



**UNIVERSITY
OF OULU**

FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING
DEGREE PROGRAMME IN ELECTRONICS AND COMMUNICATIONS ENGINEERING

COMMUNICATIONS NETWORKS I - 521340S

LABORATORY WORK - IMPLEMENTATION

Asanka Amarasinghe, 2307144

Athmajan Vivekananthan, 2305573

Farouk Aouaneche, 2307643

November 2023

CONTENTS

CONTENTS

PREFACE

1	DEPLOYING APPLICATION IN LOCAL SERVER	3
1.1	Tasks	3
1.2	Questions	6
2	DEPLOYMENT ON HOSTED-HYPERVERSOR	9
2.1	Tasks	9
2.2	Questions	11
3	CONTAINER-BASED VIRTUALIZATION	18
3.1	Tasks	18
3.2	Questions	21
4	DISCUSSION	24
5	APPENDICES	25
6	REFERENCES	34

PREFACE

This report is presented as the outcome of the laboratory work to implement an IoT application in three different scenarios. The tasks in each of the scenario are to get the dummy sensor data, send the collected data to a time-series database, store the dummy data into the time-series database, and display them into graph. This IoT application was first deployed in our local machine, and then in a hypervisor and lastly in a containerized environment.

These tasks gave clear idea about how each of the scenarios are implemented in real world and enabled us to understand the challenges these pose. However the discussion questions embedded in the tasks were interesting in a way to bridge the gap between the concepts we learned in the classroom and the practical experience we got during this lab.

1 DEPLOYING APPLICATION IN LOCAL SERVER

In this task, four services are installed in the personal computer to implement the objective of the project using locally hosted services.

Instructions were followed as per given in the task description documentation and following services were installed.

1. InfluxDB

InfluxDB v1.8.10 was installed in to local computer.

2. Grafana

Grafana 10.2.0 was installed to monitor real time dummy data obtained from InfluxDB.

3. CoAP server

CoAP server written using Python was used which came along with the task instructions.

4. CoAP client

CoAP client written using Python was used which came along with the task instructions.

1.1 Tasks

1. Screenshots showing the dummy temperature and humidity data.

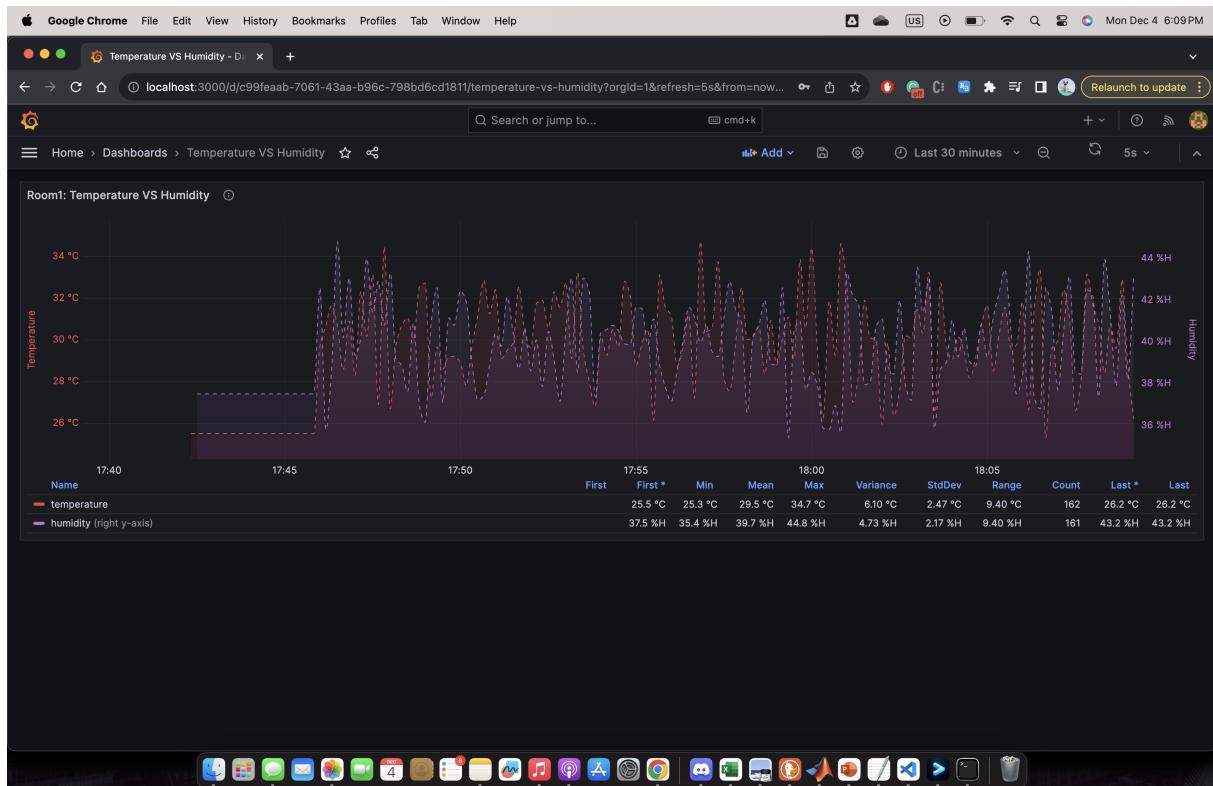


Figure 1.1. Grafana Dashboard for past 30 minutes

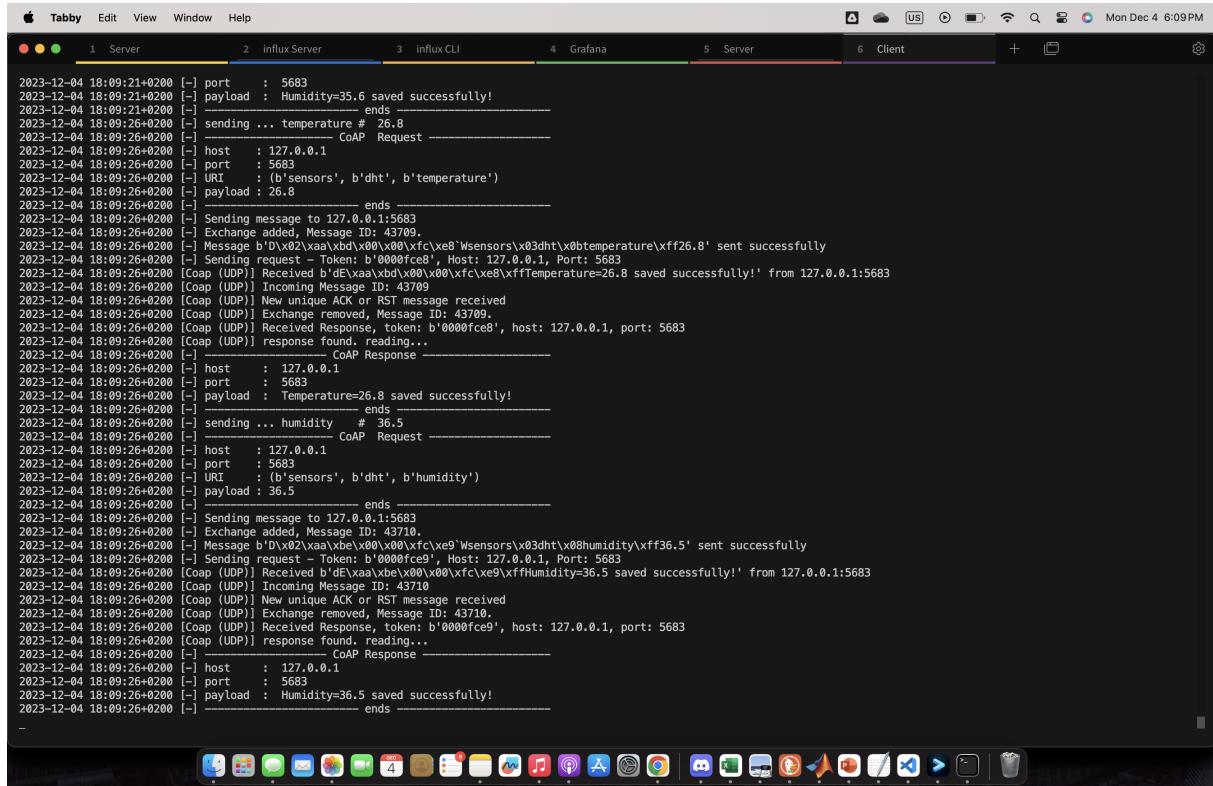


Figure 1.2. Terminal view of Python based local client

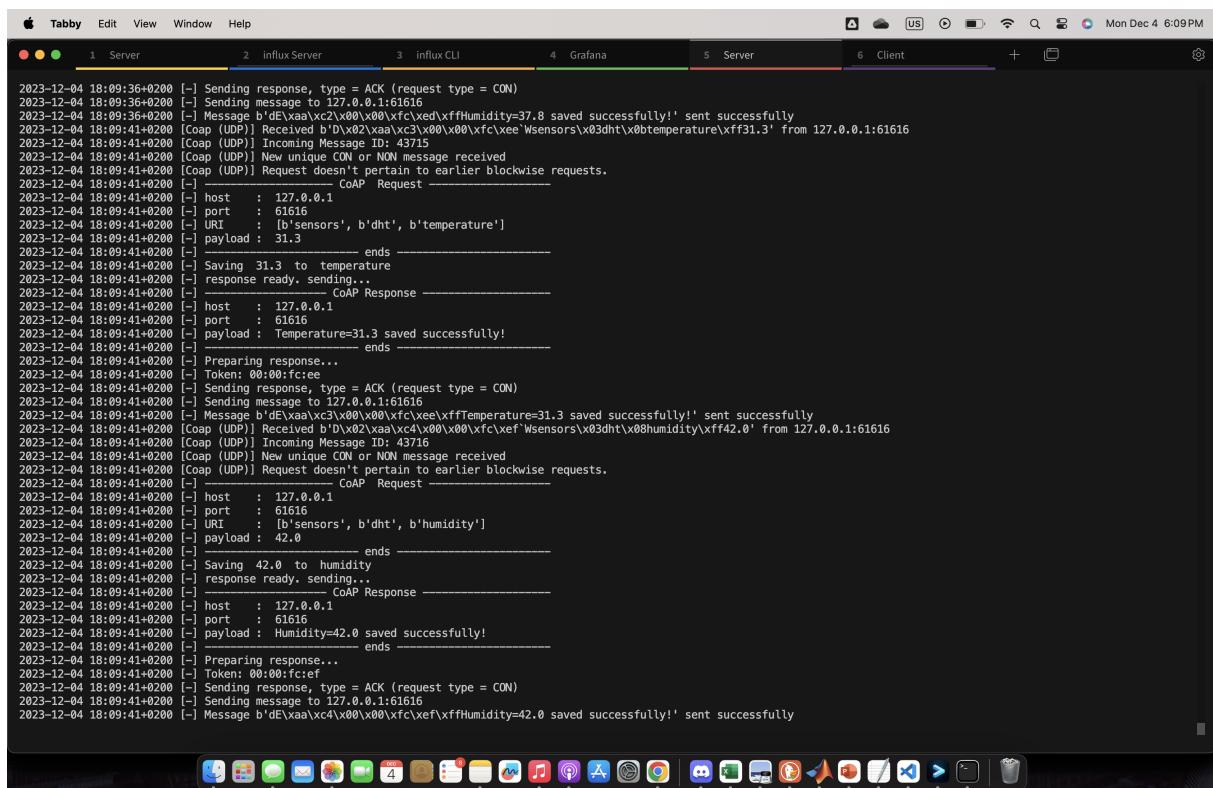


Figure 1.3. Terminal view of Python based local server

```
m1') AND time >= now() - 30m AND time <= now() GROUP BY time(10s) fill(previous)"  
2023-12-04T16:09:55.829107 	info Executing query {"log_id": "0luNpyhG000", "service": "query", "query": "SELECT mean(value) FROM dht.autogen.humidity WHERE (place::tag = 'room1' ) AND time >= now() - 30m AND time <= now() GROUP BY time(10s) fill(previous)"  
[httpd]: ::1, ::1::1 - influxDBUser [04/Dec/2023:18:09:55 +0200] "GET /query?db=dht&epoch=ms&q=SELECT+mean%2822value%22+FROM%42+temperature%22+WHERE%42%22place%27%3A3atag%3D%27 room1%27%29+AND+time%3E%3D+now%28%29+-30m+and+time%3C%3D+now%28%29+GROUP+BY+time%2810s%29+fill%28previous%29%3BSELECT+mean%2822value%22+humidity%22+WHERE%42%22 place%27%3A3atag%3D%27 room1%27%29+AND+time%3E%3D+now%28%29+-30m+and+time%3C%3D+now%28%29+GROUP+BY+time%2810s%29+fill%28previous%29 HTTP/1.1" 200 1646 "-" "Grafana/10.2.2" 907ff165-92bf-11 ee-84cf-3ae59be1157 8618  
2023-12-04T16:09:56.811504Z 	info Executing query {"log_id": "0luNpyhG000", "service": "query", "query": "CREATE DATABASE dht"}  
[httpd] 127.0.0.1 - influxDBUser [04/Dec/2023:18:09:56 +0200] "POST /query?q=CREATE+DATABASE+%22dht%22 HTTP/1.1" 200 58 "-" "python-requests/2.31.0" 913c2a34-92bf-11ee-84d0-3a4e59be11 57 584  
[httpd] 127.0.0.1 - influxDBUser [04/Dec/2023:18:09:56 +0200] "POST /write?db=dht HTTP/1.1" 204 0 "-" "python-requests/2.31.0" 913c915e-92bf-11ee-84d1-3a4e59be1157 6254  
2023-12-04T16:09:56.826514Z 	info Executing query {"log_id": "0luNpyhG000", "service": "query", "query": "CREATE DATABASE dht"}  
[httpd] 127.0.0.1 - influxDBUser [04/Dec/2023:18:09:56 +0200] "POST /query?q=CREATE+DATABASE+%22dht%22 HTTP/1.1" 200 58 "-" "python-requests/2.31.0" 913c767c-92bf-11ee-84d2-3a4e59be11 57 485  
[httpd] 127.0.0.1 - influxDBUser [04/Dec/2023:18:09:56 +0200] "POST /write?db=dht HTTP/1.1" 204 0 "-" "python-requests/2.31.0" 913eb984-92bf-11ee-84d3-3a4e59be1157 4006  
2023-12-04T16:10:00.565387Z 	info Executing query {"log_id": "0luNpyhG000", "service": "query", "query": "SELECT mean(value) FROM dht.autogen.temperature WHERE (place::tag = 'room1') AND time >= now() - 30m AND time <= now() GROUP BY time(10s) fill(previous)"  
[httpd] 127.0.0.1 - influxDBUser [04/Dec/2023:10:00:571667Z 	info Executing query {"log_id": "0luNpyhG000", "service": "query", "query": "SELECT mean(value) FROM dht.autogen.humidity WHERE (place::tag = 'room1') AND time >= now() - 30m AND time <= now() GROUP BY time(10s) fill(previous)"  
[httpd]: ::1, ::1::1 - influxDBUser [04/Dec/2023:18:10:01 +0200] "GET /query?db=dht&epoch=ms&q=SELECT+mean%2822value%22+FROM%42+temperature%22+WHERE%42%22place%22%3A3atag%3D%27 room1%27%29+AND+time%3E%3D+now%28%29+-30m+and+time%3C%3D+now%28%29+GROUP+BY+time%2810s%29+fill%28previous%29%3BSELECT+mean%2822value%22+humidity%22+WHERE%42%22 place%27%3A3atag%3D%27 room1%27%29+AND+time%3E%3D+now%28%29+-30m+and+time%3C%3D+now%28%29+GROUP+BY+time%2810s%29+fill%28previous%29 HTTP/1.1" 200 1652 "-" "Grafana/10.2.2" 9378aca9-92bf-11 ee-84d4-3ae59be1157 1786  
2023-12-04T16:10:01.843462Z 	info Executing query {"log_id": "0luNpyhG000", "service": "query", "query": "CREATE DATABASE dht"}  
[httpd] 127.0.0.1 - influxDBUser [04/Dec/2023:18:10:01 +0200] "POST /query?q=CREATE+DATABASE+%22dht%22 HTTP/1.1" 200 58 "-" "python-requests/2.31.0" 943c01a0-92bf-11ee-84d5-3a4e59be11 57 239  
[httpd] 127.0.0.1 - influxDBUser [04/Dec/2023:18:10:01 +0200] "POST /write?db=dht HTTP/1.1" 204 0 "-" "python-requests/2.31.0" 943c31d4-92bf-11ee-84d6-3a4e59be1157 7727  
2023-12-04T16:10:01.855562Z 	info Executing query {"log_id": "0luNpyhG000", "service": "query", "query": "CREATE DATABASE dht"}  
[httpd] 127.0.0.1 - influxDBUser [04/Dec/2023:18:10:01 +0200] "POST /query?db=dht&epoch=ms&q=SELECT+mean%2822value%22+FROM%42+temperature%22+WHERE%42%22place%22%3A3atag%3D%27 room1%27%29+AND+time%3E%3D+now%28%29+-30m+and+time%3C%3D+now%28%29+GROUP+BY+time%2810s%29+fill%28previous%29%3BSELECT+mean%2822value%22+humidity%22+WHERE%42%22 place%27%3A3atag%3D%27 room1%27%29+AND+time%3E%3D+now%28%29+-30m+and+time%3C%3D+now%28%29+GROUP+BY+time%2810s%29+fill%28previous%29 HTTP/1.1" 200 1652 "-" "Grafana/10.2.2" 9378aca9-92bf-11 ee-84d4-3ae59be1157 1786  
2023-12-04T16:10:01.876769Z 	info Executing query {"log_id": "0luNpyhG000", "service": "query", "query": "CREATE DATABASE dht"}  
[httpd] 127.0.0.1 - influxDBUser [04/Dec/2023:18:10:01 +0200] "POST /query?q=CREATE+DATABASE+%22dht%22 HTTP/1.1" 200 58 "-" "python-requests/2.31.0" 943dd9da-92bf-11ee-84d7-3a4e59be11 57 577  
[httpd] 127.0.0.1 - influxDBUser [04/Dec/2023:18:10:01 +0200] "POST /write?db=dht HTTP/1.1" 204 0 "-" "python-requests/2.31.0" 943e037e-92bf-11ee-84d8-3a4e59be1157 4910  
2023-12-04T16:10:05.576995Z 	info Executing query {"log_id": "0luNpyhG000", "service": "query", "query": "SELECT mean(value) FROM dht.autogen.temperature WHERE (place::tag = 'room1') AND time >= now() - 30m AND time <= now() GROUP BY time(10s) fill(previous)"  
[httpd] 127.0.0.1 - influxDBUser [04/Dec/2023:18:10:05.581397Z 	info Executing query {"log_id": "0luNpyhG000", "service": "query", "query": "SELECT mean(value) FROM dht.autogen.humidity WHERE (place::tag = 'room1') AND time >= now() - 30m AND time <= now() GROUP BY time(10s) fill(previous)"  
[httpd]: ::1, ::1::1 - influxDBUser [04/Dec/2023:18:10:05 +0200] "GET /query?db=dht&epoch=ms&q=SELECT+mean%2822value%22+FROM%42+temperature%22+WHERE%42%22place%22%3A3atag%3D%27 room1%27%29+AND+time%3E%3D+now%28%29+-30m+and+time%3C%3D+now%28%29+GROUP+BY+time%2810s%29+fill%28previous%29%3BSELECT+mean%2822value%22+humidity%22+WHERE%42%22 place%27%3A3atag%3D%27 room1%27%29+AND+time%3E%3D+now%28%29+-30m+and+time%3C%3D+now%28%29+GROUP+BY+time%2810s%29+fill%28previous%29 HTTP/1.1" 200 1654 "-" "Grafana/10.2.2" 96758f68-92bf-11 ee-84d9-3ae59be1157 8713  
2023-12-04T16:10:06.876769Z 	info Executing query {"log_id": "0luNpyhG000", "service": "query", "query": "CREATE DATABASE dht"}  
[httpd] 127.0.0.1 - influxDBUser [04/Dec/2023:18:10:06 +0200] "POST /query?q=CREATE+DATABASE+%22dht%22 HTTP/1.1" 200 58 "-" "python-requests/2.31.0" 973c00da-92bf-11ee-84da-3a4e59be11 57 577  
[httpd] 127.0.0.1 - influxDBUser [04/Dec/2023:18:10:06 +0200] "POST /write?db=dht HTTP/1.1" 204 0 "-" "python-requests/2.31.0" 973c6034-92bf-11ee-84db-3a4e59be1157 7933  
2023-12-04T16:10:06.891483Z 	info Executing query {"log_id": "0luNpyhG000", "service": "query", "query": "CREATE DATABASE dht"}  
[httpd] 127.0.0.1 - influxDBUser [04/Dec/2023:18:10:06 +0200] "POST /query?db=dht&epoch=ms&q=SELECT+mean%2822value%22+FROM%42+temperature%22+WHERE%42%22place%22%3A3atag%3D%27 room1%27%29+AND+time%3E%3D+now%28%29+-30m+and+time%3C%3D+now%28%29+GROUP+BY+time%2810s%29+fill%28previous%29%3BSELECT+mean%2822value%22+humidity%22+WHERE%42%22 place%27%3A3atag%3D%27 room1%27%29+AND+time%3E%3D+now%28%29+-30m+and+time%3C%3D+now%28%29+GROUP+BY+time%2810s%29+fill%28previous%29 HTTP/1.1" 200 1654 "-" "Grafana/10.2.2" 973e41ce-92bf-11ee-84dc-3a4e59be11 57 436  
[httpd] 127.0.0.1 - influxDBUser [04/Dec/2023:18:10:06 +0200] "POST /write?db=dht HTTP/1.1" 204 0 "-" "python-requests/2.31.0" 973e8850-92bf-11ee-84dd-3a4e59be1157 4597
```

Figure 1.4. Terminal view of InfluxDB server

```
> SHOW DATABASES  
name: databases  
name:  
internal  
dht  
> USE dht  
Using database dht  
> SHOW MEASUREMENTS  
> INSERT temperature,place=room1 value=25.5  
> INSERT humidity,place=room1 value=37.5  
> SHOW MEASUREMENTS  
name: measurements  
name:  
humidity  
temperature  
> SELECT COUNT(*) FROM humidity  
name: humidity  
time count_value  
-----  
0 1  
> SELECT COUNT(*) FROM humidity  
name: humidity  
time count_value  
-----  
0 10  
> SELECT COUNT(*) FROM humidity  
name: humidity  
time count_value  
-----  
0 10  
> SELECT COUNT(*) FROM humidity  
name: humidity  
time count_value  
-----  
0 11  
> SELECT COUNT(*) FROM humidity  
name: humidity  
time count_value  
-----  
0 11  
> SELECT COUNT(*) FROM humidity  
name: humidity  
time count_value  
-----  
0 284  
> _
```

Figure 1.5. Terminal view of InfluxDB CLI

2. Make a table and write down the memory consumption of the services used in this lab work.

Table 1.1. Processes Information

Service	PID	%CPU	%MEM	VSZ (/Bytes)
InfluxDB	76474	0.0	0.2	34059300
Grafana	74874	0.0	0.7	410713744
Python Client	76734	0.0	0.1	408569520
Python Server	76636	0.0	0.2	409443264

```
(base) athmajanvivekananthan@Athmajans-MacBook-Pro client % ps aux | grep -E '76474|74874|76734|76636'
athmajanvivekananthan 76474  0.2  0.2 34059300 33172 s006 S+    5:36PM  0:19.71 influxd
athmajanvivekananthan 74874  0.2  0.4 410713744 60352 ?? S    4:34PM  0:21.71 /opt/homebrew/opt/grafana/bin/grafana ser
ath /opt/homebrew/opt/grafana/share/grafana --packaging=brew cfg:default.paths.logs=/opt/homebrew/var/log/grafana cfg:default.
.plugins=/opt/homebrew/var/lib/grafana/plugins
athmajanvivekananthan 76734  0.0  0.1 408569520 14944 s013 S+    5:46PM  0:03.25 python3 client.py
athmajanvivekananthan 76636  0.0  0.2 408918976 26688 s011 S+    5:43PM  0:09.75 python3 server.py
athmajanvivekananthan 81034  0.0  0.0 408112816 1296 s019 S+    6:47PM  0:00.00 grep -E 76474|74874|76734|76636
(base) athmajanvivekananthan@Athmajans-MacBook-Pro client % _
```

Figure 1.6. Memory Usage of Processes

1.2 Questions

1. Define the cloud, fog (edge) and mist computing.

Cloud Computing, Fog Computing (Edge Computing), and Mist Computing are different ways of handling data in the world of computing. Here's a simplified breakdown:

Cloud Computing: Cloud computing can be considered as an online computer service where storage space, processing power, and applications are online. Whenever a service is needed, it can be taken from the internet without own it or managing the actual equipment.

Fog Computing (Edge Computing): Fog computing, also known as edge computing, It is kind of bringing the power of the cloud computing closer to where it is needed. Instead of sending all requests far away to a central location (like the cloud), some of the work is done nearby, at the edge of the network. This makes things happen faster and reduces the network traffic.

Mist Computing: Mist computing is the computing in sensors and micro-controllers used in smarts objects in the Internet of Things (IoT). Instead of relying on a central place (cloud) or a nearby one (fog), these smart devices do some thinking on their own. Mist computing provides smart objects smarter right where they are, without always needing help from far away. It's handy for things that need quick decisions, like industrial IoT applications.

2. Explain monolith, microservices and nanoservices architecture. Explain why nanoservices are important for IoT networks.

Monolith Architecture: In a monolith architecture, an entire application is built as a single, tightly integrated unit. All components and functions of the application, such as user interface, business logic, and database access, are packaged and deployed together. Key characteristics are,

- Single, unified codebase.
- All functionalities are tightly coupled.
- Scaling usually involves replicating the entire application.

Microservices Architecture:

Microservices architecture breaks down an application into a collection of small, independent services, each focused on a specific business capability. These services can run and be scaled independently, communicate through well-defined APIs, and often use their own databases. Key characteristics are,

- Distributed architecture with modular and independent services.
- Each service is responsible for a specific business capability.
- Services communicate through APIs, promoting flexibility and scalability.

Nanoservices Architecture: Overview: Nanoservices architecture takes the concept of microservices to an even smaller scale. In nanoservices, each service is extremely fine-grained, focusing on a very specific function. The idea is to break down services into the smallest possible components for increased flexibility and composability. Key characteristics are,

- Extremely small and specialized services.
- Granular level of functionality for each service.
- Offers high flexibility and ease of composition.

Nanoservices are crucial for IoT networks because they offer lightweight and specialized functions, addressing specific needs in a simple and efficient way. In IoT, where devices often have limited resources, nanoservices optimize resource usage and allow for flexible scaling. Their modular nature promotes interoperability among diverse devices, and their small size enables quick decision-making, making them suitable for real-time applications. Nanoservices also align well with edge computing, bringing computation closer to where data is generated. Overall, their simplicity enhances security by isolating functions, reducing potential vulnerabilities.

3. Define IoT. Explain working principle of MQTT and CoAP protocols.

IoT stands for the "Internet of Things." It refers to a network of interconnected physical devices, vehicles, appliances, and other objects embedded with sensors, software, and network connectivity. These devices collect and exchange data, allowing them to communicate and interact with each other via the internet. The primary goal of IoT is to enable these devices to gather and share information, automate processes, and provide valuable insights, often leading to enhanced efficiency, convenience, and decision-making in various aspects of everyday life and industrial processes. In simple manner, IoT can

be explained as connecting unconnected things to the internet to provides services in immersive manner.

Both CoAP and MQTT are communication protocols designed for the Internet of Things (IoT). Contrast with traditional client-server architecture, these two are in the publish-subscribe model.

The working principle of MQTT

MQTT typically operates over TCP, providing reliable and ordered communication and it is primarily focused on the exchange of messages. In MQTT architecture, there are three parities known as publishers, brokers and subscribers. Information are exchanged in forms of messages where message consists of topic and content. Publishers such as sensors can sent their sensor data as messages to the Central location known as broker. Then broker forwards the message only to the subscribers such as actuators if they have subscribed the topic. This model is well-suited for applications requiring real-time telemetry, monitoring, and remote sensing. In other hand MQTT does not provide direct interaction between devices where communication only through brokers.

The working principle of CoAP

CoAP operates over UDP, emphasizing a connectionless and lightweight approach. This choice is particularly advantageous in constrained networks and devices with limited resources, as it minimizes the overhead associated with connection establishment. CoAP supports both request-response interactions and asynchronous communication. Its design aligns with RESTful architectures, allowing direct interaction with resources and making it suitable for scenarios where devices need to communicate directly. Therefore, COAP is well-suited for applications involving the monitoring and control of IoT devices.

In conclusion, the choice between CoAP and MQTT depends on the specific requirements of an IoT application. CoAP suits scenarios that demand direct resource interaction and adherence to RESTful principles, while MQTT excels in efficient message broadcasting to multiple subscribers, making it well-suited for real-time monitoring and telemetry applications.

2 DEPLOYMENT ON HOSTED-HYPERVISOR

In the chapter discussions about task 2 activities are presented. In task 2, virtualization of the deployment done in task 1 is achieved by the usage of hypervisor.

2.1 Tasks

1. Monitor the application on the guest OS.

2.1 shows the Grafana dashboard with dummy data connected to InfluxDB data source.

The figures in Appendix section 5.1-5.4 show the screenshots of terminals of each of the services deployed in the guest OS.

The figures in Appendix 5.5 - 5.8 present the observations made when monitoring the memory usage of application on the guest OS. Observations made on the host OS also have been presented in the figure 5.9 in Appendix.

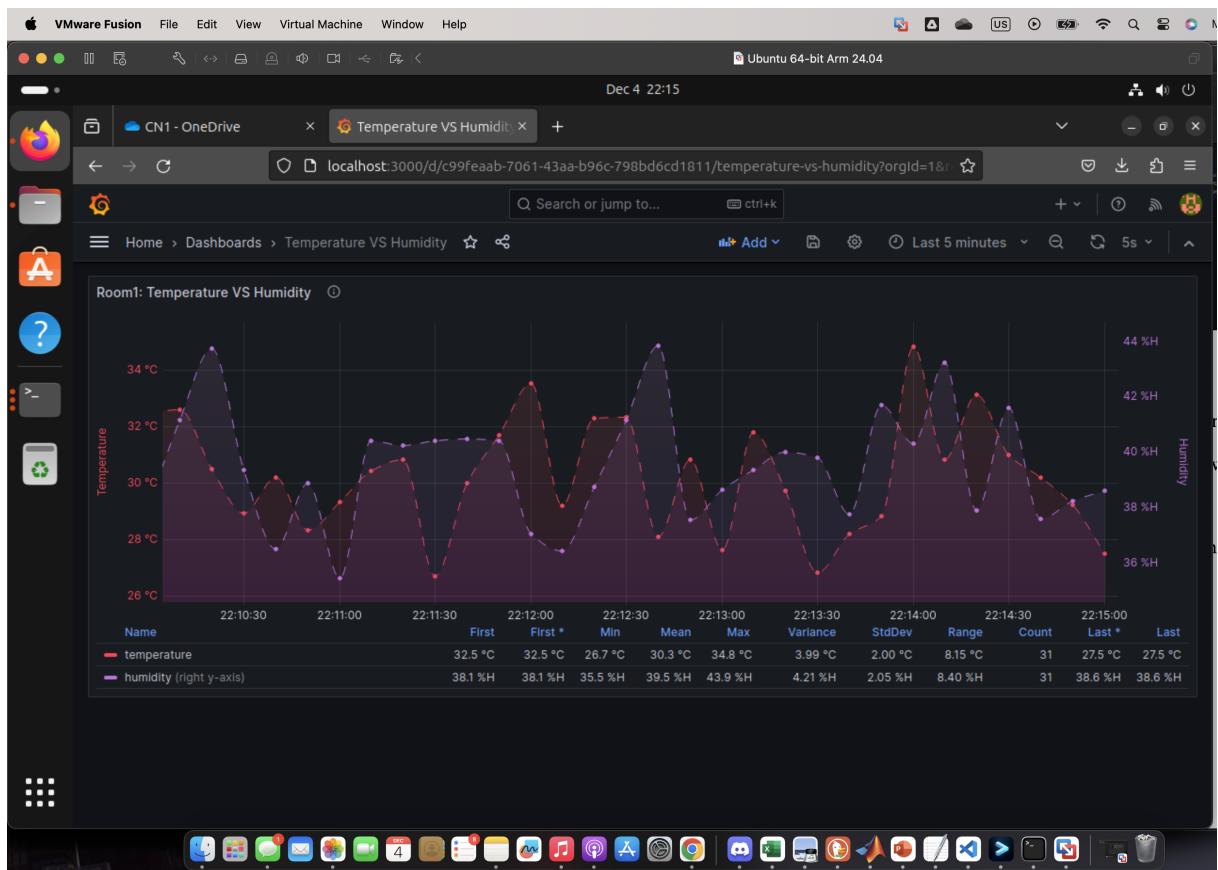


Figure 2.1. Grafana Dashboard as views on the Guest OS

2. Export the guest OS in OVA format along with the applications.

Hypervisor settings for virtualization of the deployment are as follows. Screenshots of this section are attached as figures 5.10 - 5.11 in Appendix 3. Configurations used for the

VM are included in Table 2.1. Information about the OVA file is included in Table 2.2. Screenshot of exported OVA file and settings can be found in figures 2.3, 2.2.

Configuration	Value
Guest OS	Ubuntu 64-bit Arm 24.04
Disk Size	20GB
Number of Processor Cores	2
RAM	4096MB

Table 2.1. Virtual Machine Configuration

Information	Details
File Size	(5869613056 Bytes)
OS Included	Ubuntu Ubuntu 22.04
Allocated CPU(s)	2
Required Memory	4096MB

Table 2.2. OVA File Information

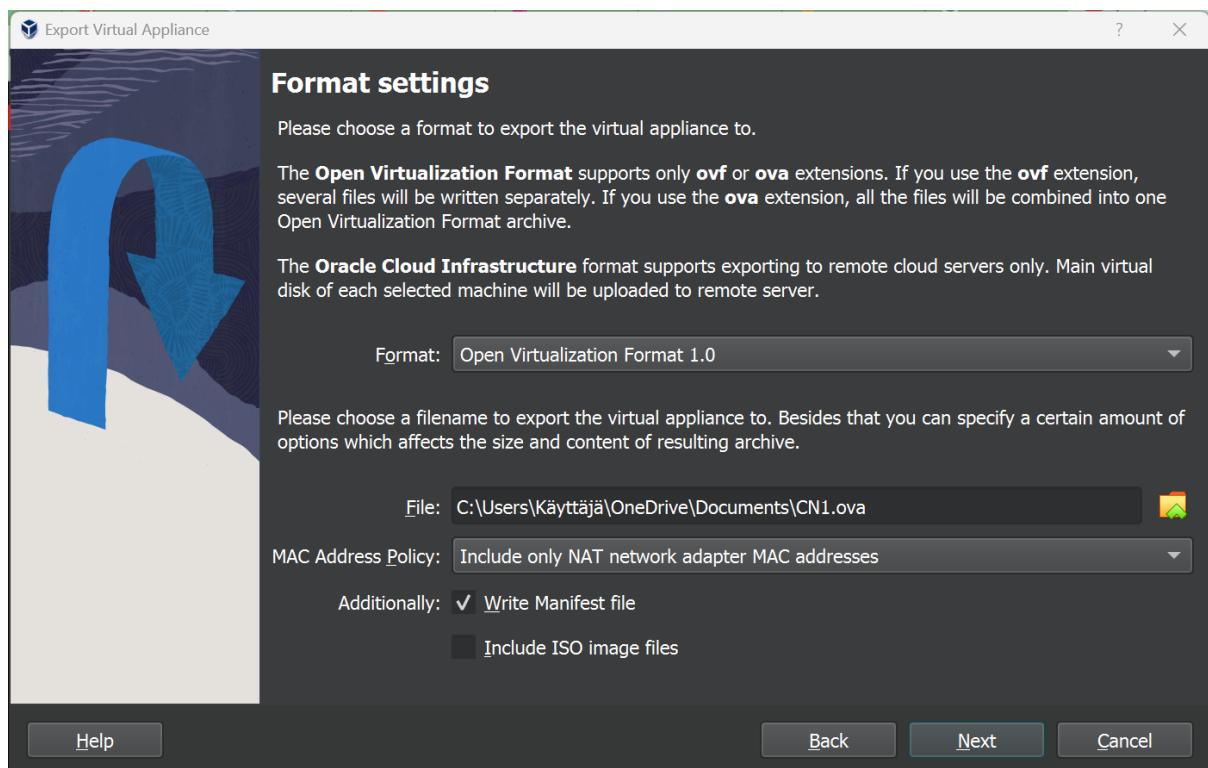


Figure 2.2. Export to OVA settings.

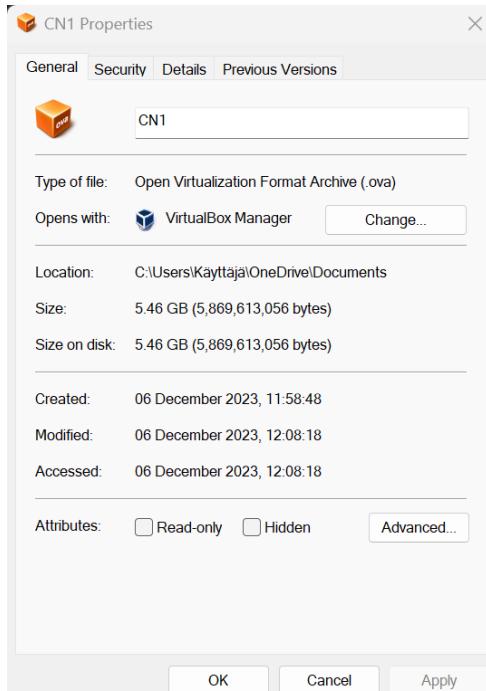


Figure 2.3. OVA File Size

2.2 Questions

4. Explain pros and cons of guest OS?

Advantages

One of the biggest advantage of using guest OS is the isolation it provides between multiple applications, allowing to run multiple operating systems. Therefore when deploying applications that are developed to run on different operating systems can be deployed on the same hardware.

This also proves the advantage of scalability since applications isolated from the hardware. Therefore during scaling an application deployment it is seldom required to upgrade the hardware.

The second best advantage in using a guest OS is that it is less vulnerable to security threats compared to other virtualization methods that exist such as containerization. As it was discussed above, since there is a separation between each of the applications, this security is further enhanced. Also applications deployed on guest OS do not share the kernel of the host OS. Therefore applications are less vulnerable from attacks launched on host OS.

Guest OS allows to save a specific state of workspace as a snapshot which allows us to replicate the same work without much effort. The guest OS exported file can also be used on another host OS as a clone without having to deploy a service from the scratch.

Disadvantages

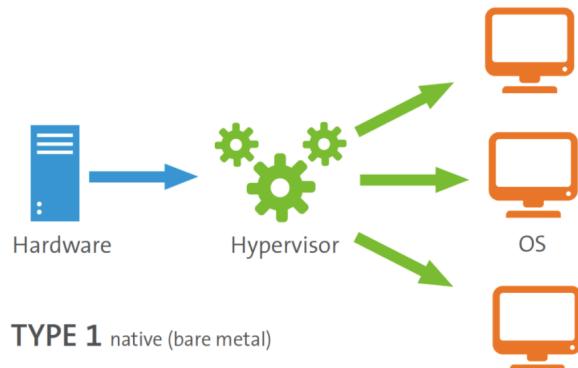


Figure 2.4. Hypervisor Type 1

Guest OS largely depends on the hypervisor for resource management. Therefore any issues faced at the hypervisor completely affects the guest OS and its deployed applications.

During the practical work it was understood that there are multiple possibilities of virtualization platforms available for guest OS. Some of them are paid version and some of them are open source. The problem we faced with Apple's silicon M1 processors (discussed in detail in later sections) proves this disadvantage that there is no open source community support for certain hypervisors after Apple launched the Silicon processors. Some of them have beta test versions but almost all of them are not industry ready. Therefore someone who had developed an application on Apple Intel processors guest OS and wanted to clone or scale it in to Silicon processors will not have any product support. This dependency is hugely disadvantageous.

When hosting multiple operating systems, each guest OS have their own CPU, Memory and Disk Space allocation from the host. Therefore this may not be an optimum solution for deployment of multiple applications. This also adds a huge overhead leading to poor performances.

5. Differentiate Type 1 and Type 2 hypervisor with necessary figures?

Type 1 hypervisor directly runs on the host's hardware as shown in the Figure 2.4 to control the host's hardware. It also monitors guest operating systems that run on a separate level above the hypervisor. Some well-known examples for type 1 hypervisors are Oracle VM, Microsoft Hyper-V, VMWare ESX and Xen.

Type 2 hypervisor runs on a conventional operating system, just as other computer programs as shown in the Figure 2.5. In this type, the guest operating system becomes a third software level above the hardware. Some Well-known examples for type 2 hypervisors are Oracle VM VirtualBox, VMWare Server and Workstation, Microsoft Virtual PC, KVM, QEMU etc.

Type 1 hypervisor has more controllability of host's hardware compared with the type 2 hypervisor, but on the other hand type 2 hypervisor is easier to implement as it runs on host's OS compared.

[1]

6. What is shell scripting and why it is important? Make a shell script to run the 4

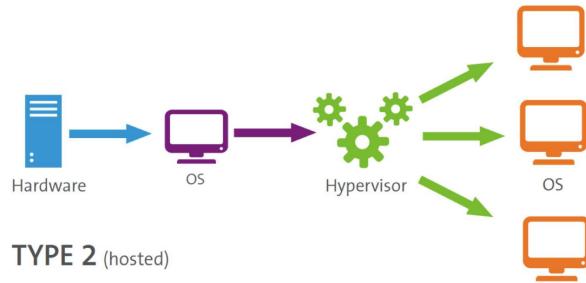


Figure 2.5. Hypervisor Type 2

services into your guest OS and explain the commands in a few sentences.

Shell scripting is a way of commanding the operating system to execute a sequence of tasks. The shell can be defined as an interface which connects humans and the computer OS. The command line interpreter takes shell commands and interprets them so that the computer OS can understand them as machine code and execute the given commands.

In the context of virtualization of deployments of applications, shell scripting plays an important role in bringing in the compatibility with cross platforms where the shell scripts given can be executed on different Unix based systems without having to code the commands from the beginning. In addition to that as mentioned in the first paragraph, shell scripting allows us to provide the commands in a particular sequence so that we have a control on the flow of tasks executed.

Finally, shell scripting allows us to automate and schedule the tasks that need to be executed in the OS without having to interfere all the time. This provides huge flexibility when operating with system virtualization [2]. A shell script to run the 4 services in our project on a Ubuntu guest OS is given in snippet listing - 6. Since the format is not much clear when reported in the text format, a screenshot of the script is attached in Figure 2.6.

Listing 2.1. Shell Script to Run 4 Services in Guest OS

```
#!/bin/bash

# Determine the location of Python3
PYTHON3_LOCATION=$(command -v python3)

# Check if Python is installed.
if [ -z "$PYTHON3_LOCATION" ]; then
    echo "Python3 is not installed. Please install Python3."
    exit 1
fi

# Add Python3 location to PATH
export PATH="$PWD:$PYTHON3_LOCATION:$PATH"
```

```

# Check if ports needed for DHT deployment are already in use
if lsof -i :3000 >/dev/null; then
    echo "Port 3000 is already in use. Please free up the
        port and try again."
    exit 1
fi

if lsof -i :8086 >/dev/null; then
    echo "Port 8086 is already in use. Please free up the
        port and try again."
    exit 1
fi

if lsof -i :5683 >/dev/null; then
    echo "Port 5683 is already in use. Please free up the
        port and try again."
    exit 1
fi

if lsof -i :61616 >/dev/null; then
    echo "Port 61616 is already in use. Please free up the
        port and try again."
    exit 1
fi

# Start Grafana server
sudo systemctl start grafana-server

# Check and install Python dependencies for client
if ! python3 -m pip show -q -r ./client/requirements.txt;
then
    echo "Installing Python dependencies for client"
    python3 -m pip install -r ./client/requirements.txt
fi

# Check and install Python dependencies for server
# Assuming separate requirements file for server is named '
server_requirements.txt'
if ! python3 -m pip show -q -r ./server/server_requirements.
    txt; then
    echo "Installing Python dependencies for server"
    python3 -m pip install -r ./server/server_requirements.
        txt
fi

# Change directory to influxdb subfolder
cd influxdb

```

```
# Run the InfluxDB daemon
sudo .influxd

# Change back to the original directory
cd ..
```

Explanation for the codes included in the shell script can be found in the table 2.3.

Table 2.3. Explanation of Key Commands and Options in the Shell Script

Command	Explanation
bin/bash	Shebang line specifying the script should be executed using the Bash shell.
command -v python3	-v: Verbose option, prints the path of the specified command (Python3 in this case).
export PATH="\$(dirname \$PYTHON3_LOCATION):\${PATH}"	export: Sets the PATH environment variable, adding the directory containing Python3 to it.
lsof -i :port	lsof: Lists open files; -i: Specifies the Internet address details; :port: Filters by the specified port.
sudo systemctl start grafana-server	sudo: Executes a command with superuser privileges; systemctl start: Starts a system service (Grafana server in this case).
if ! python3 -m pip show -q -r ./client/requirements.txt; then	Conditional statement: if checks the result of the pip show command; -q: Quiet mode (no output); -r ./client/requirements.txt: Specifies the requirements file.
python3 -m pip install -r ./client/requirements.txt	python3 -m pip: Invokes the Python3 package manager; install -r: Installs packages listed in a requirements file.
cd influxdb	cd: Changes the current working directory to influxdb.
sudo influxd	sudo: Executes the command with superuser privileges; influxd: Runs the InfluxDB daemon.
cd ..	cd: Changes the current working directory back to the parent directory.

```

1  #!/bin/bash
2
3  # Determine the location of Python3
4  PYTHON3_LOCATION=$(command -v python3)
5
6  # Check if Python is installed.
7  if [ -z "$PYTHON3_LOCATION" ]; then
8      echo "Python3 is not installed. Please install Python3."
9      exit 1
10 fi
11
12 # Add Python3 location to PATH
13 export PATH="$(dirname $PYTHON3_LOCATION):$PATH"
14
15 # Check if ports needed for DHT deployment are already in use
16 if lsof -i :3000 >/dev/null; then
17     echo "Port 3000 is already in use. Please free up the port and try again."
18     exit 1
19 fi
20
21 if lsof -i :8086 >/dev/null; then
22     echo "Port 8086 is already in use. Please free up the port and try again."
23     exit 1
24 fi
25
26 if lsof -i :5683 >/dev/null; then
27     echo "Port 5683 is already in use. Please free up the port and try again."
28     exit 1
29 fi
30
31 if lsof -i :61616 >/dev/null; then
32     echo "Port 61616 is already in use. Please free up the port and try again."
33     exit 1
34 fi
35
36 # Start Grafana server
37 sudo systemctl start grafana-server
38
39 # Check and install Python dependencies for client
40 if ! python3 -m pip show -q -r ./client/requirements.txt; then
41     echo "Installing Python dependencies for client"
42     python3 -m pip install -r ./client/requirements.txt
43 fi
44
45 # Check and install Python dependencies for server
46 # Assuming separate requirements file for server is named 'server_requirements.txt'
47 if ! python3 -m pip show -q -r ./server/server_requirements.txt; then
48     echo "Installing Python dependencies for server"
49     python3 -m pip install -r ./server/server_requirements.txt
50 fi
51
52 # Change directory to influxdb subfolder
53 cd influxdb
54
55 # Run the InfluxDB daemon
56 sudo ./influxd
57
58 # Change back to the original directory
59 cd ..
60

```

Figure 2.6. Screenshot of the shell script

7. **Make a summary of memory consumption with a table. It may include installation size of the hypervisor, guest OS (in ova format) and all 4 services and their dependencies.**

Table 2.4. Summary of Memory Consumption

Component	Memory Consumption in Bytes
Hypervisor in Host OS	409436160
CoAP Client on Guest OS	51244
CoAP Server on Guest OS	61188
InfluxDB on Guest OS	218144
Influx on Guest OS	144208
Grafana on Guest OS	1534484
Firefox on Guest OS (Sum of all processes)	27917676

Table 2.5. Summary of File Sizes

Component	File Size
Hypervisor Installation Size	1.51GB
Guest OS (OVA) File Size	5.46GB

3 CONTAINER-BASED VIRTUALIZATION

3.1 Tasks

1. Make an yml or yaml formatted config file (`docker-compose.yml`) so that the application could be run through docker compose command.

The configuration can be found in code snippet listing - 3.1. A streamlined view can be found as a screenshot in Figure 3.1.

```

1 version: '3'
2 services:
3   dht_coap_client:
4     image: dht_coap_client:1.0
5     network_mode: "host"
6     ports:
7       - "61616:61616"
8
9   dht_coap_server:
10    image: dht_coap_server:1.0
11    network_mode: "host"
12    ports:
13      - "5683:5683"
14
15 grafana:
16   image: grafana/grafana:10.2.0
17   ports:
18     - "3000:3000"
19
20 influxdb:
21   image: influxdb:1.8.10
22   ports:
23     - "8086:8086"
24   environment:
25     - INFLUXDB_DB=dht
26     - INFLUXDB_ADMIN_USER=influxDBuser
27     - INFLUXDB_ADMIN_PASSWORD=influxDBpass
28     - INFLUXDB_HTTP_AUTH_ENABLED=true

```

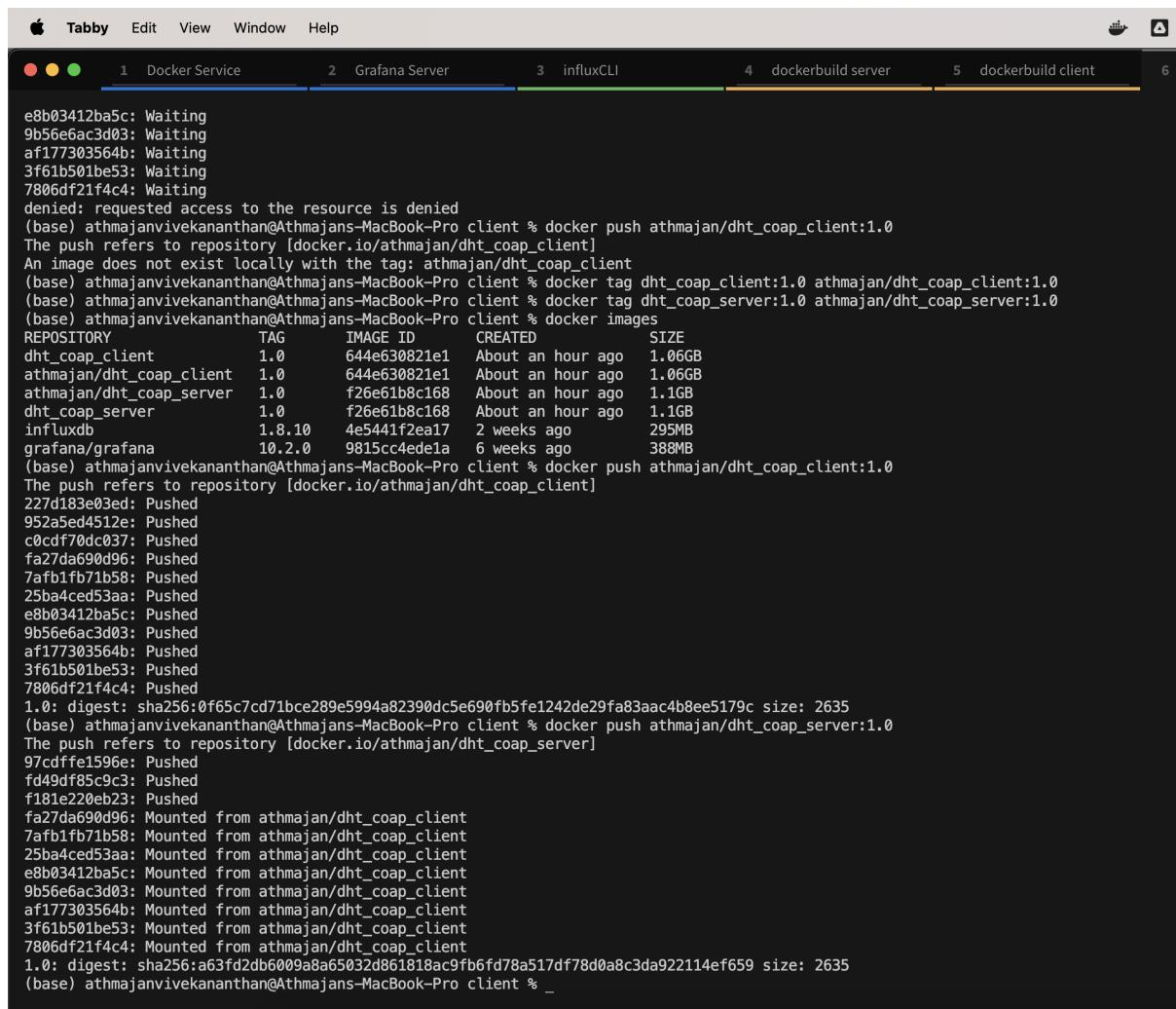
Listing 3.1. yml configuration for docker composition

```
docker-compose.yml
Users > athmajanvivekananthan > WCE > Communications Network 1 > Lab Work > task3 > docker-compose.yml
1 version: '3'
2 services:
3   dht_coap_client:
4     image: dht_coap_client:1.0
5     network_mode: "host"
6     ports:
7       - "61616:61616"
8
9   dht_coap_server:
10    image: dht_coap_server:1.0
11    network_mode: "host"
12    ports:
13      - "5683:5683"
14
15   grafana:
16     image: grafana/grafana:10.2.0
17     ports:
18       - "3000:3000"
19
20   influxdb:
21     image: influxdb:1.8.10
22     ports:
23       - "8086:8086"
24     environment:
25       - INFLUXDB_DB=dht
26       - INFLUXDB_ADMIN_USER=influxDBuser
27       - INFLUXDB_ADMIN_PASSWORD=influxDBpass
28       - INFLUXDB_HTTP_AUTH_ENABLED=true
29
```

Figure 3.1. Screenshot of YML file

2. **Create an account to Docker Hub and push your CoAP server and client images to the Docker-Hub repository.**

Screenshots of pushed images to Docket Hub can be found in Figures 3.2-3.3.



The screenshot shows a Tabby terminal window with five tabs open. The active tab, labeled '2 Grafana Server', contains the following terminal session:

```

e8b03412ba5c: Waiting
9b56e6ac3d03: Waiting
af177303564b: Waiting
3f61b501be53: Waiting
7806df21f4c4: Waiting
denied: requested access to the resource is denied
(base) athmajanvivekananthal@Asthmajans-MacBook-Pro client % docker push athmajan/dht_coap_client:1.0
The push refers to repository [docker.io/athmajan/dht_coap_client]
An image does not exist locally with the tag: athmajan/dht_coap_client
(base) athmajanvivekananthal@Asthmajans-MacBook-Pro client % docker tag dht_coap_client:1.0 athmajan/dht_coap_client:1.0
(base) athmajanvivekananthal@Asthmajans-MacBook-Pro client % docker tag dht_coap_server:1.0 athmajan/dht_coap_server:1.0
(base) athmajanvivekananthal@Asthmajans-MacBook-Pro client % docker images
REPOSITORY          TAG      IMAGE ID      CREATED        SIZE
dht_coap_client    1.0      644e630821e1  About an hour ago  1.06GB
athmajan/dht_coap_client  1.0      644e630821e1  About an hour ago  1.06GB
athmajan/dht_coap_server  1.0      f26e61b8c168  About an hour ago  1.1GB
dht_coap_server    1.0      f26e61b8c168  About an hour ago  1.1GB
influxdb           1.8.10   4e5441f2ea17  2 weeks ago    295MB
grafana/grafana    10.2.0   9815cc4eed1a  6 weeks ago    388MB
(base) athmajanvivekananthal@Asthmajans-MacBook-Pro client % docker push athmajan/dht_coap_client:1.0
The push refers to repository [docker.io/athmajan/dht_coap_client]
227d183e03ed: Pushed
952a5ed4512e: Pushed
c0cd70dc037: Pushed
fa27da690096: Pushed
7afb1fb71b58: Pushed
25ba4ced53aa: Pushed
e8b03412ba5c: Pushed
9b56e6ac3d03: Pushed
af177303564b: Pushed
3f61b501be53: Pushed
7806df21f4c4: Pushed
1.0: digest: sha256:0f65c7cd71bce289e5994a82390dc5e690fb5fe1242de29fa83aac4b8ee5179c size: 2635
(base) athmajanvivekananthal@Asthmajans-MacBook-Pro client % docker push athmajan/dht_coap_server:1.0
The push refers to repository [docker.io/athmajan/dht_coap_server]
97cdffe1596e: Pushed
fd49df85c9c3: Pushed
f181e220eb23: Pushed
fa27da690096: Mounted from athmajan/dht_coap_client
7afb1fb71b58: Mounted from athmajan/dht_coap_client
25ba4ced53aa: Mounted from athmajan/dht_coap_client
e8b03412ba5c: Mounted from athmajan/dht_coap_client
9b56e6ac3d03: Mounted from athmajan/dht_coap_client
af177303564b: Mounted from athmajan/dht_coap_client
3f61b501be53: Mounted from athmajan/dht_coap_client
7806df21f4c4: Mounted from athmajan/dht_coap_client
1.0: digest: sha256:a63fd2db6009a8a65032d861818ac9fb6fd78a517df78d0a8c3da922114ef659 size: 2635
(base) athmajanvivekananthal@Asthmajans-MacBook-Pro client %

```

Figure 3.2. Screenshot of Terminal during Docker Push

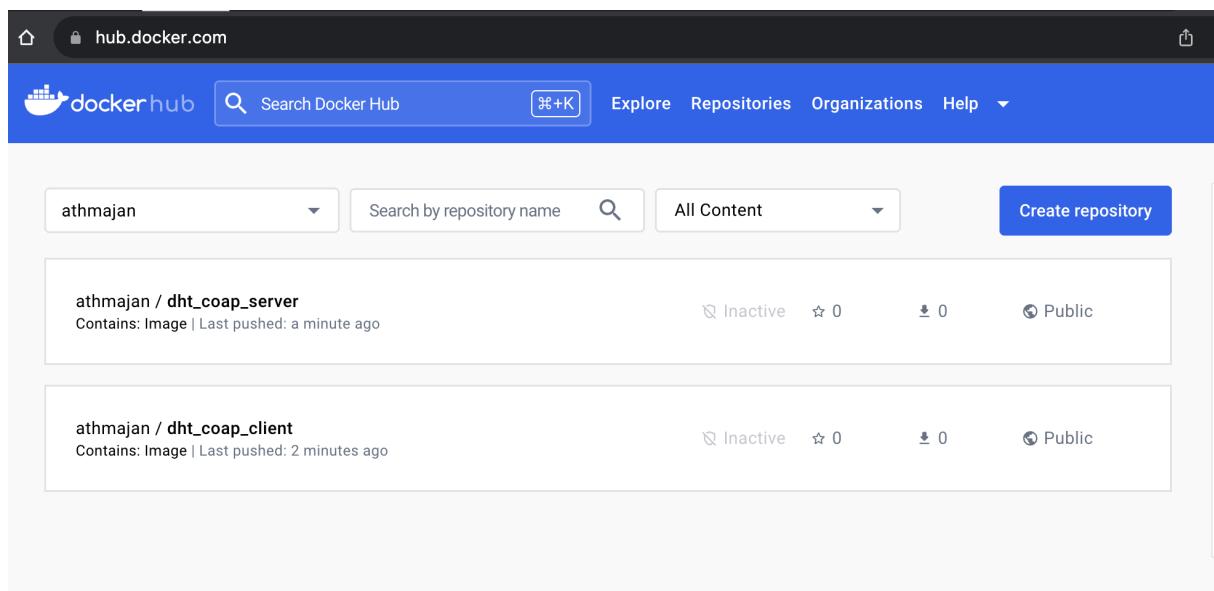


Figure 3.3. Screenshot of Docker Hub Repository

3.2 Questions

8. **How a container works on top on a host operating system? Mention few benefits of container-based virtualization over hypervisor-based virtualization.**

A container works on top of a host operating system by sharing the OS kernel while providing an isolated environment for its processes. Containers are lightweight and efficient since they use the host OS, which is Unlike virtual machines that require their own OS. They run in isolated user spaces provided by the host kernel, ensuring that each container has its own filesystem, CPU, memory, and process space. This isolation is the result of the use of kernel features like namespaces and cgroups. Containers are created from images that contain everything needed to run an application, including code, runtime, system tools, and libraries.

Some benefits of using containers are:

Efficiency and speed:Containers share the host OS kernel and maximize hardware usage, making them more lightweight and efficient than hypervisor-based VM, with faster startup time.

Portability:Since they encapsulate everything the application needs, they can be moved or shared easily across different environments without any compatibility issues.

Microservices Architecture: supports microservices by allowing individual components of an application to be deployed independently. **Greater density:** It is possible to run more containers on a given hardware than Vms, which results in better resource utilization

9. **Why an image starts with *FROM* instruction?**

In Docker and container technologies, an image begins with the FROM instruction, which specifies the base image, providing the essential environment and dependencies for the container. This base image is a lightweight starting point, containing the essential

filesystem, libraries, and settings required for an application or environment. By specifying a base image with the FROM instruction, we set the foundation for the container, which could be used to simplify the image creation process by allowing users to use it as a template, like a minimal operating system or any environment for a specific programming language

10. How a host machine can talk to a container?

A host machine can communicate with a container through exposed ports and shared volumes. When running a container, specific ports can be mapped to ports on the host machine. Additionally, the host and container can share data through bind mounts or volumes, where a directory on the host is mapped to a location within the container. This setup enables file sharing and persistent data storage for the container. These methods ensure interaction between host machine and containerized applications.

11. Summarize the Docker size, size of the 4 docker images (InfluxDB, Grafana, CoAP client and CoAP server) and all 4 containers. Finally, compare with the answer you got at section 1.3 Task (2) and Question 7.

Summary listing of Docker container sizes are available in Table 3.2. Screenshot of Docker ps output can be seen in Figure 3.4, 3.5.

Compared to results obtained in Task 2, The total size used in Docker containers are way less as compared in Table 3.1.

Task Name	Total size
Task 2 - VM and OVA	5.46GM
Task 3 - Dockerized	2.853

Table 3.1. Comparison of Task 2 and Task 3

Docker Image	Docker Container Size
CoAP Client	1.06GB
CoAP Server	1.1GB
InfluxDB	295MB
Grafana	389MB

Table 3.2. Docker Container and Image Sizes

```
build cache: 20      0      1.105MB  1.105MB
(base) athmajanivekanthan@Athmajans-MacBook-Pro:client % docker ps -a -s
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS          NAMES
7501411f897f        dht_coap_client:1.0   "python3 client.py"   5 hours ago       Up 5 hours         vigilant_tharp   87.2kB (virtual 1.06GB)
1ceccda0b92d        dht_coap_server:1.0    "python3 server.py"   5 hours ago       Up 5 hours         vigorous_almeida 87.5kB (virtual 1.1GB)
91bc7a785ac0        grafana/grafana:10.2.0  "/run.sh"           7 hours ago       Up 7 hours         pedantic_blackwell 977kB (virtual 389MB)
d157cf5a2b9b        influxdb:1.8.10      "/entrypoint.sh infl..." 7 hours ago       Up 7 hours         focused_gould   279B (virtual 295MB)
(base) athmajanivekanthan@Athmajans-MacBook-Pro:client %
```

Figure 3.4. Docker ps output

Images Give feedback						
Local		Hub	Artifactory	EARLY ACCESS		
		1.83 GB / 0 Bytes in use 6 images			Last refresh: 2 hours ago	
<input type="text"/> Search						
Name	Tag	Status	Created	Size	Actions	
athmajan/dht_coap_client 644e630821e1	1.0	In use	1 hour ago	1.06 GB		
dht_coap_client 644e630821e1	1.0	In use	1 hour ago	1.06 GB		
athmajan/dht_coap_server f26e61b8c168	1.0	In use	1 hour ago	1.09 GB		
dht_coap_server f26e61b8c168	1.0	In use	1 hour ago	1.09 GB		
influxdb 4e5441f2ea17	1.8.10	In use	14 days ago	294.67 MB		
grafana/grafana 9815cc4ede1a	10.2.0	In use	1 month ago	388 MB		

Figure 3.5. Docker Images

4 DISCUSSION

Throughout the project, we gained a comprehensive understanding of deploying applications and services using three distinct deployment methods. The experience of containerizing was novel for us and learned new concepts.

The task descriptions provided clear instructions, facilitating the completion of assignments without significant challenges. However, an obstacle arose during Task 2 when we discovered that VirtualBox was not compatible with the Apple M1 Pro chipset, the latest chip compared to earlier Intel versions. Since local deployment had already been executed on an Apple M1 Pro, we decided to proceed with the same host for Task 2, utilizing alternative hypervisors.

Opting for VMWare Fusion enabled us to successfully deploy a guest OS of Ubuntu. During this process, we identified a suggestion for improvement in the instructions. Instead of providing a direct link to download the guest OS in the instructional PDF, it would be beneficial to direct students to a general website where they can choose the correct OS suitable for their host operating system. This adjustment aims to prevent the download of an incompatible ISO image, as experienced in our case.

However, exporting to OVA is not yet supported for Apple M1 Pro chips, leading us to recreate everything on another computer from scratch solely for exporting the guest OS in OVA format. This experience served as an eye-opener, prompting us to recommend including such dependencies in the instructions for future projects.

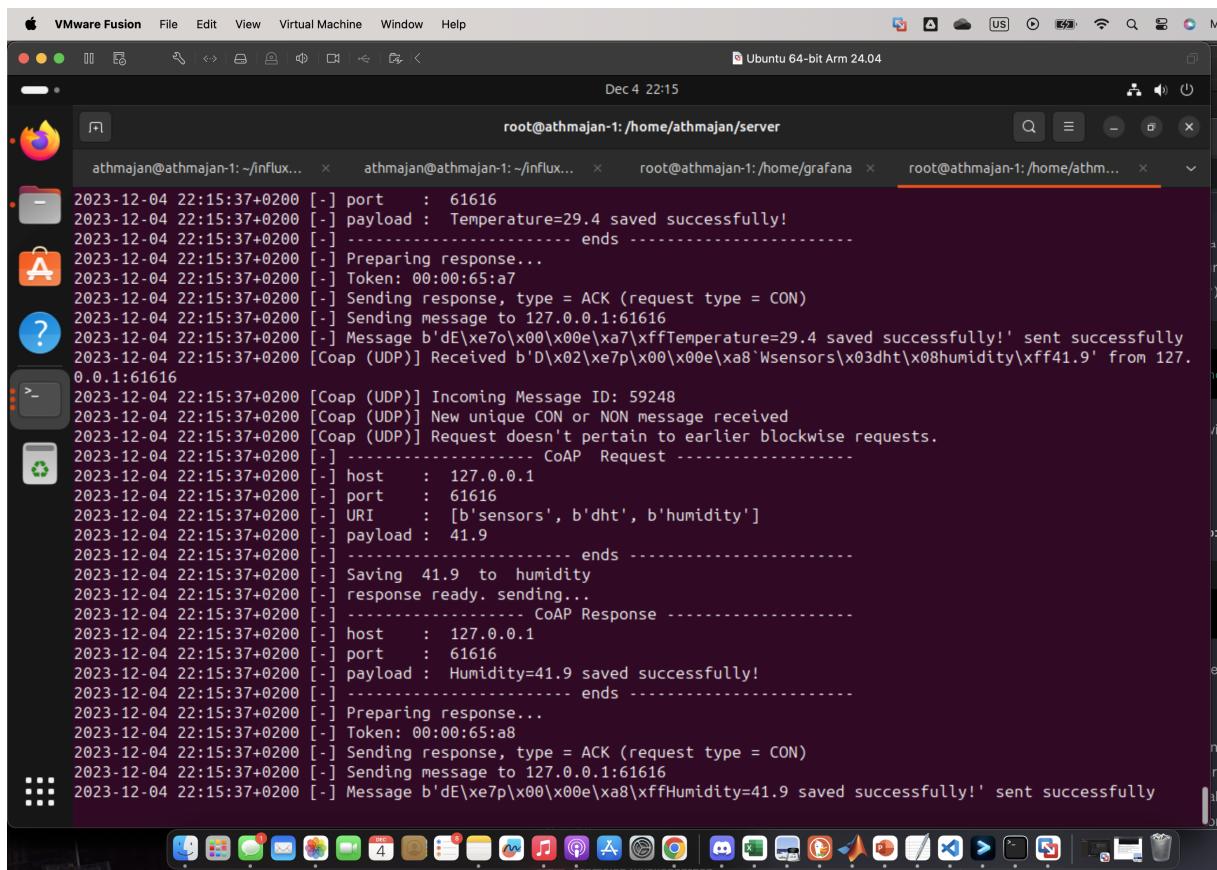
Despite the challenges encountered, the overall difficulty level of the project's implementation was not particularly high. We anticipate that future tasks might present slightly greater challenges, providing an opportunity for continuous learning and improvement.

5 APPENDICES

Appendix 1 Task 2 - Screenshots of terminals on Guest OS Applications

Appendix 2 Task 2 - Screenshots of memory Usages of Guest OS applications and their dependencies.

Appendix 3 Task 2 - Screenshots of VM Configurations and OVA format exports.



```
root@athmajan-1:/home/athmajan/server
Dec 4 22:15
athmajan@athmajan-1:~/influx...  athmajan@athmajan-1:~/influx...  root@athmajan-1:/home/grafana...  root@athmajan-1:/home/athm...
2023-12-04 22:15:37+0200 [-] port      : 61616
2023-12-04 22:15:37+0200 [-] payload   : Temperature=29.4 saved successfully!
2023-12-04 22:15:37+0200 [-] ----- ends -----
2023-12-04 22:15:37+0200 [-] Preparing response...
2023-12-04 22:15:37+0200 [-] Token: 00:00:65:a7
2023-12-04 22:15:37+0200 [-] Sending response, type = ACK (request type = CON)
2023-12-04 22:15:37+0200 [-] Sending message to 127.0.0.1:61616
2023-12-04 22:15:37+0200 [-] Message b'dE\xe70\x00\x00e\x a7\xffTemperature=29.4 saved successfully!' sent successfully
2023-12-04 22:15:37+0200 [Coap (UDP)] Received b'D\x02\xe7p\x00\x00e\x a8`wsensors\x03dht\x08humidity\xff41.9' from 127.0.0.1:61616
2023-12-04 22:15:37+0200 [Coap (UDP)] Incoming Message ID: 59248
2023-12-04 22:15:37+0200 [Coap (UDP)] New unique CON or NON message received
2023-12-04 22:15:37+0200 [Coap (UDP)] Request doesn't pertain to earlier blockwise requests.
2023-12-04 22:15:37+0200 [-] ----- CoAP Request -----
2023-12-04 22:15:37+0200 [-] host      : 127.0.0.1
2023-12-04 22:15:37+0200 [-] port      : 61616
2023-12-04 22:15:37+0200 [-] URI       : [b'sensors', b'dht', b'humidity']
2023-12-04 22:15:37+0200 [-] payload   : 41.9
2023-12-04 22:15:37+0200 [-] ----- ends -----
2023-12-04 22:15:37+0200 [-] Saving 41.9 to humidity
2023-12-04 22:15:37+0200 [-] response ready. sending...
2023-12-04 22:15:37+0200 [-] ----- CoAP Response -----
2023-12-04 22:15:37+0200 [-] host      : 127.0.0.1
2023-12-04 22:15:37+0200 [-] port      : 61616
2023-12-04 22:15:37+0200 [-] payload   : Humidity=41.9 saved successfully!
2023-12-04 22:15:37+0200 [-] ----- ends -----
2023-12-04 22:15:37+0200 [-] Preparing response...
2023-12-04 22:15:37+0200 [-] Token: 00:00:65:a8
2023-12-04 22:15:37+0200 [-] Sending response, type = ACK (request type = CON)
2023-12-04 22:15:37+0200 [-] Sending message to 127.0.0.1:61616
2023-12-04 22:15:37+0200 [-] Message b'dE\xe7p\x00\x00e\x a8\xffHumidity=41.9 saved successfully!' sent successfully
```

Figure 5.1. Terminal of CoAP Server on Guest OS

```
root@athmajan-1: /home/athmajan/client
Dec 4 22:17
[Coap (UDP)] New unique ACK or RST message received
[Coap (UDP)] Exchange removed, Message ID: 59287.
[Coap (UDP)] Received Response, token: b'000065cf', host: 127.0.0.1, port: 5683
[Coap (UDP)] response found. reading...
[CoAP Response]
[host : 127.0.0.1]
[port : 5683]
[payload : Temperature=28.8 saved successfully!]
[ends]
[sending ... humidity # 42.8]
[CoAP Request]
[host : 127.0.0.1]
[port : 5683]
[URI : (b'sensors', b'dht', b'humidity')]
[payload : 42.8]
[ends]
[Sending message to 127.0.0.1:5683]
[Exchange added, Message ID: 59288.
Message b'D\x02\xe7\x98\x00\x0e\xd0\xffHumidity=42.8' sent successfully]
[Token: b'000065d0', Host: 127.0.0.1, Port: 5683]
[Received b'dE\xe7\x98\x00\x0e\xd0\xffHumidity=42.8 saved successfully!' from 127.0.0.1:5683]
[Incoming Message ID: 59288]
[Coap (UDP)] New unique ACK or RST message received
[Coap (UDP)] Exchange removed, Message ID: 59288.
[Coap (UDP)] Received Response, token: b'000065d0', host: 127.0.0.1, port: 5683
[Coap (UDP)] response found. reading...
[CoAP Response]
[host : 127.0.0.1]
[port : 5683]
[payload : Humidity=42.8 saved successfully!]
[ends]
```

Figure 5.2. Terminal of CoAP Client on Guest OS

```
athmajan@athmajan-1: ~/influxdb-1.8.10-1/usr/bin
Dec 4 22:16
athmajan@athmajan-1: ~/influx... x athmajan@athmajan-1: ~/influx... x root@athmajan-1: /home/grafana x root@athmajan-1: /home/athm...
/2.31.0" 073bea13-92e2-11ee-81c8-000c293fc52a 4124
2023-12-04T20:16:37.682507Z info Executing query {"log_id": "0lu_6rtG000", "service": "query", "query": "CREATE DATABASE dht"}
[httpd] 127.0.0.1 - influxDBUser [04/Dec/2023:22:16:37 +0200] "POST /query?q=CREATE+DATABASE+%22dht%22 HTTP/1.1" 200 5
8 "-" "python-requests/2.31.0" 073ddb04-92e2-11ee-81c9-000c293fc52a 1075
[httpd] 127.0.0.1 - influxDBUser [04/Dec/2023:22:16:37 +0200] "POST /write?db=dht HTTP/1.1" 204 0 "-" "python-requests/2.31.0" 073bea13-92e2-11ee-81c8-000c293fc52a 561
2023-12-04T20:16:42.696388Z info Executing query {"log_id": "0lu_6rtG000", "service": "query", "query": "CREATE DATABASE dht"}
[httpd] 127.0.0.1 - influxDBUser [04/Dec/2023:22:16:42 +0200] "POST /query?q=CREATE+DATABASE+%22dht%22 HTTP/1.1" 200 5
8 "-" "python-requests/2.31.0" 0a3adc7a-92e2-11ee-81cb-000c293fc52a 941
[httpd] 127.0.0.1 - influxDBUser [04/Dec/2023:22:16:42 +0200] "POST /write?db=dht HTTP/1.1" 204 0 "-" "python-requests/2.31.0" 0a3b96b7-92e2-11ee-81cc-000c293fc52a 2362
2023-12-04T20:16:42.709827Z info Executing query {"log_id": "0lu_6rtG000", "service": "query", "query": "CREATE DATABASE dht"}
[httpd] 127.0.0.1 - influxDBUser [04/Dec/2023:22:16:42 +0200] "POST /query?q=CREATE+DATABASE+%22dht%22 HTTP/1.1" 200 5
8 "-" "python-requests/2.31.0" 0a3cf858-92e2-11ee-81cd-000c293fc52a 312
[httpd] 127.0.0.1 - influxDBUser [04/Dec/2023:22:16:42 +0200] "POST /write?db=dht HTTP/1.1" 204 0 "-" "python-requests/2.31.0" 0a3f4d6c-92e2-11ee-81ce-000c293fc52a 4395
2023-12-04T20:16:47.741487Z info Executing query {"log_id": "0lu_6rtG000", "service": "query", "query": "CREATE DATABASE dht"}
[httpd] 127.0.0.1 - influxDBUser [04/Dec/2023:22:16:47 +0200] "POST /query?q=CREATE+DATABASE+%22dht%22 HTTP/1.1" 200 5
8 "-" "python-requests/2.31.0" 0d3caffa-92e2-11ee-81cf-000c293fc52a 864
[httpd] 127.0.0.1 - influxDBUser [04/Dec/2023:22:16:47 +0200] "POST /write?db=dht HTTP/1.1" 204 0 "-" "python-requests/2.31.0" 0d3d035e-92e2-11ee-81d0-000c293fc52a 3168
2023-12-04T20:16:47.751669Z info Executing query {"log_id": "0lu_6rtG000", "service": "query", "query": "CREATE DATABASE dht"}
[httpd] 127.0.0.1 - influxDBUser [04/Dec/2023:22:16:47 +0200] "POST /query?q=CREATE+DATABASE+%22dht%22 HTTP/1.1" 200 5
8 "-" "python-requests/2.31.0" 0d3e4ad0-92e2-11ee-81d1-000c293fc52a 327
[httpd] 127.0.0.1 - influxDBUser [04/Dec/2023:22:16:47 +0200] "POST /write?db=dht HTTP/1.1" 204 0 "-" "python-requests/2.31.0" 0d3e787e-92e2-11ee-81d2-000c293fc52a 1075
```

Figure 5.3. Terminal of InfluxDB Server on Guest OS

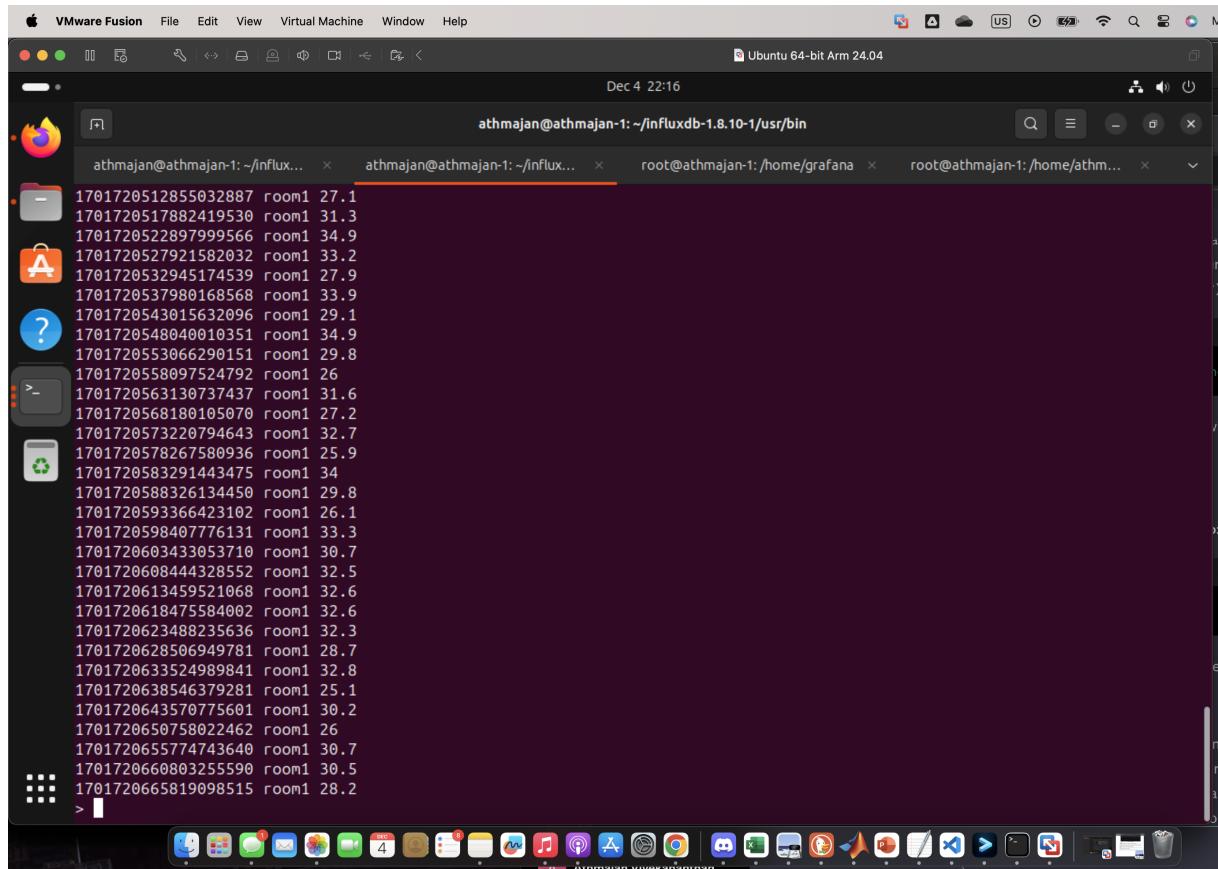
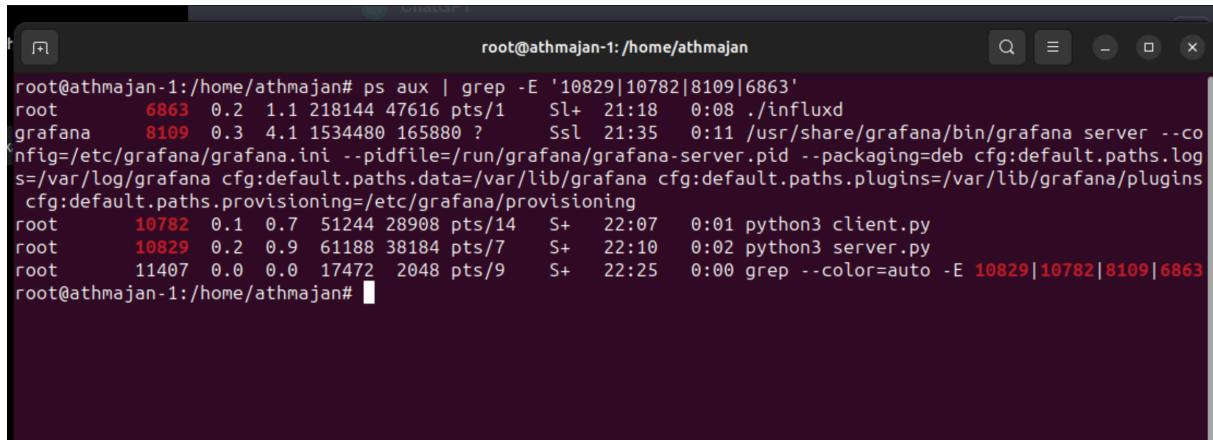
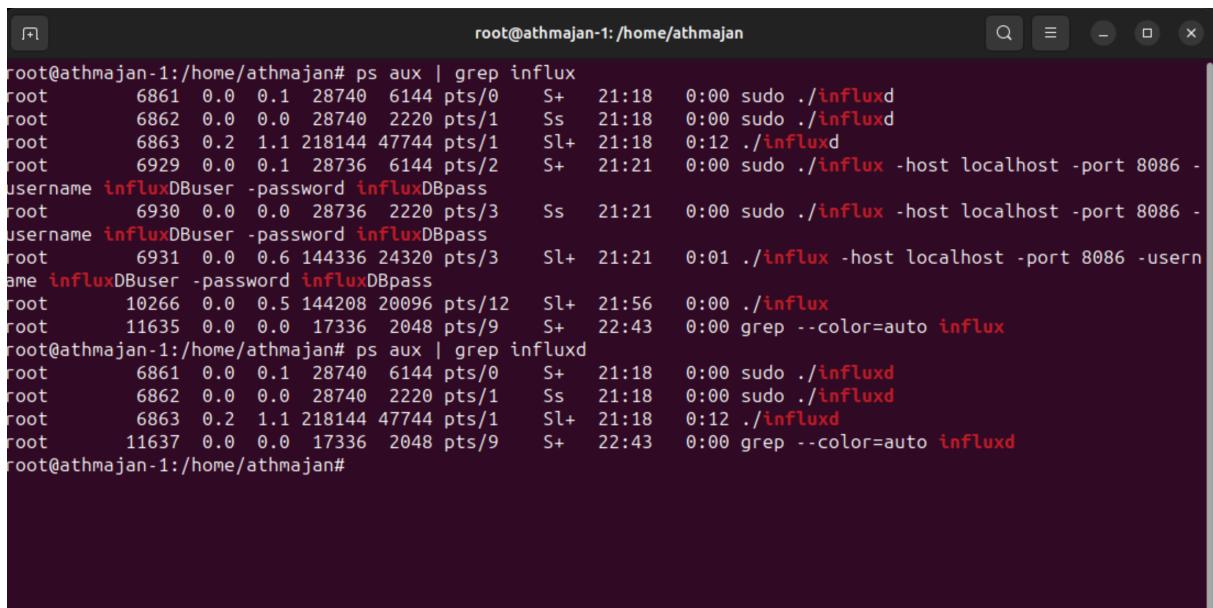


Figure 5.4. Terminal of Influx CLI on Guest OS



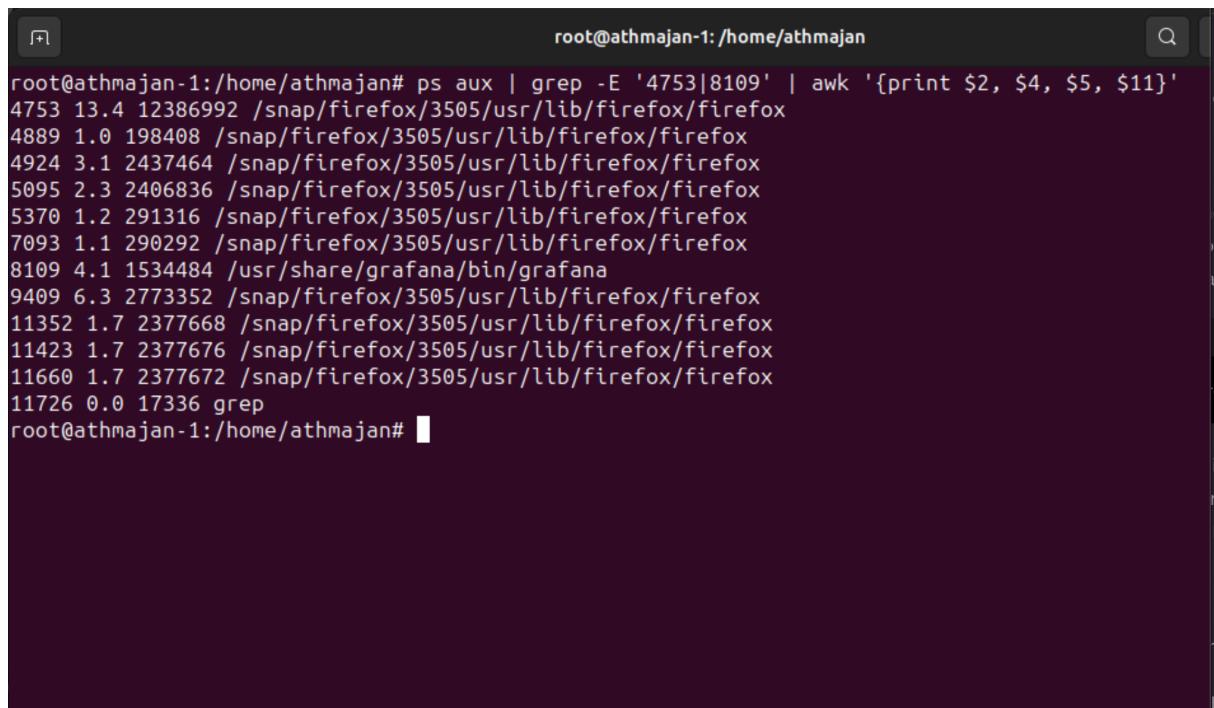
```
root@athmajan-1:/home/athmajan# ps aux | grep -E '10829|10782|8109|6863'
root      6863  0.2  1.1 218144 47616 pts/1    Sl+ 21:18   0:08 ./influxd
grafana   8109  0.3  4.1 1534480 165880 ?      Ssl 21:35   0:11 /usr/share/grafana/bin/grafana server --co
nfig=/etc/grafana/grafana.ini --pidfile=/run/grafana/grafana-server.pid --packaging=deb cfg:default.paths.log
s=/var/log/grafana cfg:default.paths.data=/var/lib/grafana cfg:default.paths.plugins=/var/lib/grafana/plugins
cfg:default.paths.provisioning=/etc/grafana/provisioning
root      10782  0.1  0.7 51244 28908 pts/14   S+   22:07   0:01 python3 client.py
root      10829  0.2  0.9 61188 38184 pts/7    S+   22:10   0:02 python3 server.py
root      11407  0.0  0.0 17472 2048 pts/9    S+   22:25   0:00 grep --color=auto -E 10829|10782|8109|6863
root@athmajan-1:/home/athmajan#
```

Figure 5.5. Terminal output showing memory usage in Guest OS



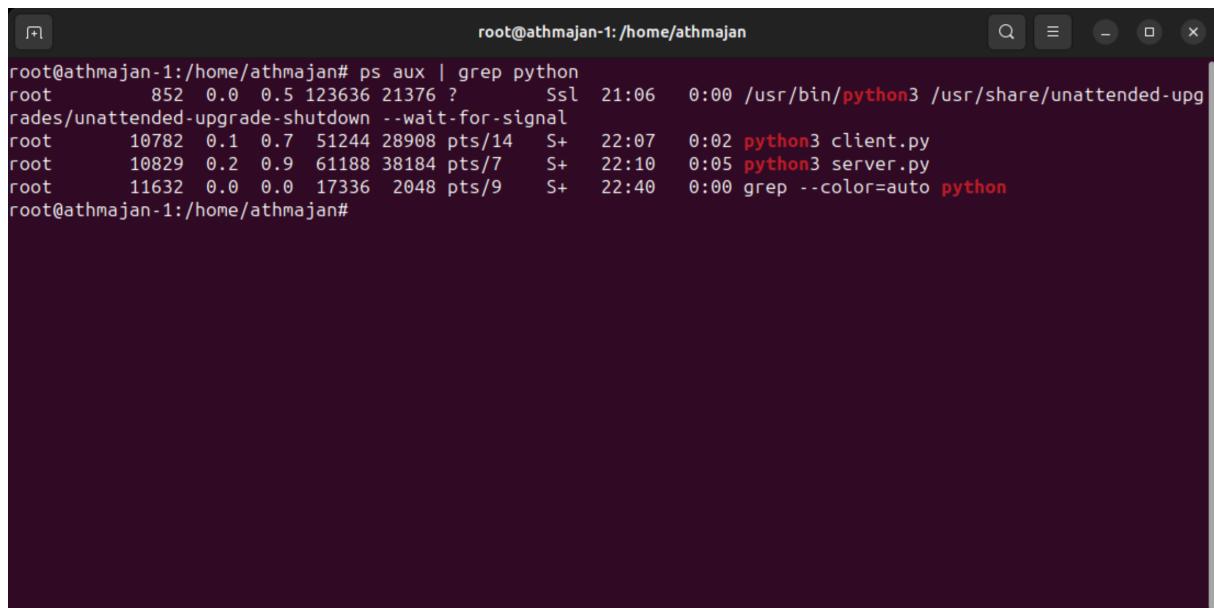
```
root@athmajan-1:/home/athmajan# ps aux | grep influx
root      6861  0.0  0.1 28740  6144 pts/0    S+   21:18   0:00 sudo ./influxd
root      6862  0.0  0.0 28740  2220 pts/1    Ss   21:18   0:00 sudo ./influxd
root      6863  0.2  1.1 218144 47744 pts/1   Sl+ 21:18   0:12 ./influxd
root      6929  0.0  0.1 28736  6144 pts/2    S+   21:21   0:00 sudo ./influx -host localhost -port 8086 -
username influxDBuser -password influxDBpass
root      6930  0.0  0.0 28736  2220 pts/3    Ss   21:21   0:00 sudo ./influx -host localhost -port 8086 -
username influxDBuser -password influxDBpass
root      6931  0.0  0.6 144336 24320 pts/3   Sl+ 21:21   0:01 ./influx -host localhost -port 8086 -usern
ame influxDBuser -password influxDBpass
root      10266 0.0  0.5 144208 20096 pts/12   Sl+ 21:56   0:00 ./influx
root      11635 0.0  0.0 17336  2048 pts/9    S+   22:43   0:00 grep --color=auto influx
root@athmajan-1:/home/athmajan# ps aux | grep influxd
root      6861  0.0  0.1 28740  6144 pts/0    S+   21:18   0:00 sudo ./influxd
root      6862  0.0  0.0 28740  2220 pts/1    Ss   21:18   0:00 sudo ./influxd
root      6863  0.2  1.1 218144 47744 pts/1   Sl+ 21:18   0:12 ./influxd
root      11637 0.0  0.0 17336  2048 pts/9    S+   22:43   0:00 grep --color=auto influxd
root@athmajan-1:/home/athmajan#
```

Figure 5.6. Terminal showing memory usage of InfluxDB and dependencies in Guest OS



```
root@athmajan-1:/home/athmajan# ps aux | grep -E '4753|8109' | awk '{print $2, $4, $5, $11}'
4753 13.4 12386992 /snap/firefox/3505/usr/lib/firefox/firefox
4889 1.0 198408 /snap/firefox/3505/usr/lib/firefox/firefox
4924 3.1 2437464 /snap/firefox/3505/usr/lib/firefox/firefox
5095 2.3 2406836 /snap/firefox/3505/usr/lib/firefox/firefox
5370 1.2 291316 /snap/firefox/3505/usr/lib/firefox/firefox
7093 1.1 290292 /snap/firefox/3505/usr/lib/firefox/firefox
8109 4.1 1534484 /usr/share/grafana/bin/grafana
9409 6.3 2773352 /snap/firefox/3505/usr/lib/firefox/firefox
11352 1.7 2377668 /snap/firefox/3505/usr/lib/firefox/firefox
11423 1.7 2377676 /snap/firefox/3505/usr/lib/firefox/firefox
11660 1.7 2377672 /snap/firefox/3505/usr/lib/firefox/firefox
11726 0.0 17336 grep
root@athmajan-1:/home/athmajan#
```

Figure 5.7. Terminal showing memory usage of Grafana and dependencies in Guest OS



```
root@athmajan-1:/home/athmajan# ps aux | grep python
root      852  0.0  0.5 123636 21376 ?        Ssl  21:06   0:00 /usr/bin/python3 /usr/share/unattended-upgrades/unattended-upgrade-shutdown --wait-for-signal
root     10782  0.1  0.7 51244 28908 pts/14   S+  22:07   0:02 python3 client.py
root     10829  0.2  0.9 61188 38184 pts/7    S+  22:10   0:05 python3 server.py
root     11632  0.0  0.0 17336  2048 pts/9    S+  22:40   0:00 grep --color=auto python
root@athmajan-1:/home/athmajan#
```

Figure 5.8. Terminal showing memory usage of CoAP Server, Client and dependencies in Guest OS

```

Apple Tabby Edit View Window Help
1 influx Server 2 influx CLI 3 Grafana 4 Server 5 Client 6 OS default 7 OS default + 
Every 1.0s: ps aux | grep "VMware Fusion"
Athmajan-MacBook-Pro.local: Mon Dec 4 22:34:48 2023
root 83840 4.5 5.0 486248192 833680 ?? S 8:59PM 32:45.74 /Applications/VMware Fusion.app/Contents/Library/vmware-vmx -E en -s vmx_studio.Keep=TRUE -# product=64;n
ames=VMware Fusion;version=13.5.0;buildNumber=22583700;licenseName=VMware Fusion for Mac OS;licenseVersion=13.0; @ duplex=3;msgs=ui -D 4 /Users/athmajanvivekananthan/Virtual Machines.l
ocalized/Ubuntu 64-bit Arm 24.04.vmvarevm/Ubuntu 64-bit Arm 24.04.vmx
athmajanvivekananthan 83639 0.5 0.4 409436168 60568 ?? S 8:58PM 8:39.06 /Applications/VMware Fusion.app/Contents/MacOS/VMware Fusion
athmajanvivekananthan 83811 0.2 0.0 408679264 2064 ?? Ss 8:58PM 0:00.48 /Library/Application Support/VMware/VMware Fusion/Services/Contents/Library/vmware-usbarbitrator
athmajanvivekananthan 83810 0.0 0.0 35304612 7836 ?? S 8:58PM 0:03.12 /Applications/VMware Fusion.app/Resources/Library/amsr -D p 8698
athmajanvivekananthan 83809 0.0 0.0 35004776 1360 ?? S 8:58PM 0:00.54 /Applications/VMware Fusion.app/Contents/Library/amsr -D p 8698
athmajanvivekananthan 83803 0.0 0.0 35579124 3956 ?? S 8:58PM 0:03.12 /Applications/VMware Fusion.app/Contents/Library/VMware Fusion Applications Menu.app/Contents/Frameworks/VMware Fusion Applications Menu Helper.app/Contents/MacOS/VMware Fusion Applications Menu.app/Contents/Resources/app.asar --node-integration=true --no-sandbox --service-pipe-token=06479B1973E93B8550EFF90D6C1202F
--lang=en-US --app-path=/Applications/VMware Fusion.app/Contents/Library/VMware Fusion Applications Menu.app/Contents/Resources/app.asar --node-integration=true --no-sandbox --service-pipe-token=06479B1973E93B8550EFF90D6C1202F
--o-sandbox --context-id=2 --enable-pinch --num-raster-threads=4 --enable-zero-copy --enable-gpu-memory-buffer-compositor-resources --enable-main-frame-before-activation --content-image-texture-target=0,0,3553;0,1,3553;0,2,3553;0,3,3553;0,4,3553;0,5,3553;0,6,3553;0,7,3553;0,8,3553;0,9,3553;0,10,3553;0,11,34037;0,12,34037;0,13,34037;0,14,3553;0,15,3553;0,16,3553;0,17,3
553;1,0,3553;1,1,3553;1,2,3553;1,3,3553;1,4,3553;1,5,3553;1,6,3553;1,7,3553;1,8,3553;1,9,3553;1,10,3553;1,11,34037;1,12,34037;1,13,34037;1,14,3553;1,15,3553;1,16,3553;1,17,3553;2,0,355
3;2,1,3553;2,2,3553;2,3,3553;2,4,3553;2,5,3553;2,6,3553;2,7,3553;2,8,3553;2,9,3553;2,10,3553;2,11,34037;2,12,34037;2,13,34037;2,14,3553;2,15,3553;2,16,3553;2,17,3553;3,0,3553;3,1,3553;3,2,3553;3,3,3553;3,4,3553;3,5,34037;3,6,3553;3,7,3553;3,8,3553;3,9,3553;3,10,3553;3,11,3553;3,12,3553;3,13,34037;3,14,34037;3,15,3553;3,16,34037;3,17,34037;4,0,3553;4,1,3553;4,2,3553;4,3,3553;4,4,3553;4,5,34037;4,6,3553;4,7,3553;4,8,3553;4,9,3553;4,10,3553;4,11,3553;4,12,3553;4,13,34037;4,14,34037;4,15,3553;4,16,34037;4,17,34037
--disable-accelerated-video-decode --enable-gpu-async-worker-context --service-request-channel-token=06479B1973E93B8550EFF90D6C1202F --renderer-client-id=3
root 83802 0.0 0.0 408725152 2064 ?? Ss 8:58PM 0:00.06 /Library/Application Support/VMware/VMware Fusion/Services/Contents/Library/vmnet-dhcpd -s 6 -cf /Library/Preferences/VMware Fusion/vmnet8/dhcpd.conf -lf /var/db/vmware/vmnet-dhcpd-vmnet8.leases -pf /var/run/vmnet-dhcpd-vmnet8.pid vmnet8
root 83800 0.0 0.0 408758656 4976 ?? S 8:58PM 0:10.50 /Library/Application Support/VMware/VMware Fusion/Services/Contents/Library/vmnet-natd -s 6 -m /Library/Preferences/VMware Fusion/vmnet8/nat_mac -c /Library/Preferences/VMware Fusion/vmnet8/nat.conf
root 83791 0.0 0.0 408725152 2000 ?? Ss 8:58PM 0:00.04 /Library/Application Support/VMware/VMware Fusion/Services/Contents/Library/vmnet-dhcpd -s 6 -cf /Library/Preferences/VMware Fusion/vmnet1/dhcpd.conf -lf /var/db/vmware/vmnet-dhcpd-vmnet1.leases -pf /var/run/vmnet-dhcpd-vmnet1.pid vmnet1
athmajanvivekananthan 83769 0.0 0.1 35683332 17440 ?? S 8:58PM 0:06.59 /Applications/VMware Fusion.app/Contents/Library/VMware Fusion Applications Menu.app/Contents/MacOS/VMware Fusion Applications Menu
athmajanvivekananthan 83765 0.0 0.0 408711616 2432 ?? S 8:58PM 0:00.02 /Library/Application Support/VMware/VMware Fusion/Services/Contents/Library/vmnet-bridge
athmajanvivekananthan 83763 0.0 0.0 408758560 7552 ?? S 8:58PM 0:00.25 /Applications/VMware Fusion.app/Contents/Library/VMware Fusion Start Menu.app/Contents/MacOS/VMware Fusion Start Menu
Fusion Start Menu
athmajanvivekananthan 83696 0.0 0.0 408636720 928 ?? S 8:58PM 0:00.01 /Library/Application Support/VMware/VMware Fusion/Services/Contents/Library/services/VMware Fusion Services 3.4
athmajanvivekananthan 89207 0.0 0.0 408102576 1200 s019 S+ 10:34PM 0:00.00 grep "VMware Fusion"
athmajanvivekananthan 89205 0.0 0.0 408103184 1568 s019 S+ 10:34PM 0:00.01 sh -c ps aux | grep "VMware Fusion"
athmajanvivekananthan 89204 0.0 0.0 408269888 944 s019 S+ 10:34PM 0:00.00 watch -n 1 ps aux | grep "VMware Fusion"

```

Figure 5.9. Terminal showing memory usage in Host OS while virtualization on hypervisor with Guest OS.

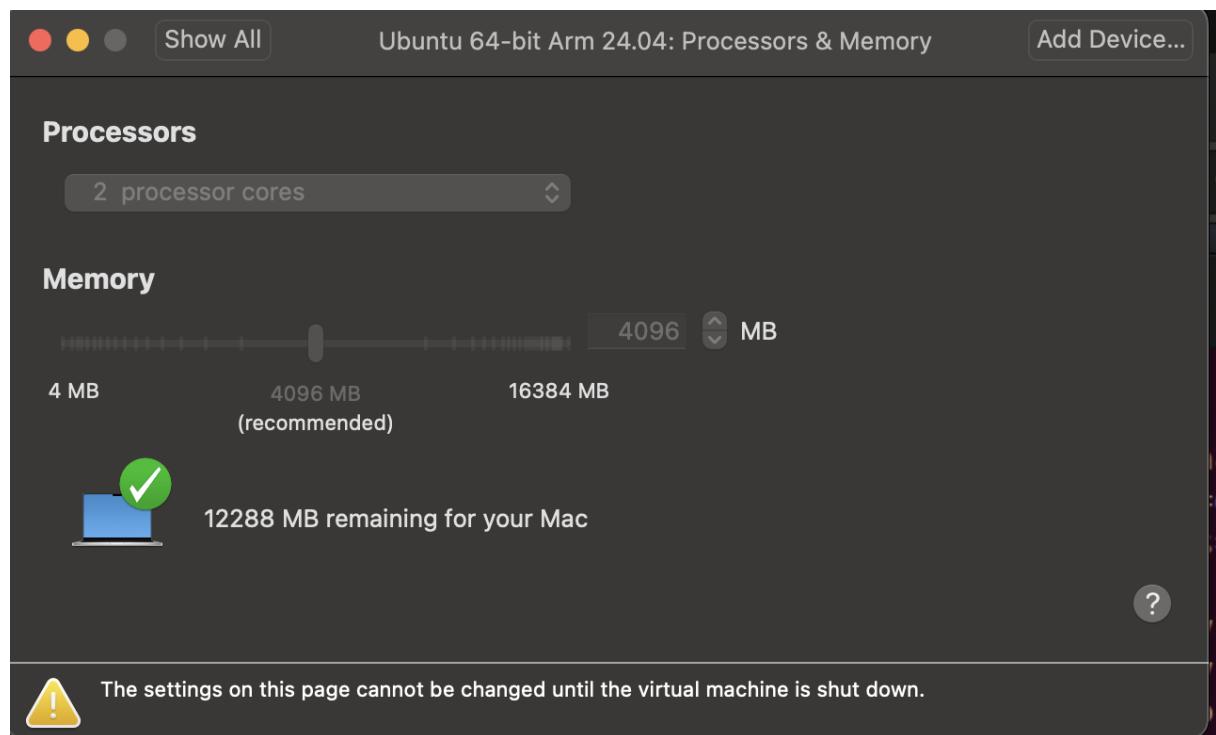


Figure 5.10. Screenshot of VM configuration on CPU and Memory

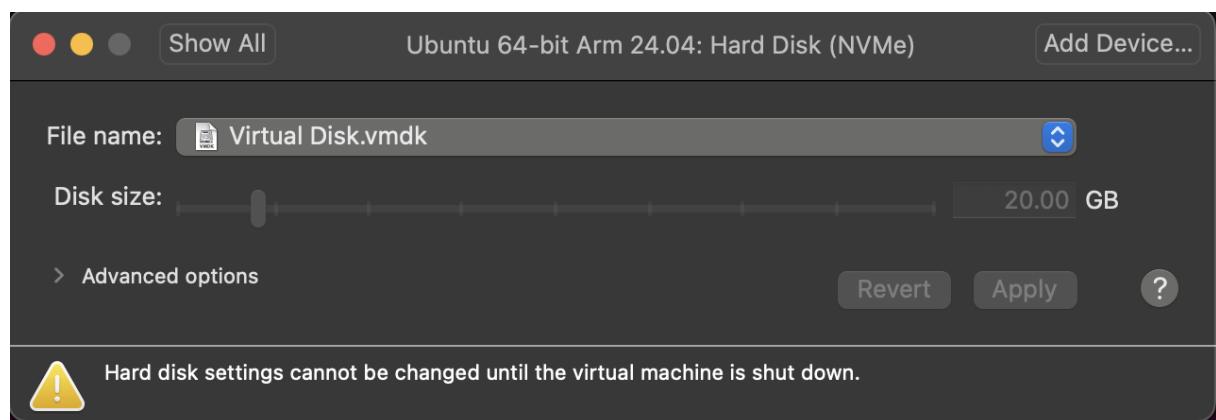


Figure 5.11. Screenshot of VM configuration on Disk Space

6 REFERENCES

- [1] Weerasinghe C.N.I.N. (2023) Lecture Notes.
- [2] Richard Blum C.B. (2020) Linux Command Line and Shell Scripting Bible, 4th Edition. John Wiley & Sons, Inc.