

# Final Project

Arthur Martins Braga

CSC7342 - Virtualization: Concept and implementation

December 10, 2024

# 1 Introduction

This report documents the steps and results of deploying two distinct projects as part of the final assignment. The first task involved deploying a full-stack application using Kubernetes in a Minikube cluster, while the second task focused on setting up a mobile network emulator using Docker Compose.

The report outlines the processes followed, challenges encountered, and solutions implemented during the deployment of each project. Screenshots and references are provided to illustrate key steps and demonstrate the functionality of the deployed systems. The project's code and configurations are also made available in a GitHub repository for further review and replication.

## 2 Repository Overview

The project's source code and configurations are available on GitHub: <https://github.com/ArthurMbraga/k8s-docker-examples>.

The repository is structured as follows:

- **fullstack-k8s/**: Files and configurations for Task 1 (deploying the full-stack application with Kubernetes).
- **docker-compose-network/**: Resources for Task 2 (deploying the mobile network emulator using Docker Compose).

## 3 Task 1: Full-Stack Deployment with Kubernetes

This section describes the steps and results of deploying a sample full-stack application using Kubernetes in a Minikube cluster.

### 3.1 Overview

The deployment includes a backend, a frontend, and a database. The configuration files were created to configure the deployments and services for each component. The file structure used for this project is shown in Figure 1.

### 3.2 Minikube Dashboard

The Minikube dashboard was used to monitor deployments, replicas, and pods. All components of the stack were successfully deployed and running. Figure 2 shows a screenshot of the Minikube dashboard.

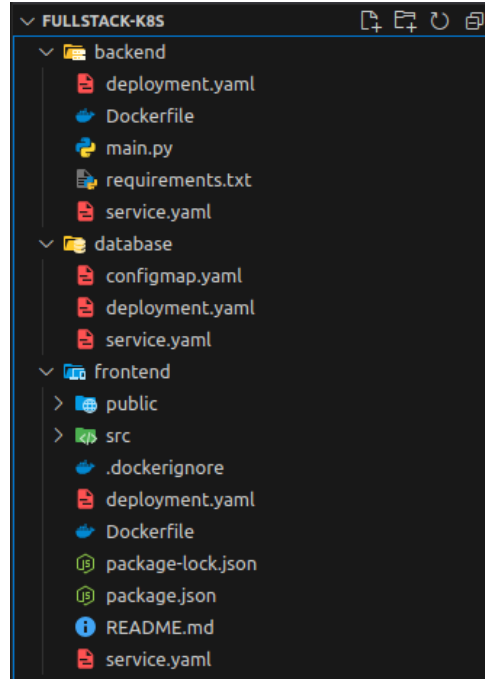


Figure 1: File structure of the project

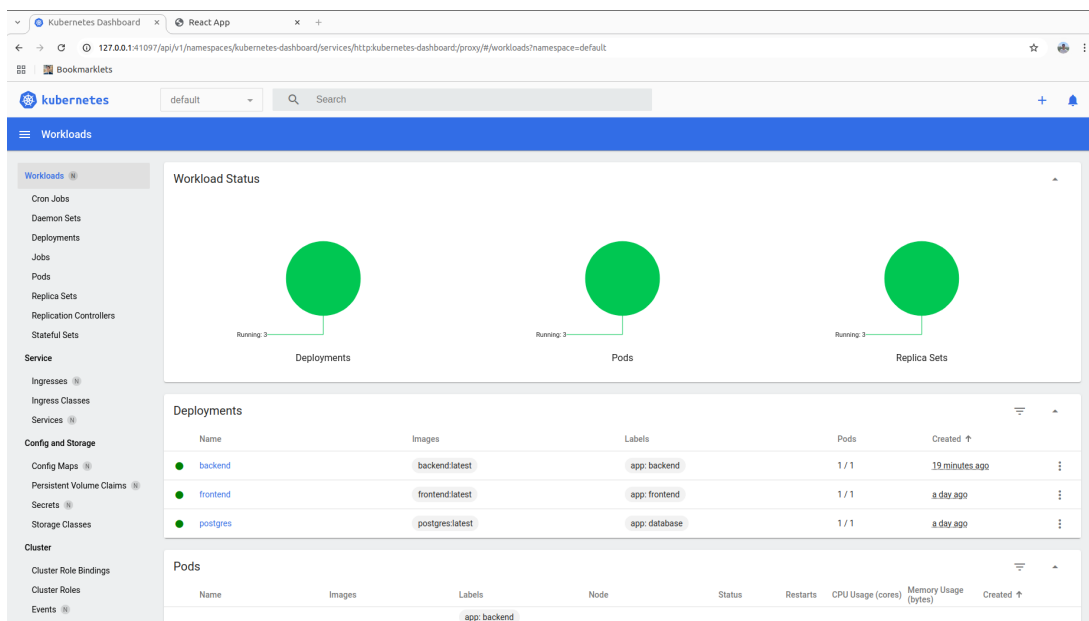


Figure 2: Minikube dashboard showing running pods and deployments

### 3.3 Frontend Application

The frontend of the application was successfully deployed. Initially, the application did not show data, as illustrated in Figure 3.

After adding some sample data, the front-end was updated as shown in Figure 4.

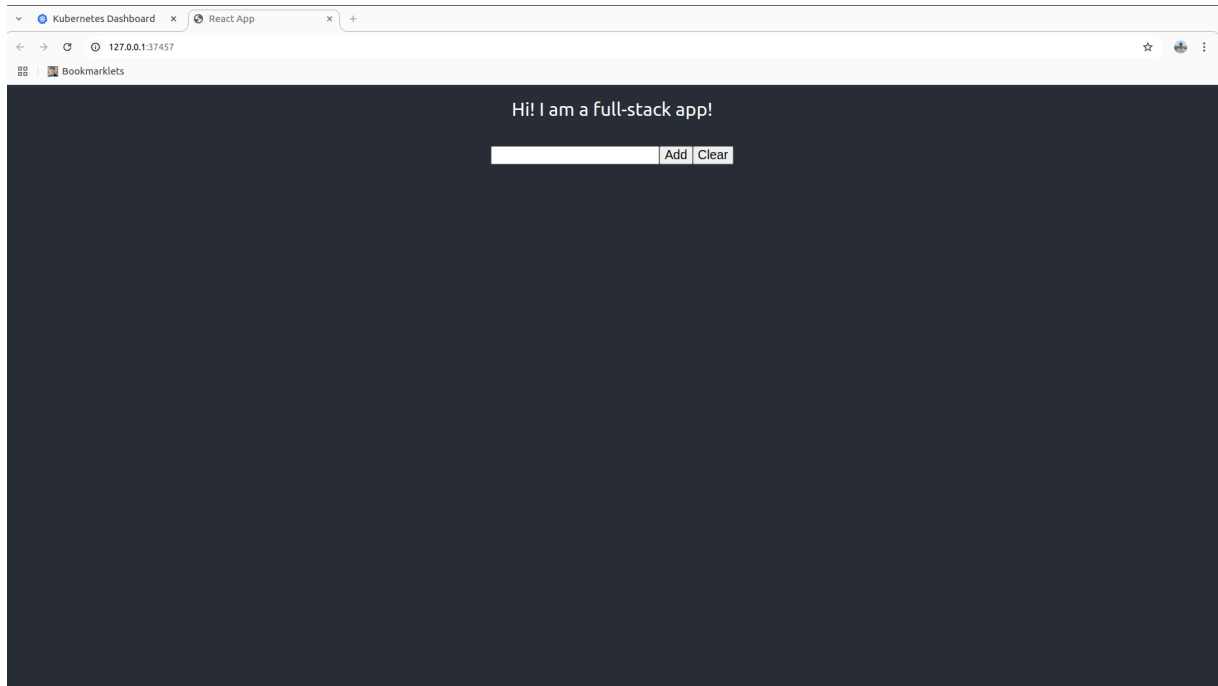


Figure 3: Frontend of the application with no data

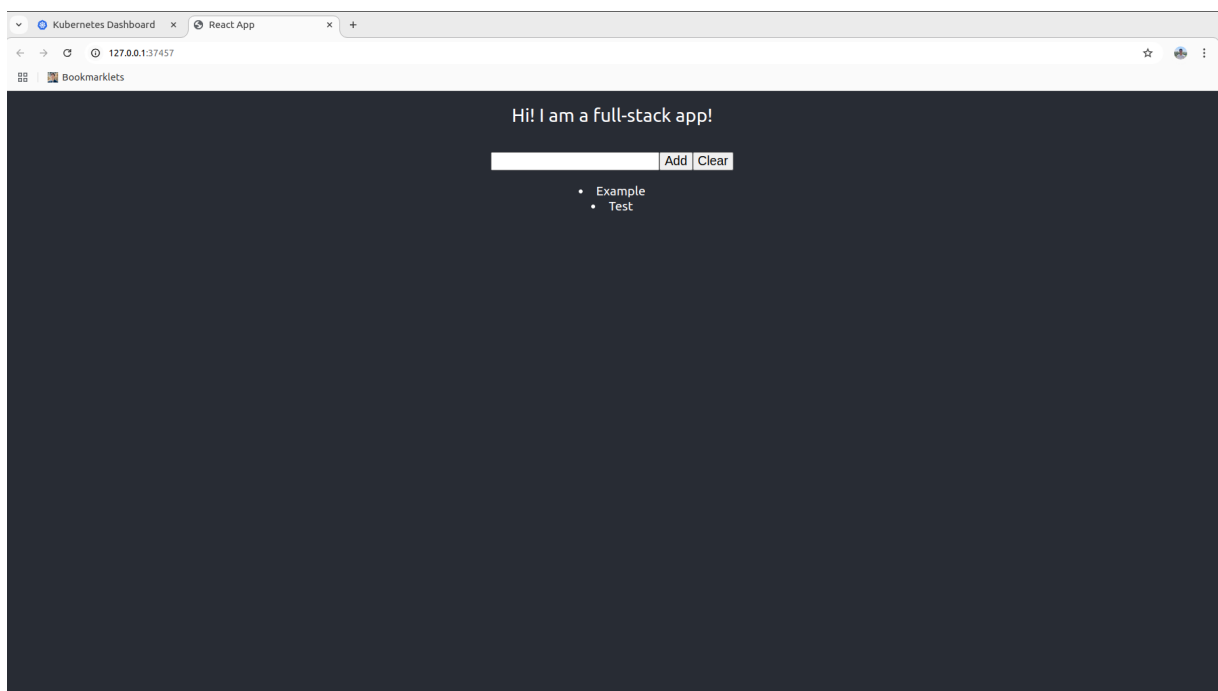


Figure 4: Frontend of the application with sample data added

### 3.4 Conclusion

Task 1 was completed successfully. The full stack application was deployed in the Minikube cluster and the front-end application operated as expected. Following the tutorial was straightforward, meaning that only minimal modifications were needed on my part to make it work. I encountered some troubleshooting related to Python and

Node versions, but these issues were easy to resolve.

## 4 Task 2: Deployment of Mobile Network Emulator Using Docker Compose

This section outlines the steps taken to deploy a mobile network emulator using Docker Compose. Screenshots of the console output are provided for each step.

### 4.1 Step 1: Download Updated Images

The first step involved downloading the updated images from the repository. These images are essential for the deployment. The console output of this step is shown in Figure 5.

```
Creating rfsim5g-mysql ... done
Creating rfsim5g-oai-nrf ... done
Creating rfsim5g-oai-amf ... done
Creating rfsim5g-oai-smf ... done
Creating rfsim5g-oai-spgwu ... done
Creating rfsim5g-oai-ext-dn ... done
```

Figure 5: Downloading updated images

### 4.2 Step 2: Deploy Network

The ‘docker-compose.yml’ file was modified to address issues with variable configurations:

- ‘services.mysql.image’ was updated to ‘mysql:5.7’.
- ‘services.mysql.healthcheck.retries’ was set to ‘5’.

After these changes, the network was deployed successfully (Figure 6).

Name	Command	State	Ports
rfsim5g-mysql	docker-entrypoint.sh mysqld	Up (healthy)	3306/tcp, 33060/tcp
rfsim5g-oai-amf	python3 /openair-amf/bin/e ...	Up (healthy)	38412/sctp, 80/tcp, 9090/tcp
rfsim5g-oai-ext-dn	/bin/bash -c iptables -t ...	Up (healthy)	
rfsim5g-oai-nrf	python3 /openair-nrf/bin/e ...	Up (healthy)	80/tcp, 9090/tcp
rfsim5g-oai-smf	python3 /openair-smf/bin/e ...	Up (healthy)	80/tcp, 8080/tcp, 8805/udp
rfsim5g-oai-spgwu	python3 /openair-spgwu-tin ...	Up (healthy)	2152/udp, 8805/udp

Figure 6: Deploying the network

### 4.3 Step 3: Check Status

During this step, an issue with the ‘mysql-healthcheck.sh’ script was resolved. The problem was related to LF (Line Feed) and CRLF (Carriage Return Line Feed) encoding, which was corrected by converting to LF. The output is shown in Figure 7.

Name	Command	State	Ports
rfsim5g-mysql	docker-entrypoint.sh mysqld	Up (healthy)	3306/tcp, 33060/tcp
rfsim5g-oai-amf	python3 /openair-amf/bin/e ...	Up (healthy)	38412/sctp, 80/tcp, 9090/tcp
rfsim5g-oai-ext-dn	/bin/bash -c iptables -t ...	Up (healthy)	
rfsim5g-oai-gnb	/tini -v -- /opt/oai-gnb/b ...	Up (healthy)	
rfsim5g-oai-nrf	python3 /openair-nrf/bin/e ...	Up (healthy)	80/tcp, 9090/tcp
rfsim5g-oai-smf	python3 /openair-smf/bin/e ...	Up (healthy)	80/tcp, 8080/tcp, 8805/udp
rfsim5g-oai-spgwu	python3 /openair-spgwu-tin ...	Up (healthy)	2152/udp, 8805/udp

Figure 7: Checking the status

### 4.4 Step 4: Check Network Interface

The ‘ifconfig’ command was executed to verify the network interface configuration. See Figure 8.

```
rfsim5g-public: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
  inet 192.168.71.129 netmask 255.255.255.192 broadcast 192.168.71.191
  inet6 fe80::42:d7ff:fe2c:b94 prefixlen 64 scopeid 0x20<link>
  ether 02:42:d7:2c:0b:94 txqueuelen 0 (Ethernet)
  RX packets 4 bytes 112 (112.0 B)
  RX errors 0 dropped 0 overruns 0 frame 0
  TX packets 68 bytes 10322 (10.3 KB)
  TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

rfsim5g-traffic: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
  inet 192.168.72.129 netmask 255.255.255.192 broadcast 192.168.72.191
  inet6 fe80::42:c3ff:fed6:de50 prefixlen 64 scopeid 0x20<link>
  ether 02:42:c3:d6:de:50 txqueuelen 0 (Ethernet)
  RX packets 1 bytes 28 (28.0 B)
  RX errors 0 dropped 0 overruns 0 frame 0
  TX packets 68 bytes 10322 (10.3 KB)
  TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Figure 8: Network interface configuration

### 4.5 Steps 5 and 6: Deploy OAI Components

The following OAI components were deployed in RF simulator mode and standalone mode:

- OAI gNB (Figure 9)
- OAI NR-UE (Figure 10).

In ‘docker-compose.yml’, the ‘-sa’ flag was removed from:

- services.oai-nr-ue.environment.USE\_ADDITIONAL\_OPTIONS
- services.oai-nr-ue2.environment.USE\_ADDITIONAL\_OPTIONS

Name	Command	State	Ports
rfsim5g-mysql	docker-entrypoint.sh mysqld	Up (healthy)	3306/tcp, 33060/tcp
rfsim5g-oai-amf	python3 /openair-amf/bin/e ...	Up (healthy)	38412/sctp, 80/tcp, 9090/tcp
rfsim5g-oai-ext-dn	/bin/bash -c iptables -t ...	Up (healthy)	
rfsim5g-oai-gnb	/tini -v -- /opt/oai-gnb/b ...	Up (healthy)	
rfsim5g-oai-nr-ue	/tini -v -- /opt/oai-nr-ue ...	Up (healthy)	
rfsim5g-oai-nrf	python3 /openair-nrf/bin/e ...	Up (healthy)	80/tcp, 9090/tcp
rfsim5g-oai-smf	python3 /openair-smf/bin/e ...	Up (healthy)	80/tcp, 8080/tcp, 8805/udp
rfsim5g-oai-spgwu	python3 /openair-spgwu-tin ...	Up (healthy)	2152/udp, 8805/udp

Figure 9: Deploying OAI gNB

```
er exec -it rfsim5g-oai-nr-ue /bin/bash
root@0def12328ef3:/opt/oai-nr-ue# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.71.150 netmask 255.255.255.192 broadcast 192.168.71.191
    ether 02:42:c0:a8:47:96 txqueuelen 0 (Ethernet)
    RX packets 3182774 bytes 49814990116 (49.8 GB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 2810137 bytes 66953983410 (66.9 GB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

oaitun_ue1: flags=209<UP,POINTOPOINT,RUNNING,NOARP> mtu 1500
    inet 12.1.1.4 netmask 255.255.255.0 destination 12.1.1.4
    inet6 fe80::66de:ed88:abf6:2502 prefixlen 64 scopeid 0x20<link>
    unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00 txqueuelen 500 (UNSPEC)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 6 bytes 288 (288.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Figure 10: Deploying OAI NR-UE

## 4.6 Step 7: Verify OAI UE Connection

The OAI UE connection was verified to ensure successful deployment (Figure 11).

```
root@0def12328ef3:/opt/oai-nr-ue# ping -I oaitun_ue1 -c 10 www.lemonde.fr
PING lemonde.map.fastly.net (151.101.122.217) from 12.1.1.4 oaitun_ue1: 56(84) bytes of data.
64 bytes from 151.101.122.217 (151.101.122.217): icmp_seq=1 ttl=253 time=87.2 ms
64 bytes from 151.101.122.217 (151.101.122.217): icmp_seq=2 ttl=253 time=91.7 ms
64 bytes from 151.101.122.217 (151.101.122.217): icmp_seq=3 ttl=253 time=60.9 ms
64 bytes from 151.101.122.217 (151.101.122.217): icmp_seq=4 ttl=253 time=83.4 ms
64 bytes from 151.101.122.217 (151.101.122.217): icmp_seq=5 ttl=253 time=50.7 ms
64 bytes from 151.101.122.217 (151.101.122.217): icmp_seq=6 ttl=253 time=86.4 ms
64 bytes from 151.101.122.217 (151.101.122.217): icmp_seq=7 ttl=253 time=61.6 ms
64 bytes from 151.101.122.217 (151.101.122.217): icmp_seq=8 ttl=253 time=62.4 ms
64 bytes from 151.101.122.217 (151.101.122.217): icmp_seq=9 ttl=253 time=80.6 ms
64 bytes from 151.101.122.217 (151.101.122.217): icmp_seq=10 ttl=253 time=78.7 ms

--- lemonde.map.fastly.net ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9015ms
rtt min/avg/max/mdev = 50.666/74.357/91.690/13.399 ms
```

Figure 11: Verifying OAI UE connection



## 4.7 Step 8: Check Internet Connectivity

Internet connectivity was checked as shown in Figure 12.

```
er exec -it rfsim5g-oai-nr-ue /bin/bash
root@0def12328ef3:/opt/oai-nr-ue# ping -I oaitun_ue1 -c 2 192.168.72.135
PING 192.168.72.135 (192.168.72.135) from 12.1.1.4 oaitun_ue1: 56(84) bytes of data.
64 bytes from 192.168.72.135: icmp_seq=1 ttl=63 time=24.3 ms
64 bytes from 192.168.72.135: icmp_seq=2 ttl=63 time=19.4 ms

--- 192.168.72.135 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 19.425/21.864/24.304/2.439 ms
```

Figure 12: Checking Internet connectivity

## 4.8 Steps 9 and 10: Test with iperf

The ‘iperf’ tool was used to test connectivity:

- Step 9: Start the ‘iperf’ server inside the NR-UE container. Initially, the command ‘iperf -B 12.1.1.2 -u -i 1 -s’ failed. Using ‘ip addr show’, it was determined that the correct IP address was ‘12.1.1.4’ (Figure 13).
- Step 10: Start the ‘iperf’ client inside the ext-dn container (Figure 14).

```
root@0def12328ef3:/opt/oai-nr-ue# iperf -B 12.1.1.4 -u -i 1 -s
-----
Server listening on UDP port 5001
UDP buffer size: 208 KByte (default)
-----
```

Figure 13: Starting iperf server

```

inet 192.168.71.150/26 brd 192.168.71.191 scope global eth0
valid lft forever preferred lft forever
root@0def12328ef3:/opt/oai-nr-ue# iperf -B 12.1.1.4 -u -i 1 -s
Server listening on UDP port 5001
UDP buffer size: 208 KByte (default)
-----
~Croot@0def12328ef3:/opt/oai-nr-ue# iperf -B 12.1.1.4 -u -i 1 -s
Server listening on UDP port 5001
UDP buffer size: 208 KByte (default)
-----
[ 1] local 12.1.1.4 port 5001 connected with 192.168.72.135 port 46082
[ ID] Interval      Transfer      Bandwidth      Jitter    Lost/Total Datagrams
[ 1] 0.0000-1.0000 sec  61.7 KBytes  506 Kbits/sec  0.454 ms  0/43 (0%)
[ 1] 1.0000-2.0000 sec  63.2 KBytes  517 Kbits/sec  0.530 ms  0/44 (0%)
[ 1] 2.0000-3.0000 sec  61.7 KBytes  506 Kbits/sec  0.892 ms  0/43 (0%)
[ 1] 3.0000-4.0000 sec  63.2 KBytes  517 Kbits/sec  0.619 ms  0/44 (0%)
[ 1] 4.0000-5.0000 sec  61.7 KBytes  506 Kbits/sec  0.914 ms  0/43 (0%)
[ 1] 5.0000-6.0000 sec  63.2 KBytes  517 Kbits/sec  0.863 ms  0/44 (0%)
[ 1] 6.0000-7.0000 sec  61.7 KBytes  506 Kbits/sec  0.640 ms  0/43 (0%)
[ 1] 7.0000-8.0000 sec  63.2 KBytes  517 Kbits/sec  0.551 ms  0/44 (0%)
[ 1] 8.0000-9.0000 sec  61.7 KBytes  506 Kbits/sec  0.695 ms  0/43 (0%)
[ 1] 9.0000-10.0000 sec 63.2 KBytes  517 Kbits/sec  0.769 ms  0/44 (0%)
[ 1] 10.0000-11.0000 sec 61.7 KBytes  506 Kbits/sec  0.750 ms  0/43 (0%)
[ 1] 11.0000-12.0000 sec 63.2 KBytes  517 Kbits/sec  0.698 ms  0/44 (0%)
[ 1] 12.0000-13.0000 sec 63.2 KBytes  517 Kbits/sec  0.780 ms  0/44 (0%)
[ 1] 13.0000-14.0000 sec 61.7 KBytes  506 Kbits/sec  0.471 ms  0/43 (0%)
[ 1] 14.0000-15.0000 sec 63.2 KBytes  517 Kbits/sec  0.502 ms  0/44 (0%)
[ 1] 15.0000-16.0000 sec 61.7 KBytes  506 Kbits/sec  0.667 ms  0/43 (0%)
[ 1] 16.0000-17.0000 sec 63.2 KBytes  517 Kbits/sec  0.891 ms  0/44 (0%)
[ 1] 17.0000-18.0000 sec 61.7 KBytes  506 Kbits/sec  0.633 ms  0/43 (0%)
[ 1] 18.0000-19.0000 sec 63.2 KBytes  517 Kbits/sec  0.456 ms  0/44 (0%)
[ 1] 19.0000-20.0000 sec 61.7 KBytes  506 Kbits/sec  0.990 ms  0/43 (0%)
[ 1] 0.0000-20.0059 sec 1.22 MBytes  512 Kbits/sec  0.957 ms  0/871 (0%)

root@8517281c9120:/tmp# iperf -c 12.1.1.4 -u -i 1 -t 20 -b 500K
Client connecting to 12.1.1.4, UDP port 5001
Sending 1470 byte datagrams, IPG target: 22968.75 us (kalman adjust)
UDP buffer size: 208 KByte (default)
-----
[ 3] local 192.168.72.135 port 46082 connected with 12.1.1.4 port 5001
[ ID] Interval      Transfer      Bandwidth
[ 3] 0.0- 1.0 sec  64.6 KBytes  529 Kbits/sec
[ 3] 1.0- 2.0 sec  63.2 KBytes  517 Kbits/sec
[ 3] 2.0- 3.0 sec  61.7 KBytes  506 Kbits/sec
[ 3] 3.0- 4.0 sec  63.2 KBytes  517 Kbits/sec
[ 3] 4.0- 5.0 sec  61.7 KBytes  506 Kbits/sec
[ 3] 5.0- 6.0 sec  63.2 KBytes  517 Kbits/sec
[ 3] 6.0- 7.0 sec  61.7 KBytes  506 Kbits/sec
[ 3] 7.0- 8.0 sec  63.2 KBytes  517 Kbits/sec
[ 3] 8.0- 9.0 sec  61.7 KBytes  506 Kbits/sec
[ 3] 9.0-10.0 sec  63.2 KBytes  517 Kbits/sec
[ 3] 10.0-11.0 sec  61.7 KBytes  506 Kbits/sec
[ 3] 11.0-12.0 sec  63.2 KBytes  517 Kbits/sec
[ 3] 12.0-13.0 sec  61.7 KBytes  506 Kbits/sec
[ 3] 13.0-14.0 sec  63.2 KBytes  517 Kbits/sec
[ 3] 14.0-15.0 sec  63.2 KBytes  517 Kbits/sec
[ 3] 15.0-16.0 sec  61.7 KBytes  506 Kbits/sec
[ 3] 16.0-17.0 sec  63.2 KBytes  517 Kbits/sec
[ 3] 17.0-18.0 sec  61.7 KBytes  506 Kbits/sec
[ 3] 18.0-19.0 sec  63.2 KBytes  517 Kbits/sec
[ 3] 0.0-20.0 sec  1.22 MBytes  512 Kbits/sec
[ 3] Sent 871 datagrams
[ 3] Server Report:
[ 3] 0.0-20.0 sec  1.22 MBytes  512 Kbits/sec  0.957 ms  0/ 871 (0%)
root@8517281c9120:/tmp#

```

Figure 14: Starting iperf client

## 4.9 Step 11: Undeploy

Finally, the emulator was undeployed, completing Task 2 (Figure 15).

```

tor$ docker-compose down
Stopping rfsim5g-oai-nr-ue ... done
Stopping rfsim5g-oai-gnb ... done
Stopping rfsim5g-oai-ext-dn ... done
Stopping rfsim5g-oai-spgwu ... done
Stopping rfsim5g-oai-smf ... done
Stopping rfsim5g-oai-amf ... done
Stopping rfsim5g-oai-nrf ... done
Stopping rfsim5g-mysql ... done
Removing rfsim5g-oai-nr-ue ... done
Removing rfsim5g-oai-gnb ... done
Removing rfsim5g-oai-ext-dn ... done
Removing rfsim5g-oai-spgwu ... done
Removing rfsim5g-oai-smf ... done
Removing rfsim5g-oai-amf ... done
Removing rfsim5g-oai-nrf ... done
Removing rfsim5g-mysql ... done
Removing network rfsim5g-oai-public-net
Removing network rfsim5g-oai-traffic-net

```

Figure 15: Undeploying the emulator

## 4.10 Conclusion

Task 2 was successfully completed with all steps implemented and verified. Challenges such as script encoding and configuration issues were resolved to ensure proper deployment and operation of the mobile network emulator.