SYSC5804 Advanced Topics in Communications Systems
COMP 5900 Selected Topics in Computer Science
ITEC 5910W Selected Topics in Network Technologies
Carleton University

DOCUMENTATION REPORT

Project: Using container orchestration tools to deploy 5G Network on cloud and local VMs.

Contributors:

Kamal Isleem
Edwin Omoigui
Samuel Hanson Hagan
Muhammad Shafayat Oshman

https://github.com/Edwin-programmer/Project5G-ansible-deployment

Table of contents

List of Figures

LIST OF ABBREVIATIONS

5G – Fifth Generation

5GC – Fifth Generation Core Network

AMF – Access and Mobility Management Function

CN – Core Network

CU – Centralized Unit

DU – Distributed Unit

ETCD - /etc distributed

gNB – Next Generation NodeB

NAT – Network Address Translation

NMA - Network Management Automation

NRF - Network Repository Function

NSSF - Network Slice Selection Function

OAI – Open Air Interface

PCF – Policy Control Function

RAN – Radio Access Network

SMF – Session Management Function

UDM – Unified Data Management

UE – User Equipment

UPF – User plane Function

WAN – Wide Area Networking

Abstract

The introduction of 5G and its deployment comes with many challenges. One of such challenges is the huge number of heterogeneous core nodes that have been introduced and the complexities of configuring each of them according to their vendor specifications. To get the full benefits of 5G, mobile service providers will need to move to a distributed edge cloud that employs virtualization and automation technology to support 5G-based applications and services. In this project, we demonstrate how mobile network operators can deploy full 5G networks using (open source) Ansible and container orchestration tools such as docker and Kubernetes on the cloud and a local setup. The setup produced a functioning Kubernetes 5G standalone and Docker 5G Non-standalone network with all the main core elements, the Radio Access Network (RAN), and a functioning User Equipment (UE). The UE was able to ping all the associated core elements and the internet while all the core elements were able to ping each other. A test call was successful using customized open source free5GC network images. Finally, with ansible and container orchestration tools, we illustrate the deployment setup time can be reduced significantly and completed within an hour.

Keywords: *Ansible, Kubernetes, Docker, 5G, RAN, Deployment*

I. Introduction

The introduction of 5G and its deployment comes with many challenges. One of such challenges is the huge number of heterogeneous core nodes that have been introduced and the complexities of configuring each of them according to their vendor specifications. To get the full benefits of 5G, mobile service providers will need to move to a distributed edge cloud that employs virtualization and automation technology to support 5G-based applications and services. Most existing tools for service orchestration are traditionally designed for physical infrastructure and assume the existence of an operating system installed in the computers of the physical infrastructure.

This project will deploy a mixture of Ansible, Docker, and containers to manage the nodes. Ansible is the simplest way to deploy your applications and can be said to be a radically simple IT automation engine that automates cloud provisioning, configuration management, application deployment, intra-service orchestration, and many other IT needs. Ansible allows you to write 'Playbooks' that are descriptions of the desired state of your systems, which are usually kept in source control. Ansible then does the hard work of getting your systems to that state no matter what state they are currently in. Playbooks make your installations, upgrades, and day-to-day management repeatable and reliable.

Docker on the other hand is a set of platform-as-a-service products that use OS-level virtualization to deliver software in packages called containers. Containers are isolated from one another and bundle their software, libraries, and configuration files; they can communicate with each other through well-defined channels.

5G Core Network Architecture

The new 5G network introduces network functions in the architecture. As per 3GPP, are imagined and represented as “reference point representation” and “service-based representation”. Figure 1 and 2 shows the graphical representation of a reference point representation and a service-based representation respectively. The reference points describe the separation of control plane and user plane functions while the service-based representation specifies a set of Network Function (NF) and a common bus that connects these Network Functions. Additionally, service-based architecture applies to the control plane section of the 5G Core Network only. [11].

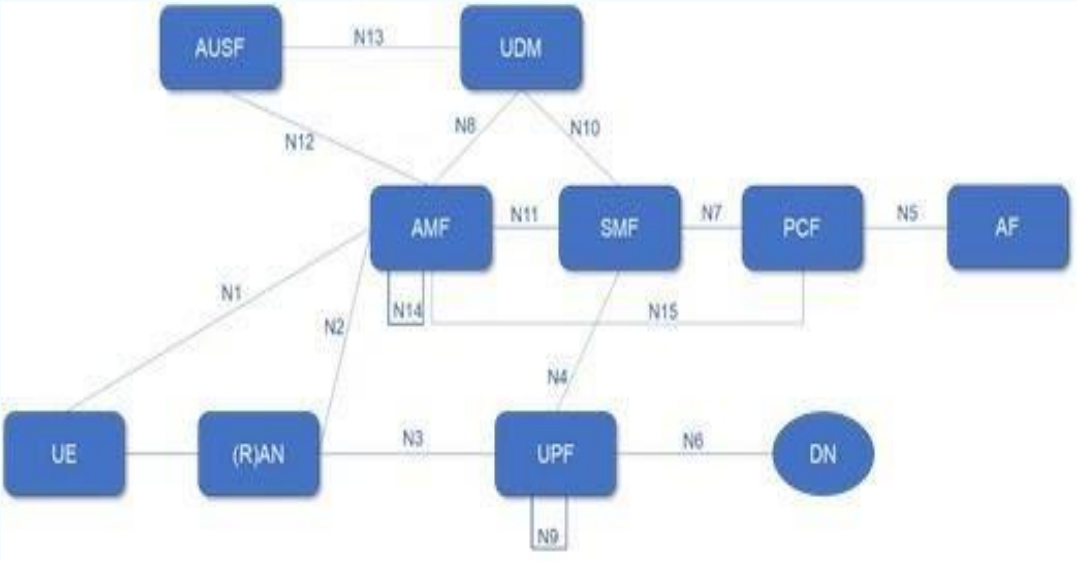

Figure 1 Non-Roaming 5G System Architecture in reference point representation [10]

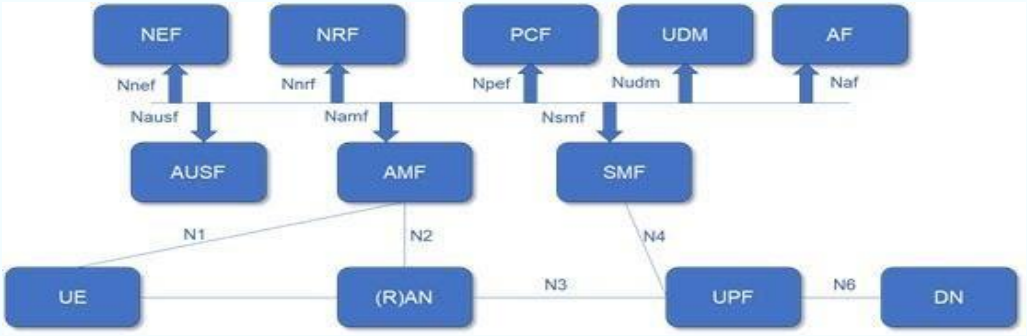

Figure. 2 — 5G System Service-based architecture [10]

The planned 5G System architecture (see figure 3) consists of the following main NFs, which constitute the 5GC:

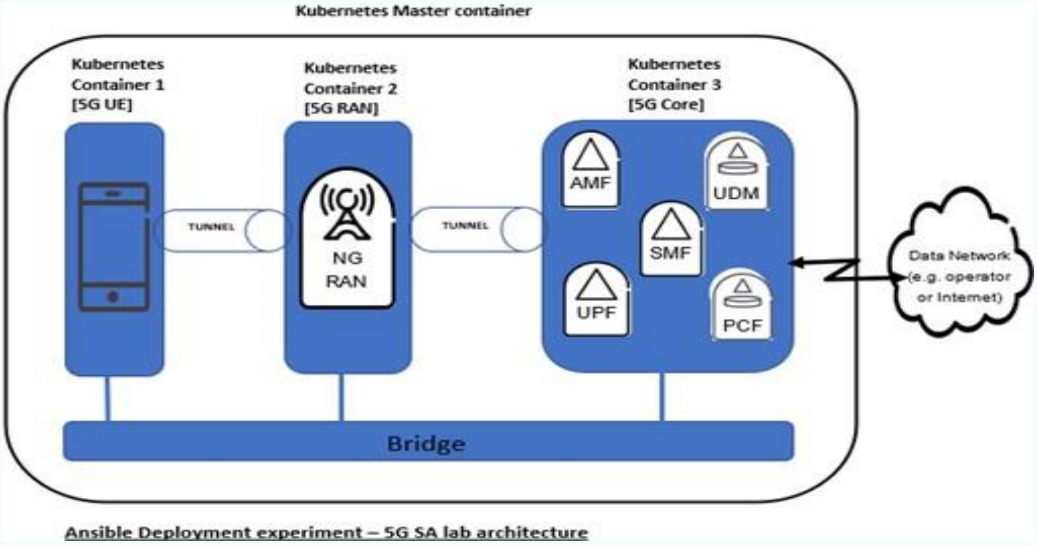

Figure 3: Planned 5G architecture (source: Self)

Access and Mobility Management Function (AMF): The AMF supports termination of NAS signaling, NAS ciphering and integrity protection, registration management, connection management, mobility management, access authentication and authorization, security context management. [11].

Session Management Function (SMF): The SMF supports session management, User Equipment (UE) IP address allocation & management, DHCP functions, termination of NAS signaling related to session management, DL data notification, traffic steering configuration for UPF for proper traffic routing. [11].

User plane function (UPF): UPF supports packet routing and forwarding, packet inspection, QoS handling, acts as an external PDU session point of interconnecting to Data Network (DN) and is an anchor point for intra- & inter-RAT mobility. [11].

Policy Control Function (PCF): The PCF supports a unified policy framework, providing policy rules to CP functions, access to subscription information for policy decisions in UDR. [11].

Unified Data Management (UDM): The UDM supports the generation of Authentication and Key Agreement (AKA) credentials, user identification handling, access authorization, subscription management. [11].

The main contributions of this research work are summarized as follows:

- To address complex networks, automation is required. Using Ansible and container orchestration provides large productivity gains to a wide variety of automation challenges and can perform the orchestration of complex multi-vendor deployments.
- We aim to demonstrate the simplification of 5G deployment, management, and operations of stateful applications in Kubernetes container clusters. Using virtualization to implement network element functions, we look at establishing a master Kubernetes and three container nodes. The 5G core solution that will be

used is free5gc (version 3.0.1) which is open-source software. We will place the core elements within container nodes running on the master Kubernetes.

- A playbook that houses all the details of the containers and master node would also contain the configuration for all the core and radio access nodes. Once the playbook is deployed, it sends configuration to all the 5G core nodes in the network at the same time. A configuration engineer would therefore not need to move from one node to the other, either on the operator's network management system or any platform to configure the core nodes.
- Using network function virtualization resolves issues related to time-wasting and configuration complexities, but instead enables enhanced performance and scalability of the 5G architecture.
- It also demonstrates that if a large number of computing nodes are required, only a few instructions are added to the Kubernetes cluster to scale the deployment and services to balance the load of the internal VNF on the computing nodes.

The rest of the paper is organized as follows: Section 2 provides details of related works done in the field of Ansible, Kubernetes, and 5G deployment. Section 3 details the proposed solution and the adopted methodology in solving the challenges that arise with the deployment of 5G. Section 4 shows the setup for the experiment. The entire deployment details including the technical deployment are documented in this section. Section 5 presents the results of a successful deployment of the experiment. Section 6 provides a conclusion for the entire report.

II. Related Works

Some previous works have been done in the field of Ansible Deployment and 5G Automation. Arouk et al. [1] demonstrated network automation using Kubernetes as container orchestration and automating application deployment, while using Openshift Operator as a tool to manage complex services, such as 5G services. They used the containerized OpenAirInterface (OAI) to deploy the network and demonstrate the automatability, such as the dynamic switch of RAN between a monolithic base station and disaggregated RAN (i.e. Distributed Unit-DU and Centralized Unit-CU), and auto-configuration. Mwanje et al. [2] analyzed the limitations of legacy Self-Organizing Networks (SON) introduced with 4G for Network Management Automation (NMA), designed to advance operability levels for mobile network operators (MNOs). They highlighted the new challenges emerging from these 5G features and presented several technological enablers that can be leveraged for 5G NMA. They included a future NMA framework combining these enablers, that addresses the NMA challenges by 1) enhancing the level of intelligence in network elements, 2) allowing the networks to better process and analyze various kinds of network information, 3) enabling identification of operational context and scenarios, and 4) leveraging the building and sharing of NM knowledge across different domains and networks. Kafle et al. [3] introduced 5G network slices (from the point of view of the non-wireless part of the network) and elaborated the necessity of automation of network functions related to the design, construction, deployment, operation, control, and management of network slice, revisiting machine learning techniques applicable to the automation of network functions and finally, presented a machine-learning-based framework for the operation and control of network

slices by continuously monitoring workload, performance, and resource utilization, and dynamically adjusting the resources allocated to network slices.

Van de Meer et al. [4] identified some of the places where automation of 5G networks is not just helpful but is required for 5G to become a reality and unearthed a conceptual approach for modeling and achieving autonomic operations and management in 5G networks positioning modern policy-based management as a key enabler for autonomic 5G network management. Huang et al. [5] investigated how OAI can be used on top of M-CORD, and presented the deployment results demonstrating a virtualized 5G Radio Access Network (RAN)-Core infrastructure on top of M-CORD. Currently, both Mosaic5G and MCORD utilize OAI and the cloud RAN idea to virtualize some RAN functions and core network functions. Chirivella-Perez et al. [6] addressed the challenge of 5g service deployment time by designing and prototyping a novel 5G service deployment orchestration architecture that is capable of automating and coordinating a series of complicated operations across physical infrastructure, virtual infrastructure, and service layers over a distributed mobile edge computing paradigm. Wiranata et al. [7] proposed 5G network automation using Kubernetes for automating application deployment while using Open-shift Operator as a tool to manage 5G services, as it allows the deployment of all Mosaic 5G inside pods. Nakimuli et al. [8] proposed the 5G EVE architecture, all the steps required to design and execute a simple experiment. 5G EVE offers an accessible and fully operational 5G end-to-end infrastructure, flexible enough to support a plethora of tests for different vertical industries, distributed among several sites. When executing the declared tests, vertical industries with expertise deploying such services may state the operational key performance indicators, and the platform will provide an analysis of

the results. To handle cloud-native deployment challenges such as the coexistence of physical and virtual functions, (near) real-time resource provisioning, service continuity, and strict latency and data rates, Arouk et al. [9] proposed Kube5G, as a realizable agile service platform in support of 4G/5G cloud-native applications. Kube5G introduces a novel approach in building and packaging a cloud-native compliant telco network function (NF) in a form of nested well-defined layers.

III. Proposed Solution

To address complex networks, automation is required. Using Ansible and container orchestration provides large productivity gains to a wide variety of automation challenges and can perform the orchestration of complex multi-vendor deployments. We aim to demonstrate the simplification of 5G deployment, management, and operations of stateful applications in Kubernetes container clusters. Using virtualization to implement network element functions, we look at establishing a master Kubernetes and three container nodes. The 5G core solution that will be used is free5gc (version 3.0.1) which is open-source software. We will place the core elements within container nodes running on the master Kubernetes. A playbook that houses all the details of the containers and master node would also contain the configuration for all the core and radio access nodes. Once the playbook is deployed, it sends configuration to all the 5G core nodes in the network at the same time. A configuration engineer would therefore not need to move from one node to the other, either on the operator's network management system or any platform to configure the core nodes. This system resolves issues related to time-wasting and

configuration complexities, but instead enables enhanced performance and scalability of the 5G architecture. It also demonstrates that if a large number of computing nodes are required, only a few instructions are added to the Kubernetes cluster to scale the deployment and services to balance the load of the internal VNF on the computing nodes.

IV. Deployment

We approach deployment using Kubernetes on the cloud and Docker on a local setup.

Kubernetes setup (Cloud deployment)

In this setup, the UE or gNB simulator will request the AMF to establish a session. Once this happens, the AMF in turn will send a request to the network slice selection function (NSSF) to know which slice to use.

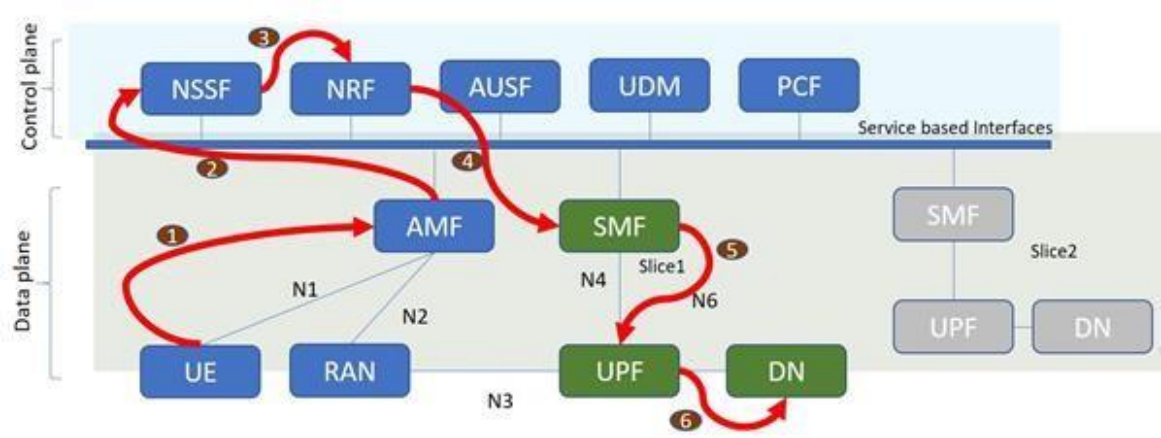

Figure 4: Session setup and service flow (Source: Self)

For this test deployment, we only use one network slice. This request will trigger the network repository function (NRF) to locate the SMF in proxy which will establish a connection to the UPF and then to the data network/internet. The UPF carries user data packets between the gNB and the external WAN and also connects back to the SMF. This process is shown in Figure 4 above. With 5G as the next iteration of mobile technology, there is an emphasis on using cloud-native technologies of which Kubernetes is meant to play an important role. Kubernetes is vital for automating deployment, scaling, and management of containerized applications. The containers that make up the 5G network are grouped into logical units for easy management and discovery. For this 5G deployment, we approach the 5G function implementation on a cloud platform instead of a single physical node or distributed across several nodes for ease of deployment and scaling. The host OS configuration used for the project deployment is Ubuntu's Linux - we tuned it to a specific kernel needed to support the UPF (GTP5G Module) and also for optimization of deployment. Furthermore, the network firewall policies for the instance were defined bearing in mind that for Kubernetes, ETCD would manage network rules and post forwarding activity. Each network function kept its network interfaces as we used declaratively configuration for Multi-network interfaces. This allows for the separation of control traffic from user traffic within the Kubernetes cluster.

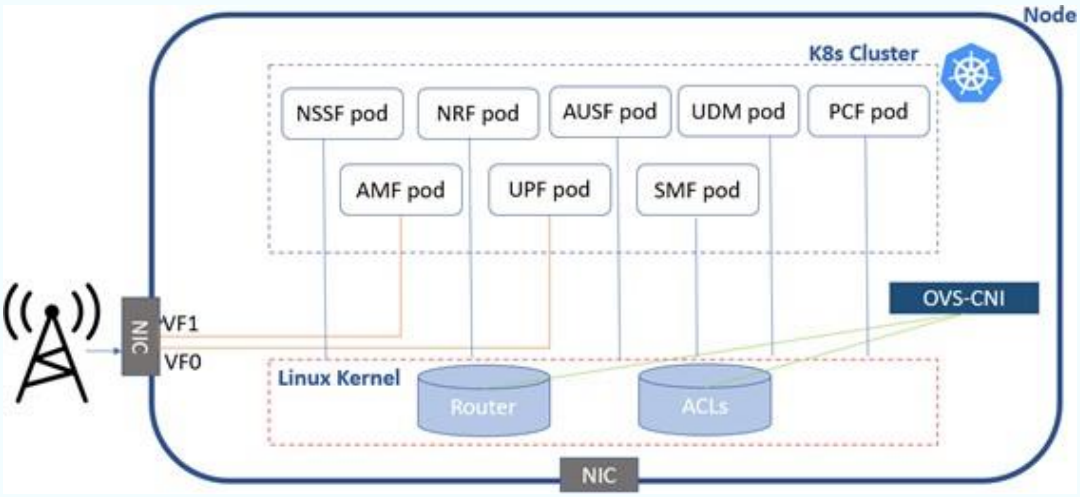

Figure 5 -Kubernetes cluster setup (Source: Self)

After setting up the Kubernetes cluster and controller, the 5G services were created using open-source free 5gc images as the composite application. The core network functions of the free5GC are deployed on K8s in a cluster as seen in Figure 3b. However, there is a requirement to create other network interfaces in the pod. One network interface will be used for the Kubernetes cluster, a second interface for control plane traffic going to SMF, and a third interface for the user plane data traffic going to UPF. The front haul traffic that goes from AMF to the UE has the same subnet. The architecture is shown in Figure 5 below. The network attachments are defined within the configuration files on the repository. It should be noted that VLANs can be used as an alternative to separate the network interfaces on large-scale deployment. For this project, a virtual layer2 bridge was used to bind the new virtual network interfaces. The configurations are deployed, and the containers are initialized for each 5G component. The web UI is configured for connecting to the AMF node and creating a subscriber ID required. The access to the web user interface of free 5gc is through the TCP exposed port. The user interface allows for entry

of the subscriber details of the SIM cards and also for saving the subscriber profile in the NSSF and UDM MongoDB database backend. Except for the SMF and UPF, all config files for the 5G SA core functions only contain the function's IP bind addresses/ local Interface names and the IP address/ DNS name of the NRF. It is important to note that we use dockerized free5gc stage3 and ride it on Kubernetes. The ETCD was also deployed in High Availability mode to ensure resilience for the 5G functions within the cluster. The structure of the 5G software components and their interconnections via APIs is shown in Figure 6

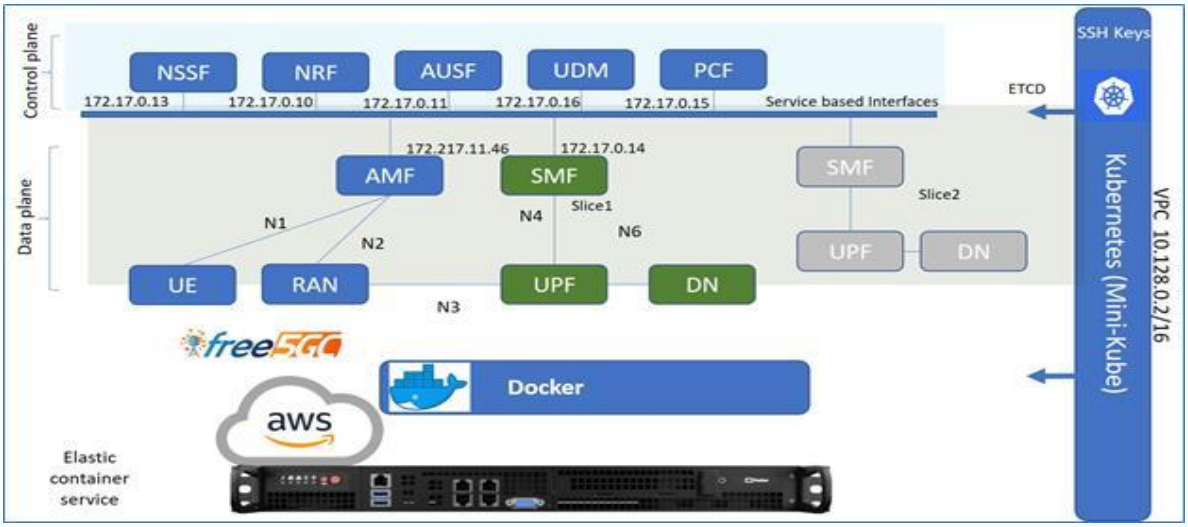

Figure 6 – Kubernetes deployment setup on the cloud (Source: Self)

Docker setup (Local deployment)

The project aims to automate the installation and the set-up of the 5G Non-standalone network to simplify 5G deployment, we used tools that can be automatable, can improve performance, and use fewer resources. Docker containers provide the ability to run applications quickly and reliably from one computing environment to another. Combined with programming we can develop a code (ansible code) to install all Non-standalone 5G applications in containers, applications such as HSS, PCRF, AMF, SMF, and UPF will represent the core network, while two more containers will represent the eNB and UE. figure 1 shows the architecture of the 5G containers network. An ansible-playbook (code) will be used to deploy the architecture shown in figure 1 users will only need to run one command to have a 5G network running either locally or on the cloud ”`ansible-playbook -K our-doker.yml -e "internet_network_interface=<< internet network interface name>>`”.

The container's internal network is created upon running the deployment file, Creating a subnet for all parts of the network UE, eNB, and the core network. The used IP subnet is 192.163.6.0/24 And then a bridge is configured to route between the 5G network and the local computer’s network which is 192.168.1.0/24.

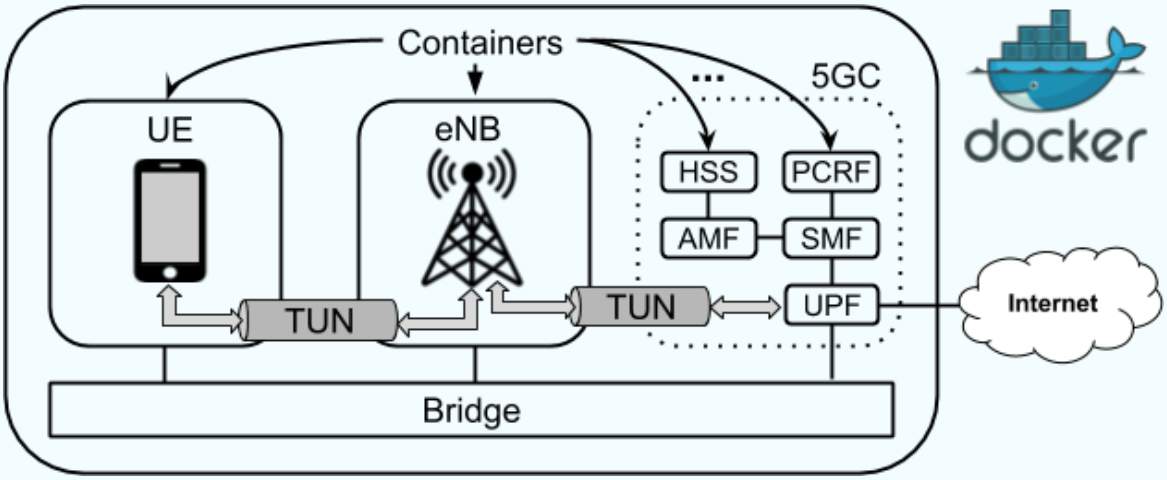

Figure 7: docker set-up architecture [12].

For the full detailed deployment steps please check our GitHub repository in the link below:https://github.com/Edwin-programmer/Project5G-ansible-deployment/tree/main/Docker%20deployment#configuration-steps

Limitation

For core node elements that need to communicate with each other, Kubernetes will call IP tables for NAT conversions. There is already a NAT conversion of the AWS firewall and where one performs another conversion on ubuntu IP tables, it might affect the performance.

Secondly, because of the requirement for multiple network cards, we limited the use of worker nodes where necessary. Using multiple worker nodes might increase the complexity beyond that which is needed for this simple deployment, especially on public

cloud environments. However, using multiple worker nodes should be easily achievable with on-premise or bare metal platforms like vSphere.

Implementation priority was to set up the Core 5G network environment for Kubernetes, its elements, and functions while integrating gNb and RAN components were out of scope due to the bounds of time limitation of the project delivery dates.

V. Result

Kubernetes deployment result

The network attachment definition is a type of Custom Resource Definitions (CRDs) within the Kubernetes cluster. It provides configuration items, such as VPC and subnet, for containers to connect to the elastic network interface. Workloads associated with network attachment definitions can connect to the elastic network interface so that the containers can be directly bound with the elastic network interfaces to expose services externally. Multus CNI is a container network interface (CNI) plugin for Kubernetes that enables attaching multiple network interfaces to pods. Typically, in Kubernetes each pod only has one network interface (apart from a loopback) -- with Multus you can create a multi-homed pod that has multiple interfaces. Below shows the YAML file for the project's network attachment definition.

```
 GNU nano 2.9.3                                    ovs-net-crd.yaml

apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: ovs-net
  annotations:
    k8s.v1.cni.cncf.io/resourceName: ovs-cni.network.kubevirt.io/br1
spec:
  config: '{
      "cniVersion": "0.3.1",
      "type": "ovs",
      "bridge": "br1"
    }'
```

Figure 8 - Network attachment definitions (Source: Self)

Also below show the active node details and master control plane after Kubernetes is running.

```
zuwaomoigui@cloudnative5g:~$ sudo kubectl get node
NAME           STATUS   ROLES                  AGE   VERSION
cloudnative5g   Ready    control-plane,master   10d   v1.20.2
zuwaomoigui@cloudnative5g:~$ sudo kubectl get svc
NAME                          TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)              AGE
etcd-ha-operator-metrics      ClusterIP   10.107.242.98    <none>        8383/TCP             10d
example-etcd-cluster          ClusterIP   None             <none>        2379/TCP,2380/TCP    10d
example-etcd-cluster-client   ClusterIP   10.111.186.32    <none>        2379/TCP             10d
free5gmano-svc                NodePort    10.103.244.87    <none>        8000:30088/TCP       10d
kube5gnfvo                    NodePort    10.101.68.197    <none>        8000:30888/TCP       10d
kube5gnfvo-mysql              NodePort    10.101.182.133   <none>        3306:30036/TCP       10d
kubernetes                    ClusterIP   10.96.0.1        <none>        443/TCP              10d
mongodb-svc                   ClusterIP   None             <none>        27017/TCP            10d
```

Figure 9 - Kubernetes nodes after setup (Source: Self)

The containerized deployment of the 5G core nodes are shown below, further details on the configuration steps are within the repository https://github.com/Edwin-programmer/Project5G-ansible-deployment/blob/main/Kubernetes%20deployment/README.md :

Deploy 5GC

```
sudo kubectl apply -f unix-daemonset.yaml
sudo kubectl apply -f free5gc-mongodb.yaml
sudo kubectl apply -f free5gc-configmap.yaml
sudo kubectl apply -f free5gc-nrf.yaml
sudo kubectl apply -f free5gc-ausf.yaml
sudo kubectl apply -f free5gc-smf.yaml
sudo kubectl apply -f free5gc-nssf.yaml
sudo kubectl apply -f free5gc-pcf.yaml
sudo kubectl apply -f free5gc-udm.yaml
sudo kubectl apply -f free5gc-udr.yaml
```

Deploy AMF/UPF

```
sudo sysctl -w net.ipv4.ip_forward=1
sudo iptables -t nat -A POSTROUTING -o ens4 -j MASQUERADE
git clone https://github.com/PrinzOwO/gtp5g
cd gtp5g
sudo apt install make
```

Figure 10 - YAML deployment for 5G core functions

After applying the manifests, the running status of the pods can be seen below with network interfaces separating the control and user planes:

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
default	etcd-ha-operator-6fbb954b8f-wtwbq	2/2	Running	0	12m	172.17.0.3	cloudnative5g
default	example-etcd-cluster-0	1/1	Running	0	11m	172.17.0.4	cloudnative5g
default	example-etcd-cluster-1	1/1	Running	0	11m	172.17.0.5	cloudnative5g
default	example-etcd-cluster-2	1/1	Running	0	11m	172.17.0.6	cloudnative5g
default	free5gc-ausf-deployment-597dc77797-2r8w6	1/1	Running	0	7m27s	172.17.0.11	cloudnative5g
default	free5gc-mongodb-ddd76dd4c-fsxcl	1/1	Running	0	7m29s	172.17.0.12	cloudnative5g
default	free5gc-nrf-deployment-7d4dc6f994-z8c2k	1/1	Running	0	7m28s	172.17.0.10	cloudnative5g
default	free5gc-nssf-deployment-77899dd9f-ggjc2	1/1	Running	0	7m25s	172.17.0.13	cloudnative5g
default	free5gc-pcf-deployment-d6dd4c65d-ddq4s	1/1	Running	0	7m25s	172.17.0.15	cloudnative5g
default	free5gc-smf-deployment-57d4d9b884-ldd8b	1/1	Running	0	7m26s	172.17.0.14	cloudnative5g
default	free5gc-udm-deployment-547bc84d44-cmxtg	1/1	Running	0	7m24s	172.17.0.16	cloudnative5g
default	free5gc-udr-deployment-5598565dcc-kpmrd	1/1	Running	0	7m22s	172.17.0.17	cloudnative5g
default	free5gmano-deployment-897dffd55-59dfg	1/1	Running	0	7m39s	172.17.0.9	cloudnative5g
default	kube5gnfvo-75679bfdb7-4pwv5	1/1	Running	1	7m41s	172.17.0.8	cloudnative5g
default	kube5gnfvo-mysql-f66559bd5-r284h	1/1	Running	0	7m42s	172.17.0.7	cloudnative5g
kube-system	coredns-74ff55c5b-14rp6	1/1	Running	3	157m	172.17.0.2	cloudnative5g
kube-system	etcd-cloudnative5g	1/1	Running	3	157m	10.128.0.2	cloudnative5g
kube-system	exporter-lpsdq	1/1	Running	0	8m40s	10.128.0.2	cloudnative5g
kube-system	kube-apiserver-cloudnative5g	1/1	Running	3	157m	10.128.0.2	cloudnative5g
kube-system	kube-controller-manager-cloudnative5g	1/1	Running	3	157m	10.128.0.2	cloudnative5g
kube-system	kube-flannel-ds-jjl5w	1/1	Running	0	12m	10.128.0.2	cloudnative5g
kube-system	kube-multus-ds-amd64-wbqfm	1/1	Running	0	8m55s	10.128.0.2	cloudnative5g
kube-system	kube-proxy-5wgkh	1/1	Running	3	157m	10.128.0.2	cloudnative5g
kube-system	kube-scheduler-cloudnative5g	1/1	Running	3	157m	10.128.0.2	cloudnative5g
kube-system	network-controller-server-unix-7jb6n	1/1	Running	0	7m31s	10.128.0.2	cloudnative5g
kube-system	ovs-cni-amd64-hs47q	1/1	Running	0	9m7s	10.128.0.2	cloudnative5g
kube-system	storage-provisioner	1/1	Running	6	157m	10.128.0.2	cloudnative5g

Figure 11 - Results after deploying 5G core network

```
smf
2021-03-09T17:26:45Z [INFO][SMF][App] SMF version:
        free5GC version: v3.0.5
        build time:      2021-03-09T16:15:40Z
        commit hash:     04c01ec5
        commit time:     2021-01-30T17:01:30Z
        go version:      go1.14.4 linux/amd64
2021-03-09T17:26:45Z [INFO][SMF][Init] SMF Log level is set to [info] level
2021-03-09T17:26:45Z [INFO][LIB][NAS] set log level : info
2021-03-09T17:26:45Z [INFO][LIB][NAS] set report call : false
2021-03-09T17:26:45Z [INFO][LIB][NGAP] set log level : info
2021-03-09T17:26:45Z [INFO][LIB][NGAP] set report call : false
2021-03-09T17:26:45Z [INFO][LIB][Aper] set log level : info
2021-03-09T17:26:45Z [INFO][LIB][Aper] set report call : false
2021-03-09T17:26:45Z [INFO][LIB][Path] set log level : info
2021-03-09T17:26:45Z [INFO][LIB][Path] set report call : false
2021-03-09T17:26:45Z [INFO][LIB][OAPI] set log level : info
2021-03-09T17:26:45Z [INFO][LIB][OAPI] set report call : false
2021-03-09T17:26:45Z [INFO][LIB][PFCP] set log level : info
2021-03-09T17:26:45Z [INFO][LIB][PFCP] set report call : false
2021-03-09T17:26:45Z [INFO][SMF][CFG] SMF config version [1.0.0]
2021-03-09T17:26:45Z [INFO][SMF][CFG] UE-Routing config version [1.0.0]
2021-03-09T17:26:45Z [INFO][SMF][CTX] smfconfig Info: Version[1.0.0] Description[SMF initial local configuration]
2021-03-09T17:26:45Z [INFO][SMF][CTX] Endpoints: [127.0.0.8]
2021-03-09T17:26:45Z [INFO][SMF][Init] Server started
2021-03-09T17:26:45Z [INFO][UDR][App] udr
2021-03-09T17:26:45Z [INFO][UDR][App] UDR version:
```

Figure 12 - SMF core function

Above shows the running test for the core nodes with an SMF example after deployment. With the logs, we check the SMF pod to see that it was able to initiate a PFCP connection with the UPF.

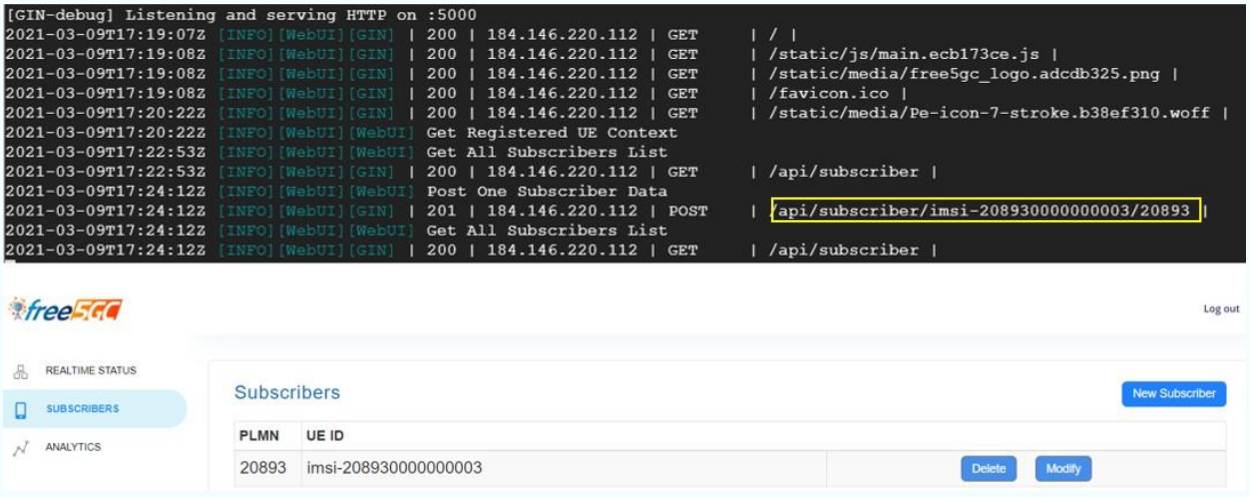

Figure 13 - Subscriber IMSI configuration and link to Core Network

TCP localhost 5000 is used for the GUI subscriber configurations, the results above show when a subscriber detail is added.

Docker deployment results

The ansible-playbook is a file that contains the code that when deployed its configured to run the command mentioned in the docker deployment command to install HSS, AMF, SMF, UPF, and PCRF images for the core network As well as the eNB and UE images are also installed. To verify that the images are installed and running we use the command " docker ps " to show the running docker container, the results of running the “ docker ps ” command is shown in figure 10.

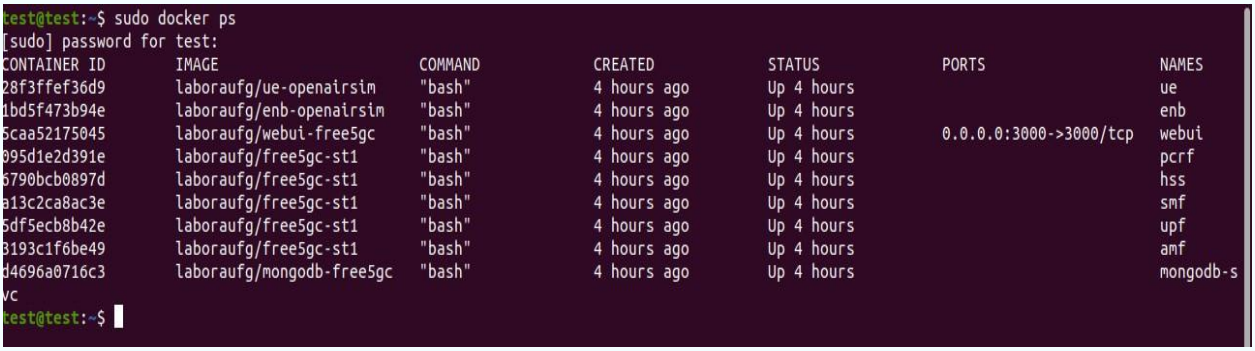

Figure 14: the results of checking the running docker containers (Source: Self)

The ansible file also installs a database and installs free5GC web UIto manage subscribers and users data, the default IP address is 0.0.0.0:3000. Figure 11 shows free5GC web UI.

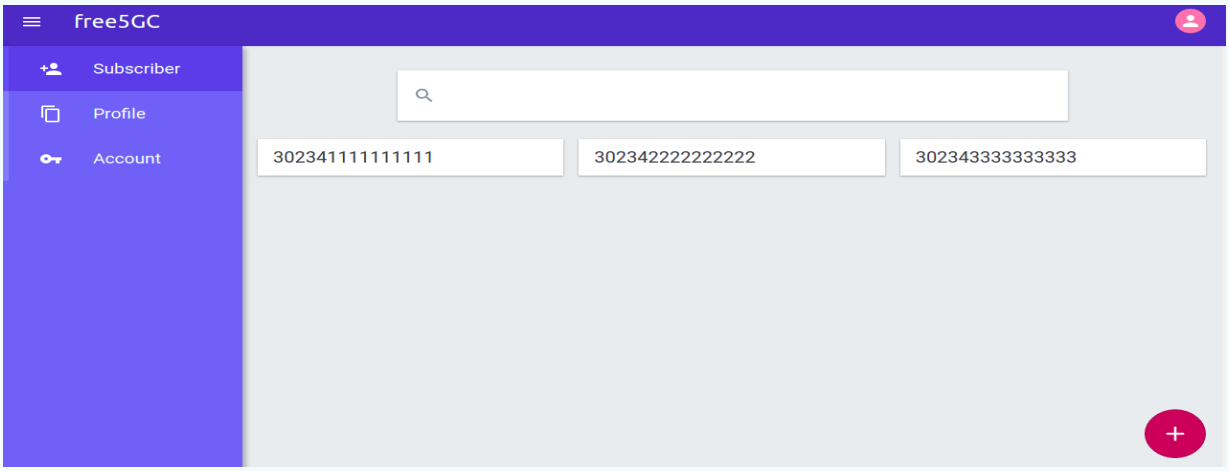

Figure 15: Free5gc web user interface (Source: Self)

To test the connectivity of the Us we accessed the UE container image and tried pinging google.com to test if the UE is connected to the internet. Figure 12 shows the successful ping results from the UE container to google.com.

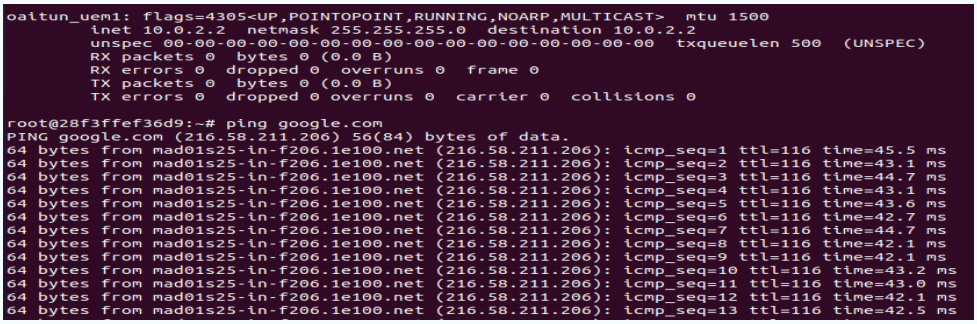

Figure 16: Results of the UE pinging google.com (Source: Self)

VI. Conclusion

Through our exploration of related works, we have seen that it is indeed important and efficient to automate network functions, especially in 5G, due to its sheer differences with previous generations of telecom technologies, namely 5G has a lot of virtualized network functions. Due to this high number, maintaining and automating them poses a significant challenge in terms of resource allocation and management. We have seen that it is possible to automate this deployment using ansible and containers, where we could make changes to one master node and the changes could be reflected throughout the whole cluster. We have demonstrated this using both cloud and local deployments. We were able to deploy 5G core network functions using Kubernetes container orchestration tools in the cloud however, a full 5G Non-Standalone network setup was done using Docker in a local setup, with necessary testing done in both cases to verify the integrity of the deployments.

VII Appendices

Appendix A: Functions Matrix

This table is created based on the default configuration files of Free5GC (Open-source).

Function	Exposed Ports	Dependencies	Dependencies URI (from default config files)
AMF	29518	NRF	NRF: https://localhost:29510
AUSF	29509	NRF	NRF: https://localhost:29510
NRF	29510	MONGODB	MONGODB: mongodb://127.0.0.1:27017
NSSF	29531	NRF	NRF: http://free5gc-nrf:29510/nnrf-nfm/v1/nf-instances
			NRF: http://free5gc-nrf:8081/nnrf-nfm/v1/nf-instances
PCF	29507	N/A	N/A
SMF	29502	PFCP	PFCP: 10.200.200.1
		UPF	UPF: 10.200.200.101
		NRF	NRF: https://localhost:29510
UDM	29503	UDR	UDR client: 127.0.0.1:29504
		NRF	NRF client: 127.0.0.1:29510
UDR	29504	MONGODB	MONGODB: mongodb://localhost:27017
		NRF	NRF: https://localhost:29510
UPF	N/A	PFCP	PFCP: 10.200.200.101
		GTPU	GTPU: 10.200.200.102
		APN_LIST	APN_LIST: 60.60.0.0/24
		APN	APN: internet

VIII. References

[1] Arouk, O., & Nikaein, N. (2020, April). 5G Cloud-Native: Network Management & Automation. In *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium* (pp. 1-2). IEEE.

[2] Mwanje, S., Decarreau, G., Mannweiler, C., Naseer-ul-Islam, M., & Schmelz, L. C. (2016, September). Network management automation in 5G: Challenges and opportunities. In *2016 IEEE 27th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)* (pp. 1-6). IEEE.

[3] Kafle, V. P., Martinez-Julia, P., & Miyazawa, T. (2019). Automation of 5G network slice control functions with machine learning. *IEEE Communications Standards Magazine*, *3*(3), 54-62.

[4] van der Meer, S., Keeney, J., & Fallon, L. (2018, April). 5g networks must be autonomic!. In *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium* (pp. 1-5). IEEE.

[5] Huang, C. Y., Ho, C. Y., Nikaein, N., & Cheng, R. G. (2018, November). Design and prototype of a virtualized 5G infrastructure supporting network slicing. In *2018 IEEE 23rd International Conference on Digital Signal Processing (DSP)* (pp. 1-5). IEEE.

[6] Chirivella-Perez, E., Calero, J. M. A., Wang, Q., & Gutiérrez-Aguado, J. (2018). Orchestration architecture for automatic deployment of 5g services from bare metal in mobile edge computing infrastructure. *Wireless Communications and Mobile Computing*, *2018*.

[7] Wiranata, F. A., Shalannanda, W., Mulyawan, R., & Adiono, T. (2020, November). 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* (pp. 1-5). IEEE.

[8] Nakimuli, W., Landi, G., Perez, R., Pergolesi, M., Molla, M., Ntogkas, C., ... & Salsano, S. (2020, September). Automatic deployment, execution and analysis of 5G experiments using the 5G EVE platform. In *2020 IEEE 3rd 5G World Forum (5GWF)* (pp. 372-377). IEEE.

[9] Arouk, O., & Nikaein, N. (2020, December). Kube5G: A Cloud-Native 5G Service Platform. In *GLOBECOM 2020-2020 IEEE Global Communications Conference* (pp. 1-6). IEEE.

[10] 3GPP TS 23.501 V15.0.0 (2017-12) *System Architecture for 5G System (Stage 2)*

[11] Rahnema,M, Dryjanski, M. (2017). “From LTE to LTE-Advanced Pro and 5G”, Artech House. Available online at: https://www.grandmetric.com/knowledge-base/research/from-lte-to-lte-advanced-pro-and-5g-book/

[12] Labora. (2021). Link taken from https://github.com/LABORA-INF-UFG/NetSoft2020-Tutorial4-Demo2-Exp1/blob/master/images/demo2-exp1.png

[13] Free5gc official website. (2021). https://www.free5gc.org/