c40f5fa0aba6bb311efcb0b8cc637ae0eb8cbe27b7a03d47be6d96142f2204c
```

A pod network had to be deployed so that pods can speak with each other. Flannel pod network was being used and installed by the command below:

```
$ sudo kubectl apply -f
https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml[8]
```

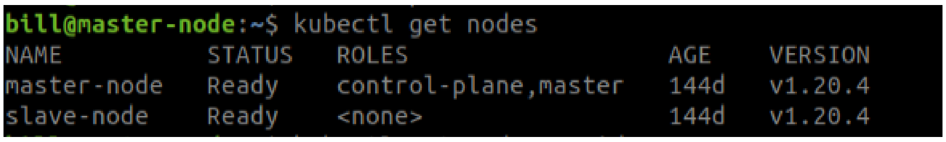
Now, the Kubernetes Worker needs to join our cluster by the following command:

```
$ sudo kubeadm join 192.168.100.6 –token 06tl4c.oqn35jyhcibz0r0m –discovery-token-
```

[8] https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml

```
ca-cert-hash
sha256:c40f5fa0aba6bb311efcb0b8cc637ae0eb8cbe27b7a03d47be6d96142f2204c
```

Inspecting the status of the cluster with kubectl command informs us that the cluster is up and running.



```
bill@master-node:~$ kubectl get nodes
NAME          STATUS   ROLES                  AGE    VERSION
master-node   Ready    control-plane,master   144d   v1.20.4
slave-node    Ready    <none>                 144d   v1.20.4
```

Figure 9. Kubernetes Nodes

3.2.2 Prometheus and Grafana

Monitoring is a crucial part of this master thesis as it helps developers to understand an application's behavior better and act responsively by scaling up or down or detecting errors and bugs. Prometheus and Grafana are the most popular tools for collecting metrics and visualizing them in meaningful charts which is why they were selected for this master thesis.

Before installing Prometheus and Grafana, a Kubernetes StorageClass is to be deployed. A StorageClass is a Kubernetes storage mechanism that let us provision persistent volumes (PV) in a cluster. So, we created a namespace with the name "local-path-provisioner" that comes with a provisioner by using the kubectl apply command:

```
$ kubectl apply -f https://github.com/rancher/local-path-provisioner/blob/master/deploy/local-path-storage.yaml
```

Firstly, we used Helm charts to install Prometheus. The following steps describes the procedure of installing Prometheus using Helm:

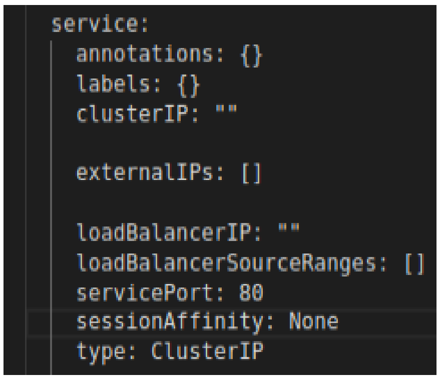
1. Attach the Prometheus repository:

```
$ helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
```

2. Save the .values file in order to modify it to our needs:

```
$ helm inspect values prometheus-community/prometheus > values/prometheus.values
```

3. Make changes to file in order to fit our needs by replacing the type field as shown below:

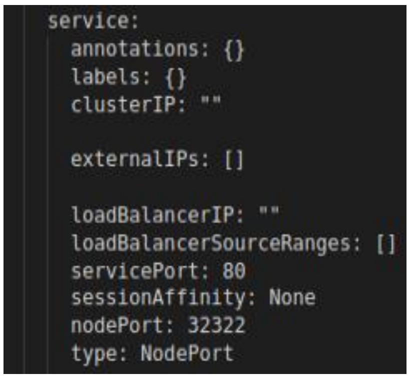


```
service:
  annotations: {}
  labels: {}
  clusterIP: ""

  externalIPs: []

  loadBalancerIP: ""
  loadBalancerSourceRanges: []
  servicePort: 80
  sessionAffinity: None
  type: ClusterIP
```

Figure 10. Inspecting .values file



```
service:
  annotations: {}
  labels: {}
  clusterIP: ""

  externalIPs: []

  loadBalancerIP: ""
  loadBalancerSourceRanges: []
  servicePort: 80
  sessionAffinity: None
  nodePort: 32322
  type: NodePort
```

Figure 11. Changing the .values file

Prometheus is being exported out of the Kubernetes cluster in order to be reachable from a machine outside of the cluster.

4. Execute this command in order to get the Prometheus User Interface running on our local host. The 8888 port matches our local port, the 10.0.1.89 matches the NITOS Node port and the 32322 is the port that the Prometheus service is running.

```
$ ssh -L 8888:10.0.1.89:32322 vzalokost@nitlab3.inf.uth.gr
```

Checking if everything goes as planned:

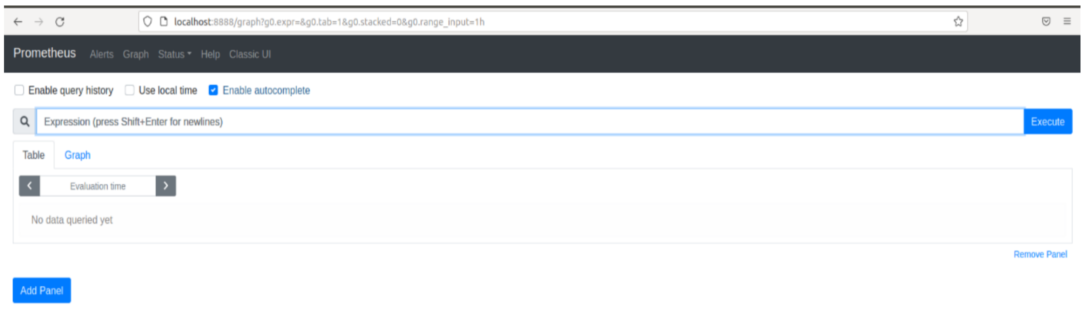
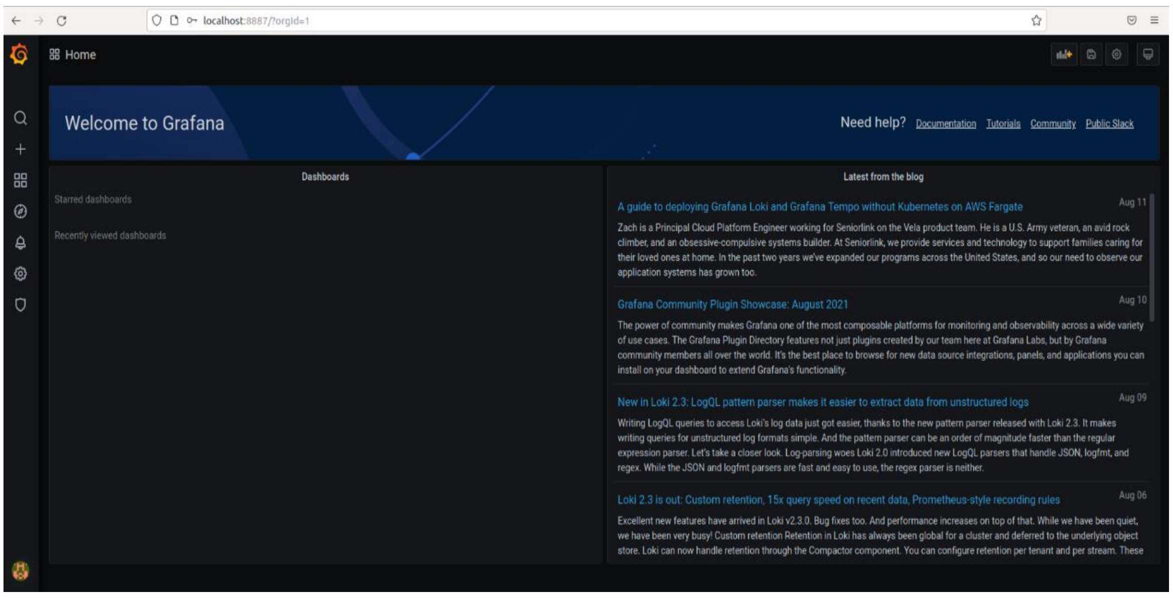


Figure 12. Prometheus UE

The same steps has been followed to install Grafana by downloading the appropriate .values file, changing the type to NodePort and the nodePort and then deploying the changes and using the kubectl apply command.

This way the UE pops up by typing localhost:8887 to our browser as shown below:



A guide to deploying Grafana Loki and Grafana Tempo without Kubernetes on AWS Fargate — Aug 11

Zach is a Principal Cloud Platform Engineer working for Seniorlink on the Vela product team. He is a U.S. Army veteran, an avid rock climber, and an obsessive-compulsive systems builder. At Seniorlink, we provide services and technology to support families caring for their loved ones at home. In the past two years we've expanded our programs across the United States, and so our need to observe our application systems has grown too.

Grafana Community Plugin Showcase: August 2021 — Aug 10

The power of community makes Grafana one of the most composable platforms for monitoring and observability across a wide variety of use cases. The Grafana Plugin Directory features not just plugins created by our team here at Grafana Labs, but by Grafana community members all over the world. It's the best place to browse for new data source integrations, panels, and applications you can install on your dashboard to extend Grafana's functionality.

New in Loki 2.3: LogQL pattern parser makes it easier to extract data from unstructured logs — Aug 09

Writing LogQL queries to access Loki's log data just got easier, thanks to the new pattern parser released with Loki 2.3. It makes writing queries for unstructured log formats simple. And the pattern parser can be an order of magnitude faster than the regular expression parser. Let's take a closer look. Log-parsing woes Loki 2.0 introduced new LogQL parsers that handle JSON, logfmt, and regex. While the JSON and logfmt parsers are fast and easy to use, the regex parser is neither.

Loki 2.3 is out: Custom retention, 15x query speed on recent data, Prometheus-style recording rules — Aug 06

Excellent new features have arrived in Loki v2.3.0. Bug fixes too. And performance increases on top of that. While we have been quiet, we have been very busy! Custom retention Retention in Loki has always been global for a cluster and deferred to the underlying object store. Loki can now handle retention through the Compactor component. You can configure retention per tenant and per stream. These

Figure 13. Grafana UE

3.2.3 OpenAirInterface

Having the Kubernetes cluster up and running and our monitoring services ready, we utilized a 5G deployment that every 5G component run in each own pod, using Helm charts. We deployed a basic setting of OAI 5G Core network and oai-gNB and oai-nr-ue in rf-simulator mode in order to perform some traffic testing [13]. At first, we cloned the git repository in which the configuration are saved with the following command:

```
$ git clone -b master https://gitlab.eurecom.fr/oai/cn5g/oai-cn5g-fed
```

Helm spray was also used, which is a helm plugin, that manages the dependencies of the network functions which are required to be deployed in a certain order. We provided helm charts for individual network functions regarding the Mysql, AMF, SMF, UPF and NRF.

After transitioning to the appropriate folder with the following command:

```
$ cd charts/oai-5g-core/oai-5g-basic
```

we used helm spray command to deploy the necessary helm charts of the 5G Core Network:

```
$ helm spray .
```

After successfully deploying the OAI 5G Core Network, we had to check if it was properly configured using kubectl commands as described below:

```
$ kubectl logs -c spgwu $SPGWU_TINY_POD_NAME | grep 'Received SX HEARTBEAT
REQUEST' | wc -l
```

```
$ kubectl logs -c smf $SMF_POD_NAME | grep 'handle_receive(16 bytes)' | wc -l
```

The value returned from these 2 commands is more than 1 indicating that SMF and UPF have successfully registered to NRF and there is a PFCP session.

In order to perform some traffic testing we also deployed oai-gNB and oai-nr-ue. At first, we had to transition to the oai-gnb folder and configure the ip-address of the AMF component in order to communicate with each other running the following commands, with the last one used for checking the connectivity of the gNB with the AMF component:

```
$ cd ../../oai-5g-ran/
$ sed -i 's/amfIpAddress: "172.17.0.8"/amfIpAddress: '"$AMF_eth0_POD_IP"'/g' oai-gnb/values.yaml
$ helm install gnb oai-gnb
$ kubectl logs -c amf $AMF_POD_NAME | grep 'Sending NG_SETUP_RESPONSE Ok'
```

Similarly to the deployment procedure of the oai-gnb, we also deployed oai-nr-ue simulator. We provided the ip-address of gnb rf-sim and used helm to install it as described below:

```
$ sed -i 's/rfSimulator: "172.17.0.9"/rfSimulator: '"$GNB_eth0_IP"'/g' oai-nr-ue/values.yaml
$ helm install nrue oai-nr-ue/
```

Now, we can finally ping successfully the Internet through the tunnel interface with name “oaitun-ue1”, that sends the traffic across the OAI 5G Core Network with “google.gr” domain as its target, as displayed below:

```
$: kubectl exec -it -c nr-ue $NR_UE_POD_NAME -- /bin/bash
$ ping -I oaitun_ue1 -c4 google.fr
PING google.fr (172.217.19.131) 56(84) bytes of data.
64 bytes from par03s12-in-f131.1e100.net (172.217.19.131): icmp_seq=1
ttl=116 time=27.4 ms
64 bytes from par03s12-in-f131.1e100.net (172.217.19.131): icmp_seq=2
ttl=116 time=12.5 ms
64 bytes from par03s12-in-f131.1e100.net (172.217.19.131): icmp_seq=3
ttl=116 time=16.5 ms
64 bytes from par03s12-in-f131.1e100.net (172.217.19.131): icmp_seq=4
ttl=116 time=29.0 ms

--- google.fr ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3003ms
rtt min/avg/max/mdev = 12.513/21.352/29.042/7.031 ms
```

3.3 System Architecture

Having analyzed the tools that make up the cloud environment we implemented, the architecture of this master thesis can be properly explained. As explained in 3.2 section, we defined a 5G OAI Core Network with the following components: Mysql, AMF, SMF, UPF and NRF. Additional components could be added such as UDR, UDM and AUSF but for the nature of our experiments they were not necessary. Also, for testing purposes in order to generate traffic through the Mobile Core we utilized, we deployed oai-gnb component to simulate a Base Station as explained in 2.5 section for the appropriate User Equipment (UE) to connect which can be a tablet, a phone, a computer etc. To simulate a User Equipment device we deployed oai-nr-ue module. As explained in the chapters above, we had scaled out 3 replicas of an OAI 5G Core Network with the respective gNBs and UEs, as it can be seen in the image below:

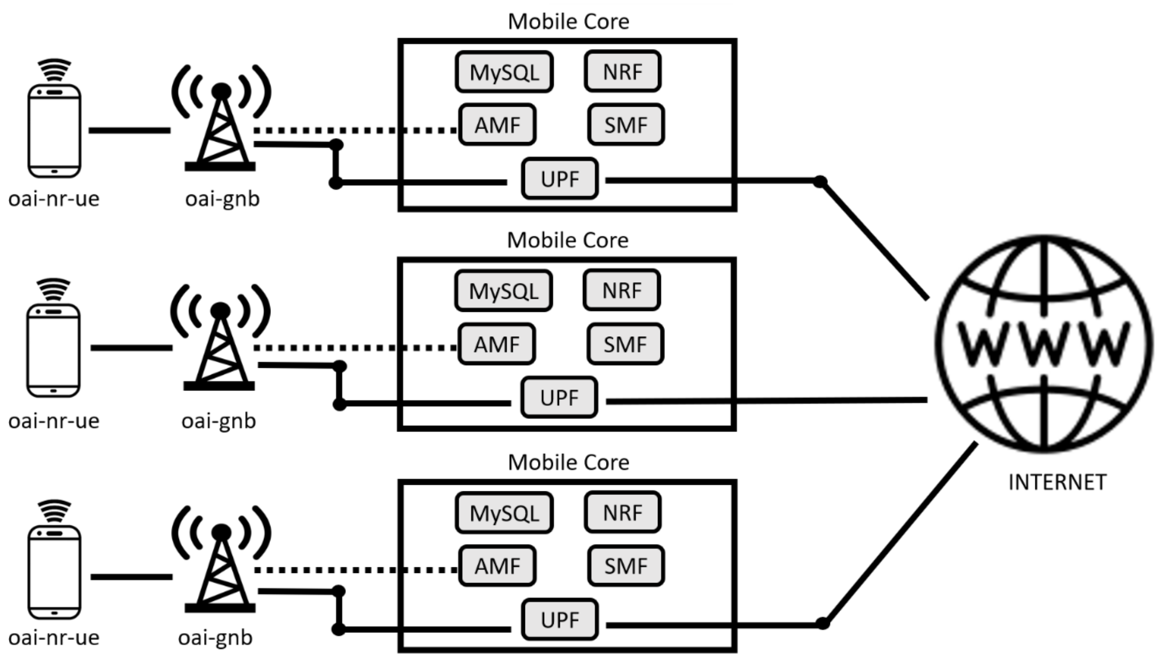


Figure 14. Master Thesis Architecture

Each UE pings a website requesting and sending packets through the Mobile Core Network. A mechanism is implemented on top of them that monitors the total web traffic that passes through every Mobile Core and distributes it accordingly. In that way, the traffic is spread equally across every replica so that there is not a Mobile Core component that overworks while the others are idle as carry out the necessary tasks accordingly.

3.4 Traffic Distribution Policy

Last but not least, the way that the web traffic is distributed across all the OAI 5G Mobile Core Network deployment is to be analyzed. We calculated the maximum traffic threshold that one of these deployment can carry out by stress testing the whole deployment, making numerous web requests and monitoring it in order to check when it will crash. We figured out that the maximum value was 150 of the following metric:

```
sum(rate(container_network_receive_bytes_total{pod=~'oai-spgwu-tiny.+'}[10m])
```

Separating this metric into 4 parts, will help to better understand it:

1. **container_network_receive_bytes_total =** is a cumulative count of bytes that a Pod received.
2. **container_network_receive_bytes_total{pod=~'oai-spgwu-tiny.+'} =** is a filter that we used in order to get the cumulative count of bytes that the Pods with the substring “oai-spgwu-tiny” in their name receive.
3. **rate(container_network_receive_bytes_total{pod=~'oai-spgwu-tiny.+'}[10m]) =** returns the cumulative rate of bytes received in the SPGWU (UPF) pods across all replicas as measured in the last 10 minutes.
4. **sum(rate(container_network_receive_bytes_total{pod=~'oai-spgwu-tiny.+'}[10m]) =** returns the sum rate of cumulative bytes received across all replicas for 10 minutes.

So, now that we know the container web traffic limits, we can use it as a threshold for our traffic distributing problem. Every time the value of this threshold is more than 150, we allocate a new replica and use it in order to balance the load between them. The web traffic is being distributed across the available replicas so that more tasks can be carried out with more replicas to be working simultaneously.

Chapter 4: Experiments and Comparisons

4.1 Chapter Introduction

This chapter comprises of the experiments that were carried out in order to test the performance of the OAI 5G Core Network deployment and its other 2 replicas. We generate traffic from the UEs pod that was propelled through each Core Network and take some measurements to conclude if indeed our implementation really benefits us in any way and if so, how.

4.2 Experiments Description

In order to measure the efficacy of our implementation, we conducted some experiments. The scenario we followed is the following: after defining proactively three identical deployments with the 5G Mobile Core, oai-gnb and oai-nr-ue components properly implemented as descripted in Chapter 3, we created a script inside the UE pod. This python script generates traffic making web requests using the ping tool from google.gr domain. The web traffic that each UE is ordered to carry out, is propelled through each corresponding Core Network destined to reach the Internet and back again. The way that each UE is being activated to send web traffic and how many requests or deactivated because of inactivity, is being done through a mechanism that we created. This mechanism reads the input web requests that is there to be done and distributes the traffic accordingly to the 3 available replicas, as shown in the image below.

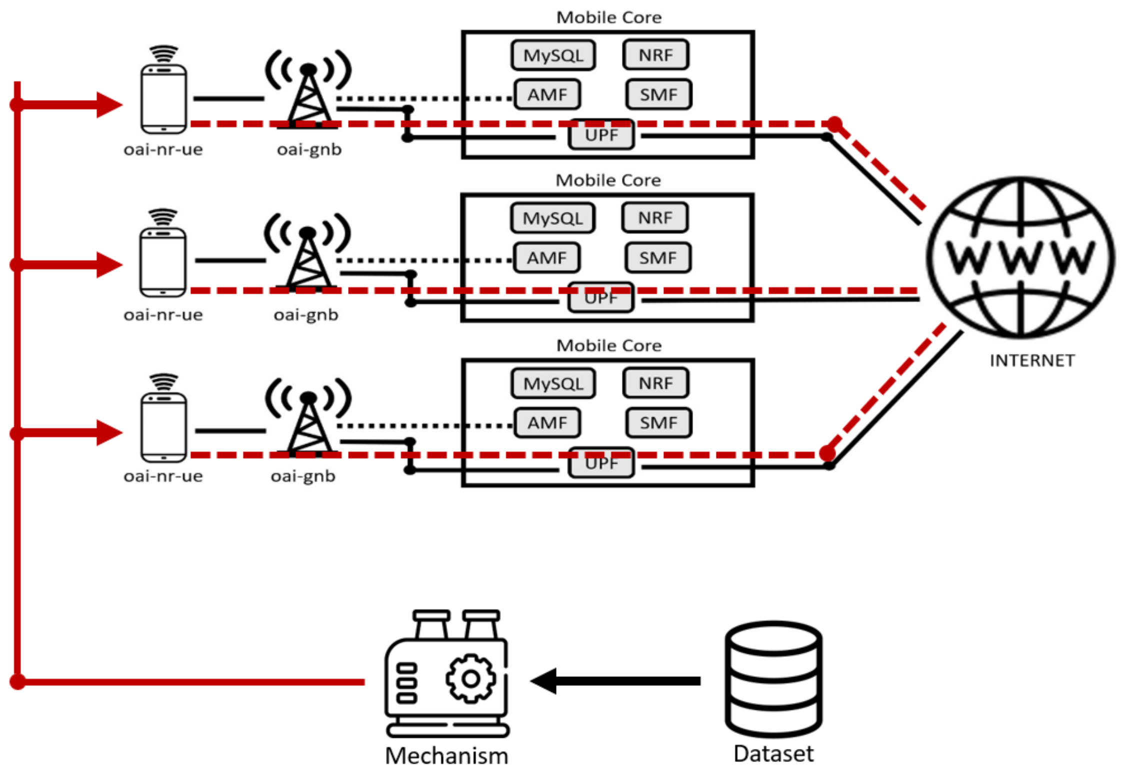
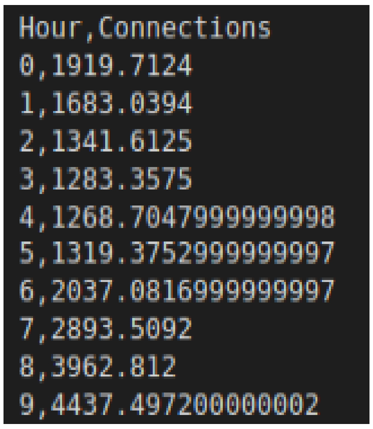


Figure 15. Master Thesis Implementation Design

The implemented mechanism runs a Python script that reads a number of web requests in its iteration and calculates the best number of deployments that can carry out this work. Then, we sends a command to the corresponding UEs to start producing web traffic through each Mobile Core Network to the Internet. With such an experiment we can measure the performance of our mechanism and take metrics regarding the total internet traffic that pass through every deployment and compare it when one and two deployments are only available.

4.3 Dataset

Regarding the dataset [14] that we worked on, it issues info about the number of SMS and calls a Base Station has received which is located in Milan, Italy and its format depicted below:



```
Hour,Connections
0,1919.7124
1,1683.0394
2,1341.6125
3,1283.3575
4,1268.7047999999998
5,1319.3752999999997
6,2037.0816999999997
7,2893.5092
8,3962.812
9,4437.497200000002
```

Figure 16. Dataset Format

We round the connections number and make a plot that constitute our input dataset where the Y-axis represents the number of connection to the base station while the X-axis the number of hours that passed:

4.4 Experimental Results

Having utilized our cloud environment correctly and the experiment that described above, the evaluation charts are to be displayed. In order to check if we achieve better performance by proactively deploying 3 replicas of a 5G Mobile Core Network along with the corresponding UEs and gNBs, we created some graphs that show the amount of web traffic that passed through all Mobile Cores when we are using 1 replica, 2 replicas and finally, 3 replicas.

The metric that we are harvesting from the cloud environment we created, using the Prometheus tool, is the one described at the 3.4 section that represents the sum rate of cumulative bytes received across all 5G Mobile Core deployments for 10 minutes.

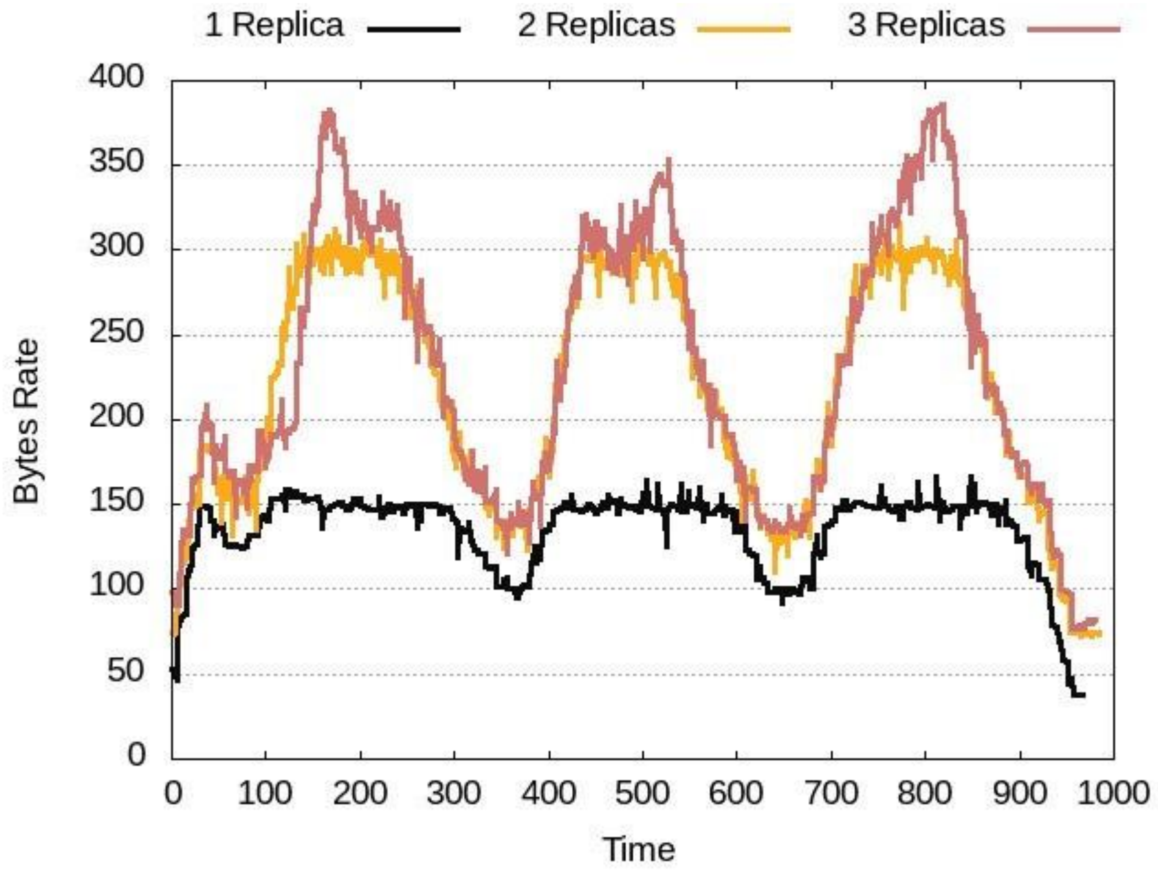


Figure 17. Bytes Rate per Time for every replica

In the chart above, the sum rate of cumulative bytes that are passing through the 5G Mobile Core Network for one replica is with the black line, for 2 replicas is with the cyan line and for 3 replicas are with the red line. As we can see, having more replicas of the 5G Core deployment along with the corresponding UEs and gNBs, implies more bytes to be transmitted through the Core Network in total. This happens because every replica has a maximum number of bytes that it can carry out before it crashes.

Chapter 5: Conclusion and Future Work

5.1 Chapter Introduction

The conclusion and the future work are yet to be presented. A brief summary of this master thesis is being debated and some ideas for extending this work even further.

5.2 Conclusion

Last but not least, the central thought of this master thesis is to utilize a mechanism that distributes web traffic to 3 replicas of a 5G Mobile Core Network deployment, along with the corresponding UEs and gNBs, accordingly. This way better resource utilization is being achieved as the traffic is being distributed to every replica equally. There are no compute resources that are overworking to carry out the necessary tasks while some others are idle. From Chapter 4, we also conclude that more web traffic can be handled out in a better way.

5.3 Future Work

Even though this master thesis's objective is thought to have been met, there are certain upgrades that can be made. Firstly, the replicas of the OAI 5G Mobile Core deployments can be created and destroyed dynamically as for now they are deployed proactively. This means that they are created before the experiments start to roll and they are not getting destroyed. They are not used when not necessary, they just exist and wait for when the mechanism wakes them up to handle some web traffic out. With this improvement, more energy can be saved as there are no resources that are idle. For further extending this master thesis, a real-world experiment can be conducted with the appropriate hardware components in order to show that scaling an environment like that can only be beneficial. For such an experiment, high end PCs with compatible RF front-ends (e.g. USRP N310s, USRP X310s) can be used in order to form fully operational 5G Networks. Moreover, 5G end user UEs are needed, e.g. the Quectel RMQ500 that is compatible with the OpenAirInterface 5G implementation. Finally, different input dataset can be used and more metrics can be exported in order to prove this work's importance such as CPU and RAM Utilization.

Bibliography

[1] Simplilearn. (2023). *What is Cloud Computing and the Top Cloud Technologies to Look Out*. https://www.simplilearn.com/cloud-technologies-article

[2] Wikipedia. (2023). *5G*. https://en.wikipedia.org/wiki/5G

[3] NITLAB. (2015). *NITOS Wireless Testbed*. http://nitlab.inf.uth.gr/NITlab_old/index.php/testbed

[4] Docker Docs. (2013). *Docker*. https://docs.docker.com/get-started/overview/

[5] Edureka. (2013). *What Is Docker & Docker Container?*. https://www.edureka.co/blog/what-is-docker-container

[6] Kubernetes. (2014). *Production-Grade Container Orchestration*. https://kubernetes.io/

[7] Toptal. (2021). *What is Kubernetes?* . https://www.toptal.com/kubernetes/what-is-kubernetes

[8] Kubernetes. (2021). *Kubernetes Components*. https://kubernetes.io/docs/concepts/overview/components/

[9] Devopscube. (2016). *How to setup Prometheus Monitoring on Kubernetes Cluster*. https://devopscube.com/setup-prometheus-monitoring-on-kubernetes/

[10] RedHat. (2021). *What is Grafana?*. https://www.redhat.com/en/topics/data-services/what-is-grafana

[11] Private 5G: A system approach. (2022). *5G*. https://5g.systemsapproach.org/arch.html

[12] Openairinterface. (2023). *Projects*. https://openairinterface.org/projects/

[13] Eurecom. (2021). *oai-cn5g-fed/deploy_sa5g_hc.md*. https://gitlab.eurecom.fr/oai/cn5g/oai-cn5g-fed/-/blob/master/docs/DEPLOY_SA5G_HC.md

[14] Barlacchi, G., De Nadai, M., Larcher, R. *et al.* A multi-source of urban life in the city of Milan and the Province of Trentino. *Sci Data* **2**, 150055 (2015). https://doi.org/10.1038/sdata.2015.55.