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

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.**

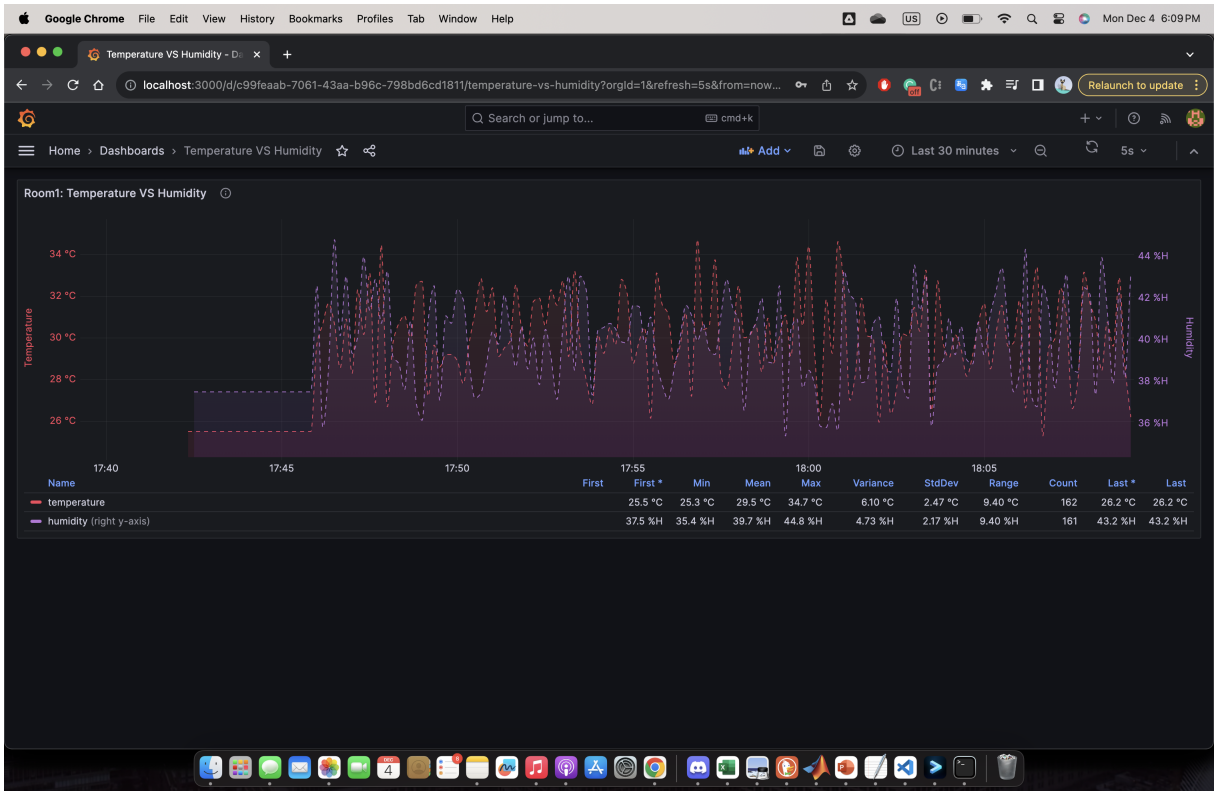


Figure 1.1. Grafana Dashboard for past 30 minutes

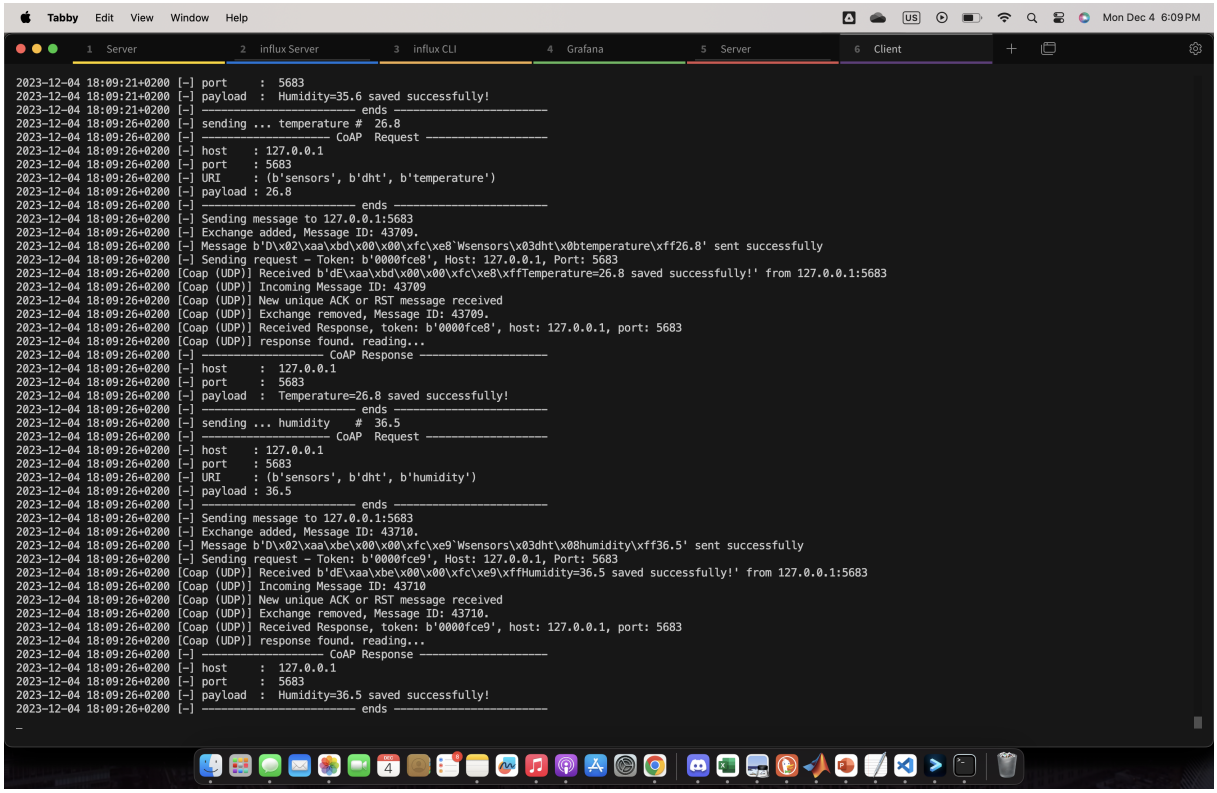


Figure 1.2. Terminal view of Python based local client

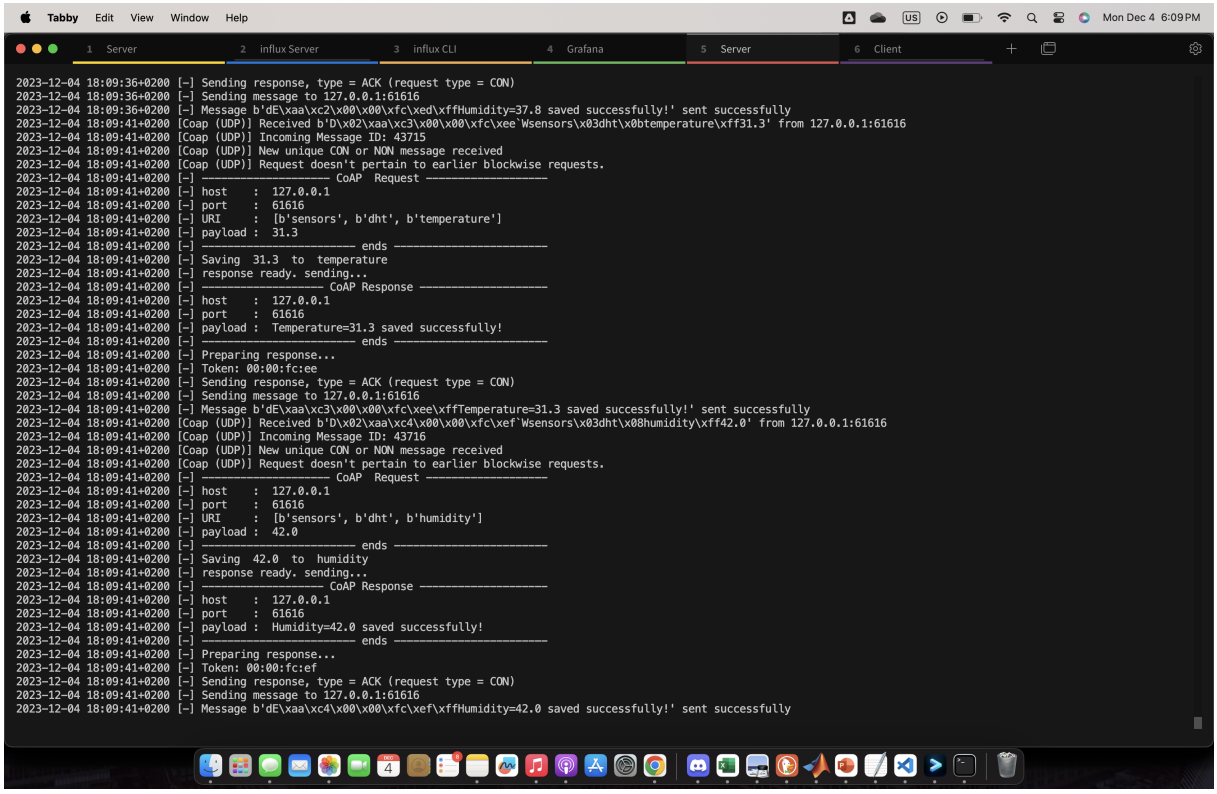


Figure 1.3. Terminal view of Python based local server

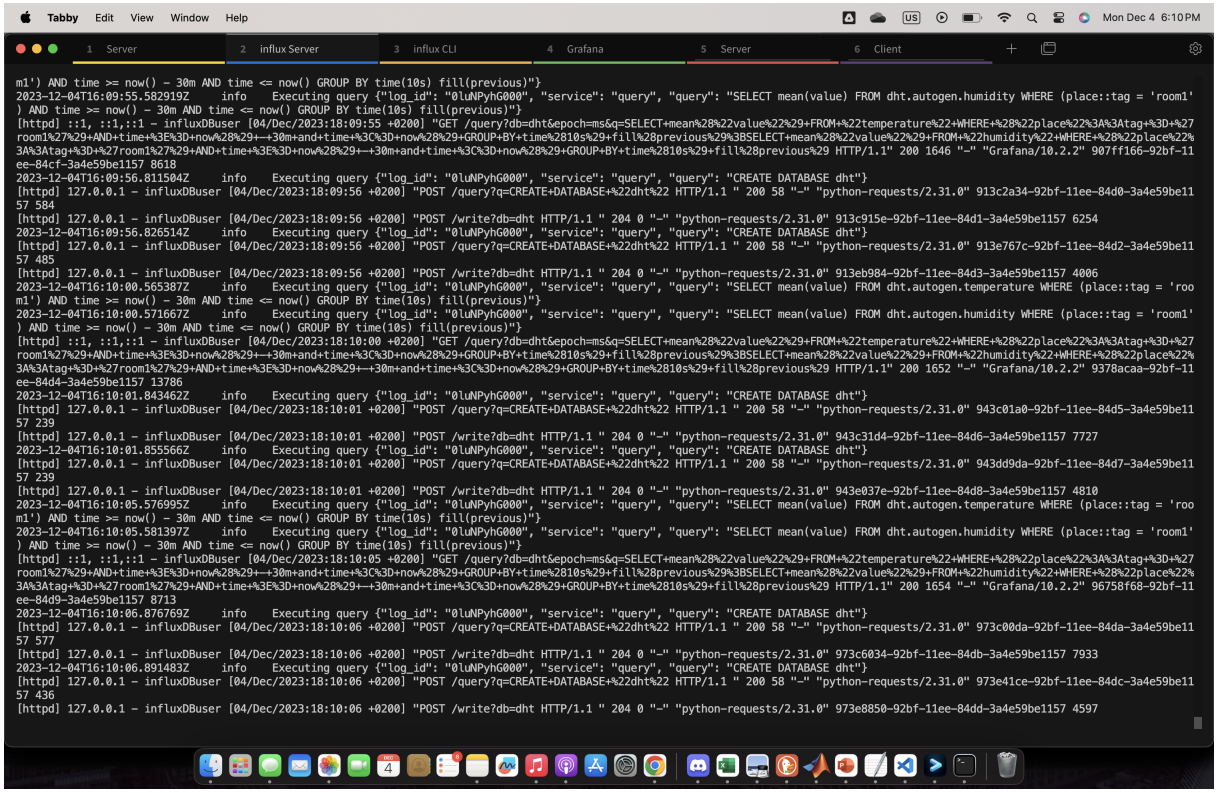


Figure 1.4. Terminal view of InfluxDB server

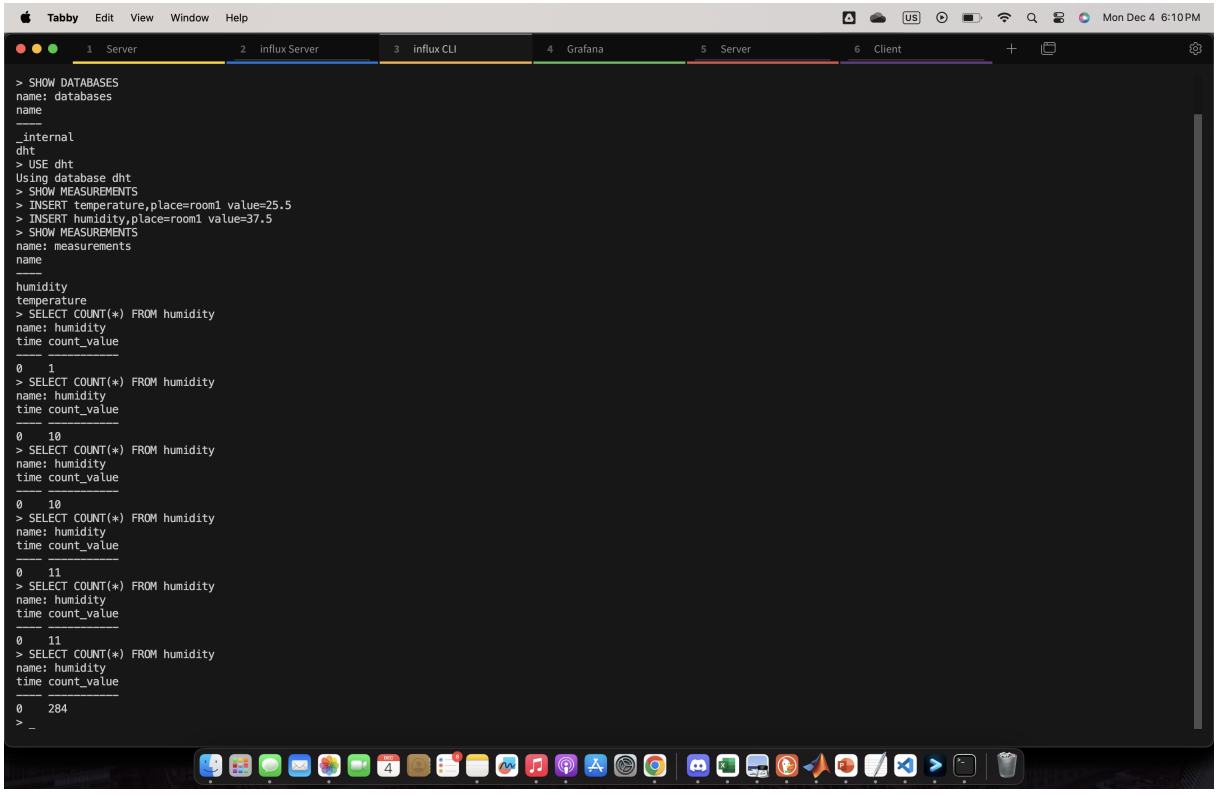


Figure 1.5. Terminal view of InfluxDB CLI

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

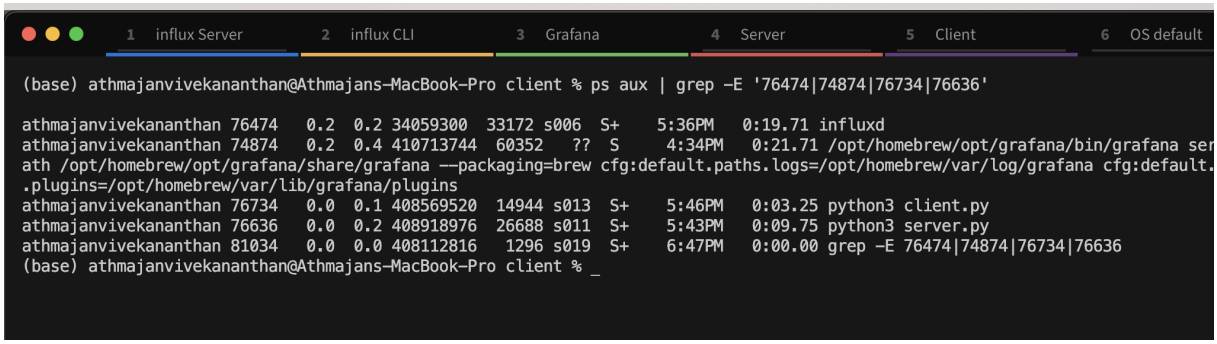


Figure 1.6. Memory Usage of Processes

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.

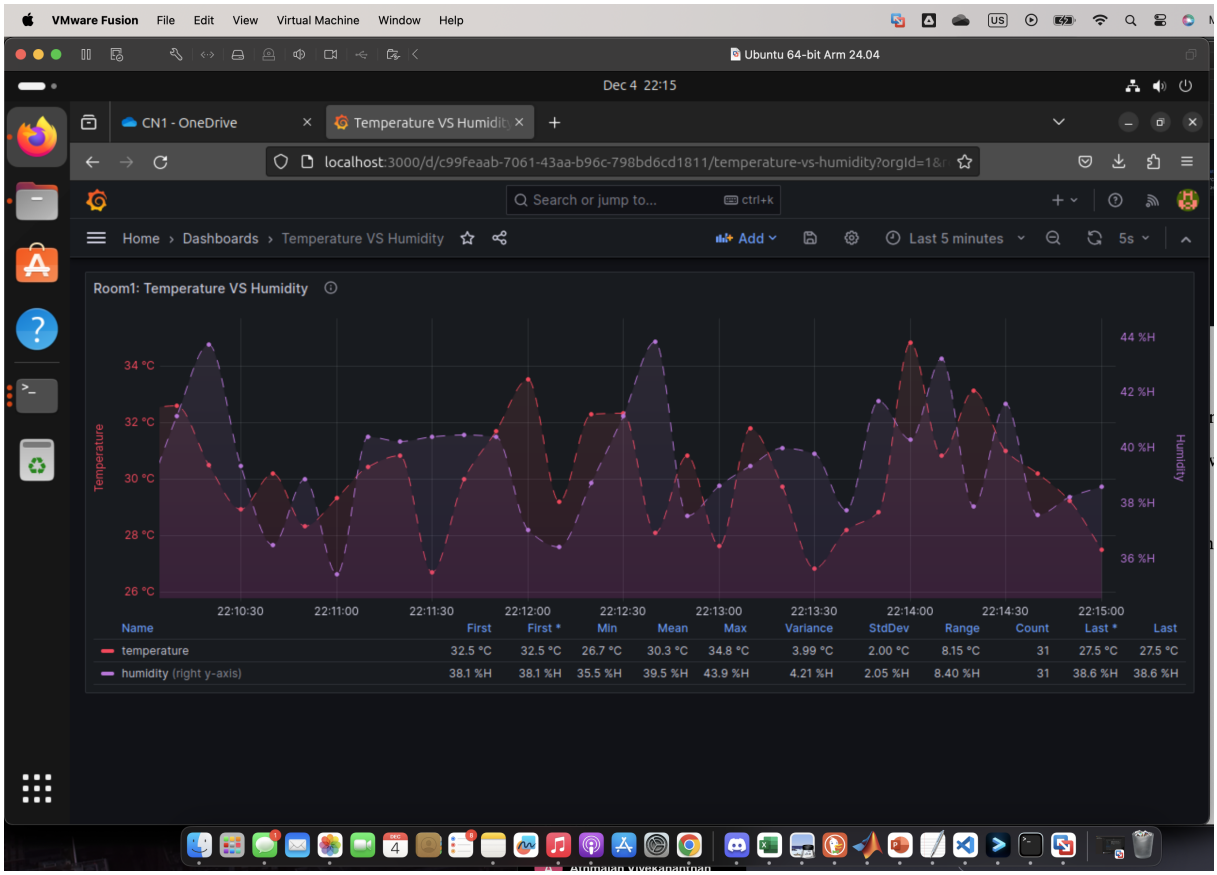


Figure 2.1. Grafana Dashboard as views on the Guest OS

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

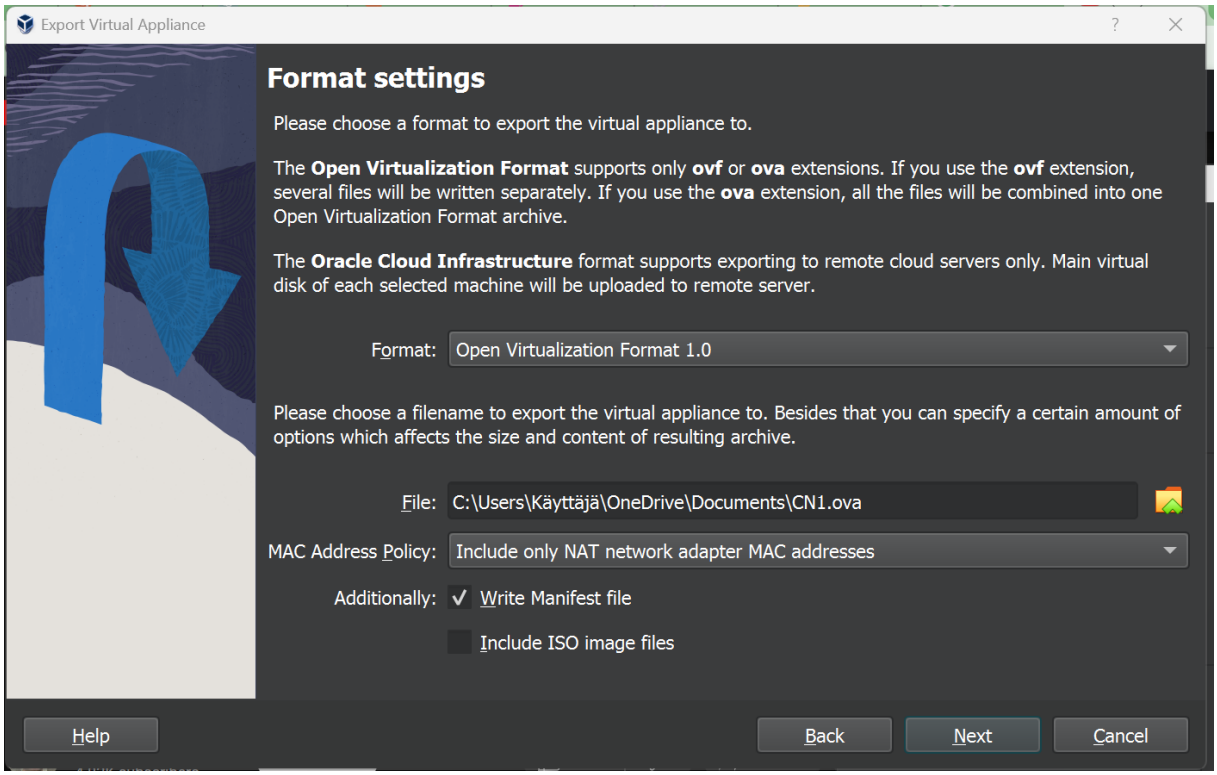


Figure 2.2. Export to OVA settings.

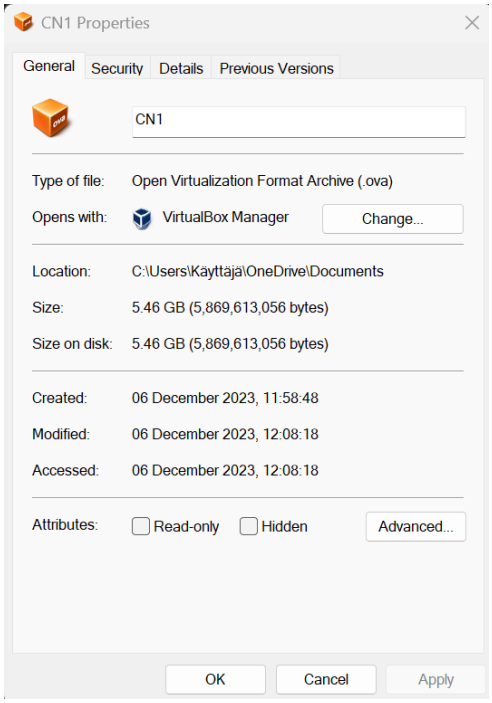


Figure 2.3. OVA File Size

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

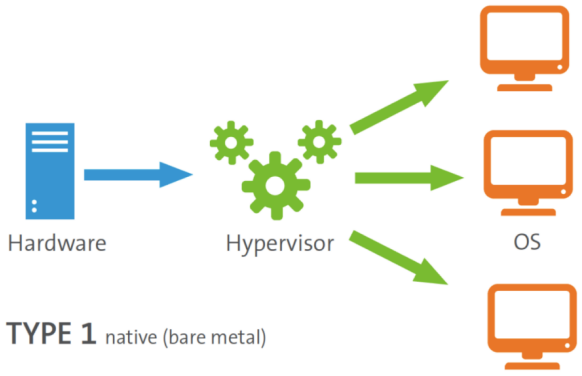


Figure 2.4. Hypervisor Type 1

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 multilpe 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 launced 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 mutiple 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. Som well-known examples for type 1 hypervisors are Oracle VM, Microsoft Hyper-V, VMWare ESX and Xen.

   Type 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 in other hand type 2 hypervisor is easy 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**

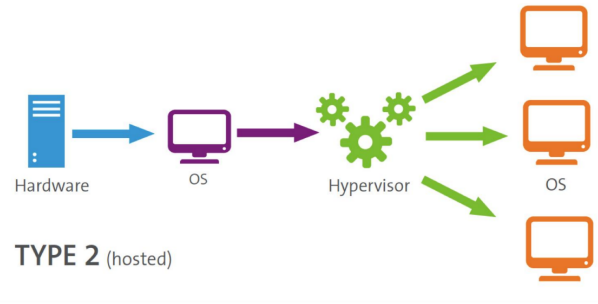


Figure 2.5. Hypervisor Type 2

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="$(dirname $PYTHON3_LOCATION):$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
```

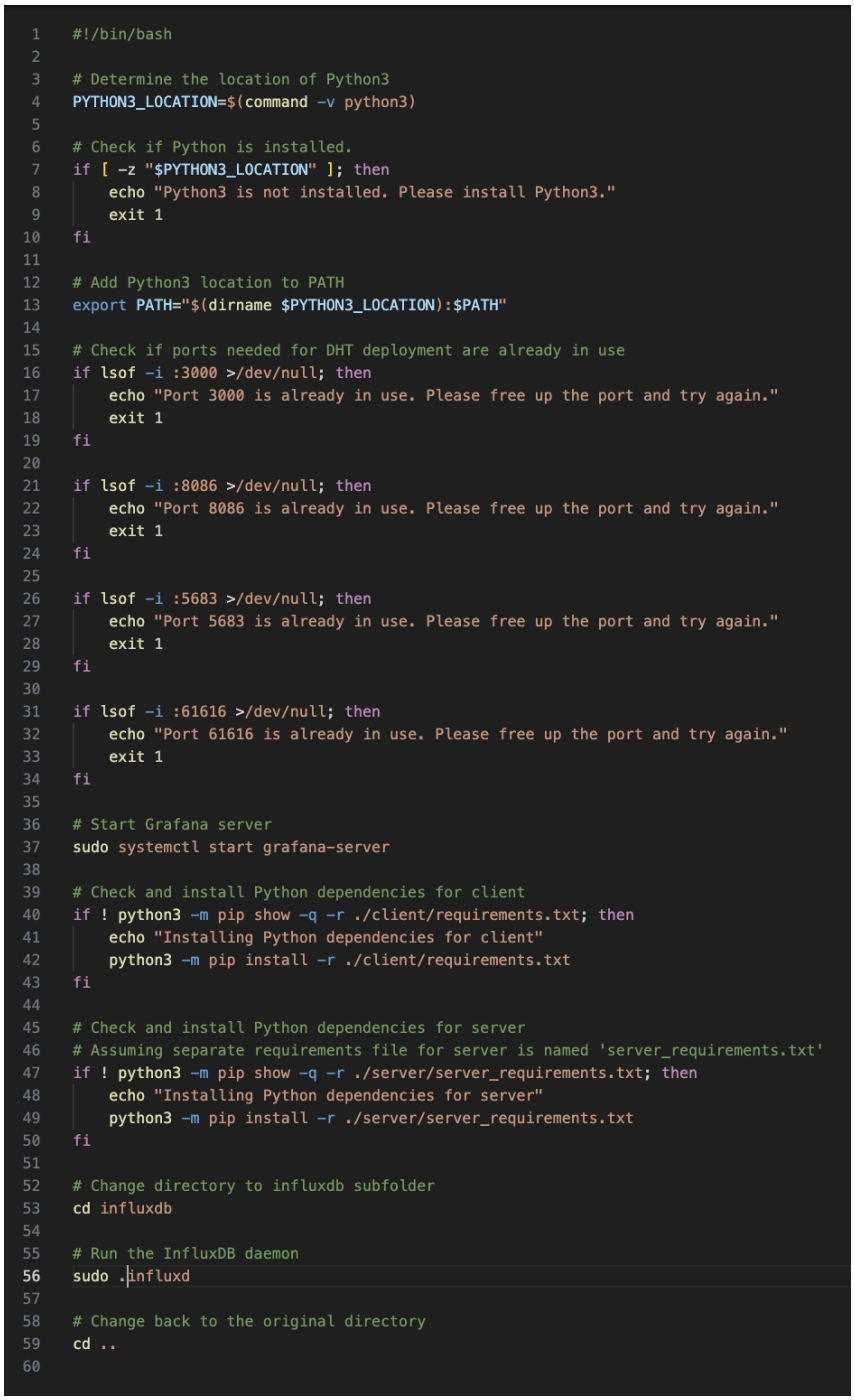
```
# Run the InfluxDB daemon
sudo .influxd

# 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.



```
#!/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="$(dirname $PYTHON3_LOCATION):$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 ..

```

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.

```
version: '3'
services:
  dht_coap_client:
    image: dht_coap_client:1.0
    network_mode: "host"
    ports:
      - "61616:61616"

  dht_coap_server:
    image: dht_coap_server:1.0
    network_mode: "host"
    ports:
      - "5683:5683"

  grafana:
    image: grafana/grafana:10.2.0
    ports:
      - "3000:3000"

  influxdb:
    image: influxdb:1.8.10
    ports:
      - "8086:8086"
    environment:
      - INFLUXDB_DB=dht
      - INFLUXDB_ADMIN_USER=influxDBuser
      - INFLUXDB_ADMIN_PASSWORD=influxDBpass
      - 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

```yaml
version: '3'
services:
  dht_coap_client:
    image: dht_coap_client:1.0
    network_mode: "host"
    ports:
      - "61616:61616"

  dht_coap_server:
    image: dht_coap_server:1.0
    network_mode: "host"
    ports:
      - "5683:5683"

  grafana:
    image: grafana/grafana:10.2.0
    ports:
      - "3000:3000"

  influxdb:
    image: influxdb:1.8.10
    ports:
      - "8086:8086"
    environment:
      - INFLUXDB_DB=dht
      - INFLUXDB_ADMIN_USER=influxDBuser
      - INFLUXDB_ADMIN_PASSWORD=influxDBpass
      - INFLUXDB_HTTP_AUTH_ENABLED=true
```

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.

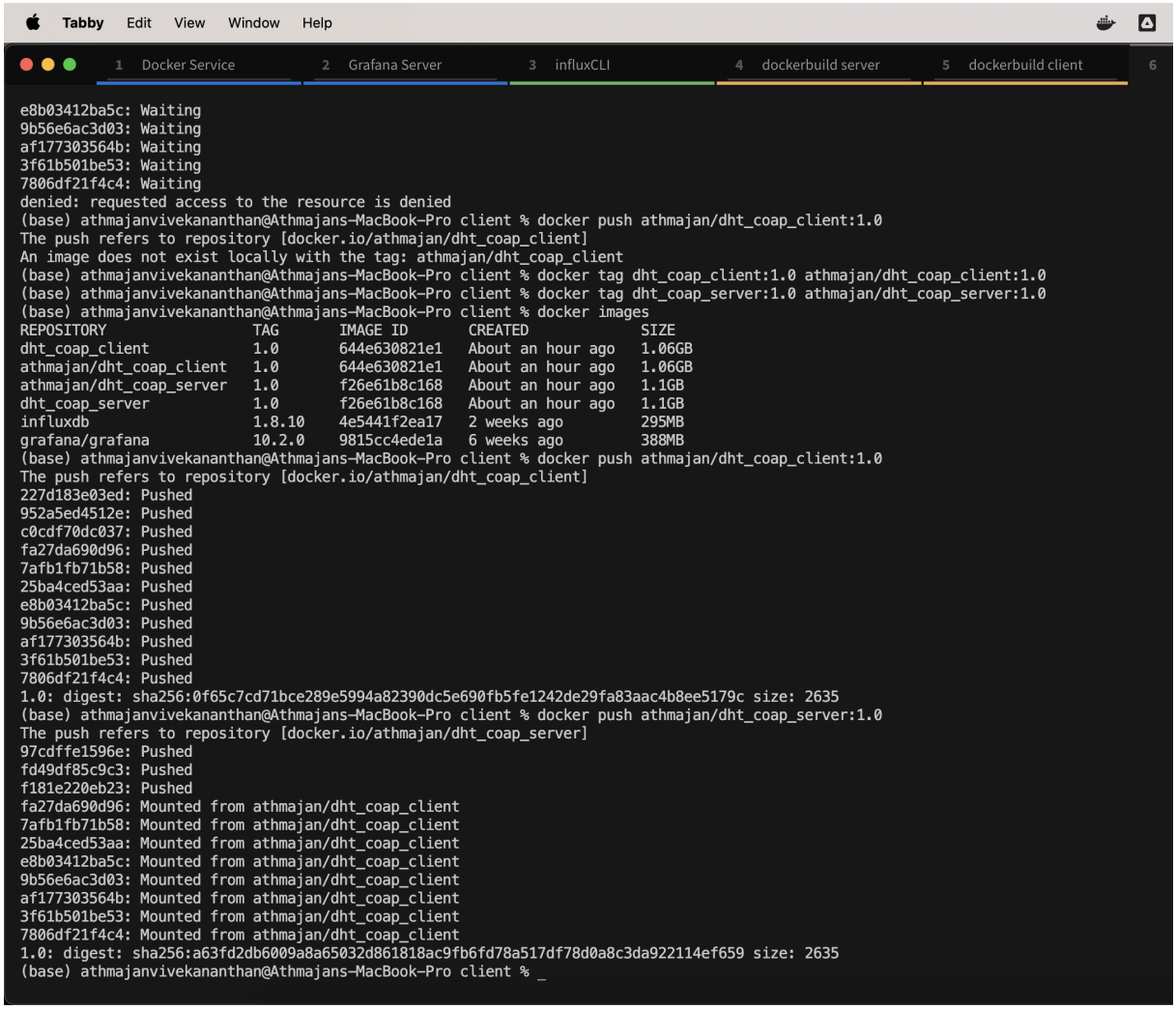


Figure 3.2. Screenshot of Terminal during Docker Push

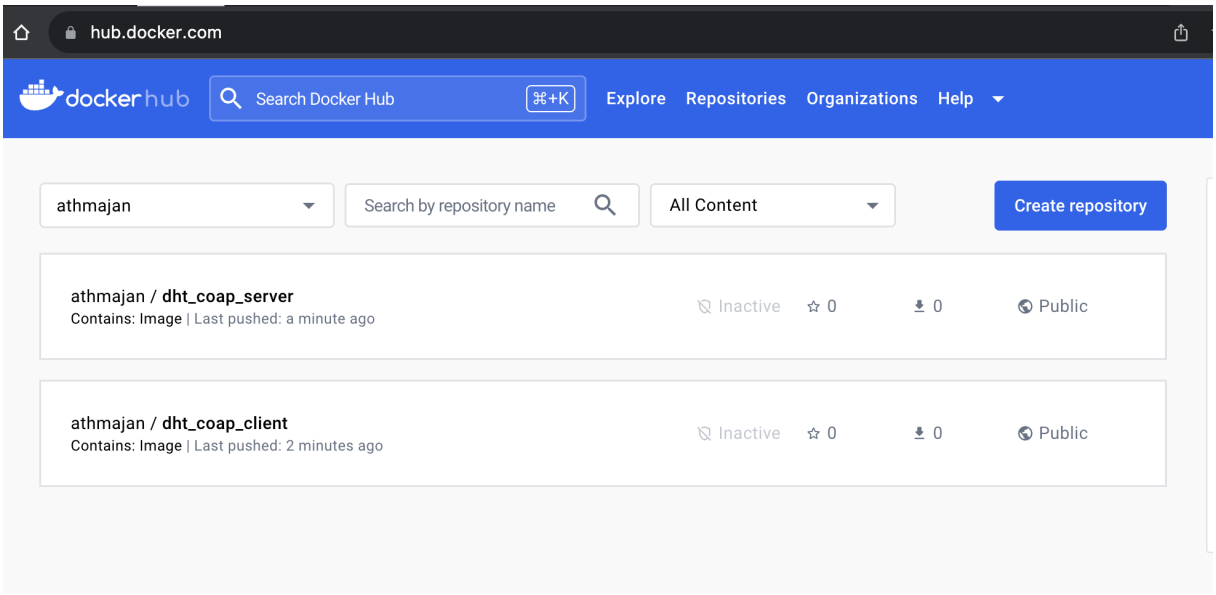


Figure 3.3. Screenshot of Docker Hub Repository

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

```
(base) athmajanvivekananthan@Athmajans-MacBook-Pro client % docker ps -a -s
CONTAINER ID   IMAGE                     COMMAND                  CREATED       STATUS       PORTS                    NAMES               SIZE
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   0.0.0.0:3000->3000/tcp   pedantic_blackwell  977kB (virtual 389MB)
d157cf5a2b9b   influxdb:1.8.10           "/entrypoint.sh infl…"   7 hours ago   Up 7 hours   0.0.0.0:8086->8086/tcp   focused_gould       279B (virtual 295MB)
(base) athmajanvivekananthan@Athmajans-MacBook-Pro client %
```

Figure 3.4. Docker ps output

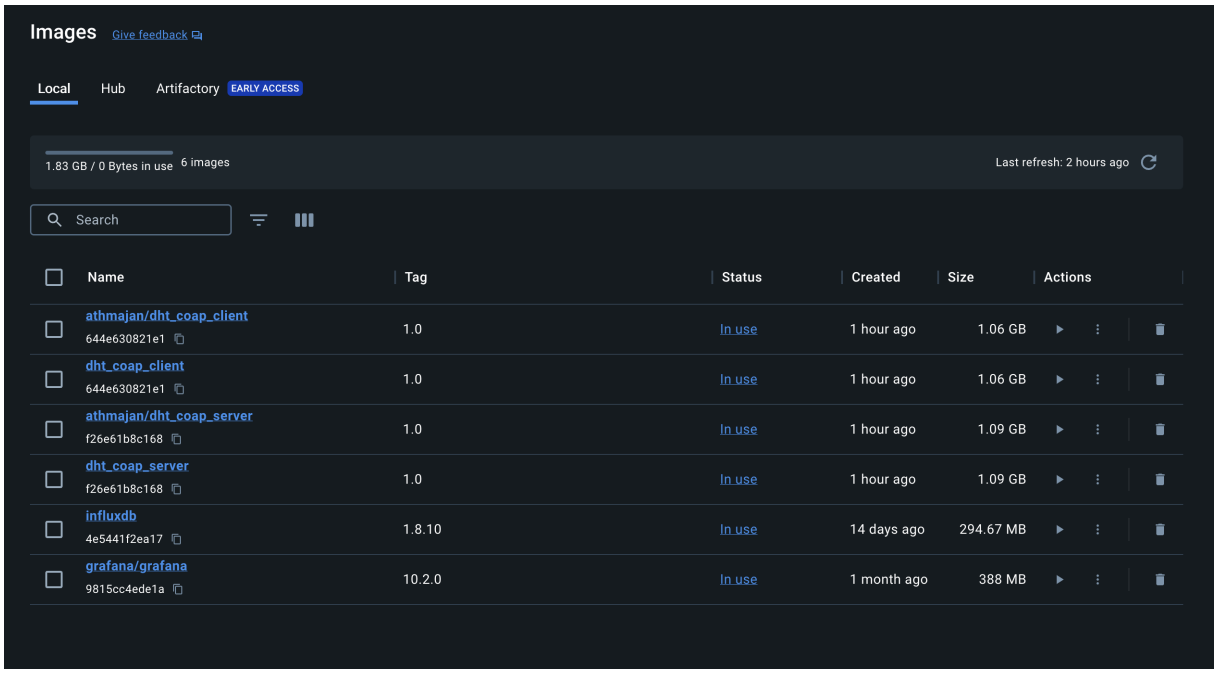


Figure 3.5. Docker Images

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.

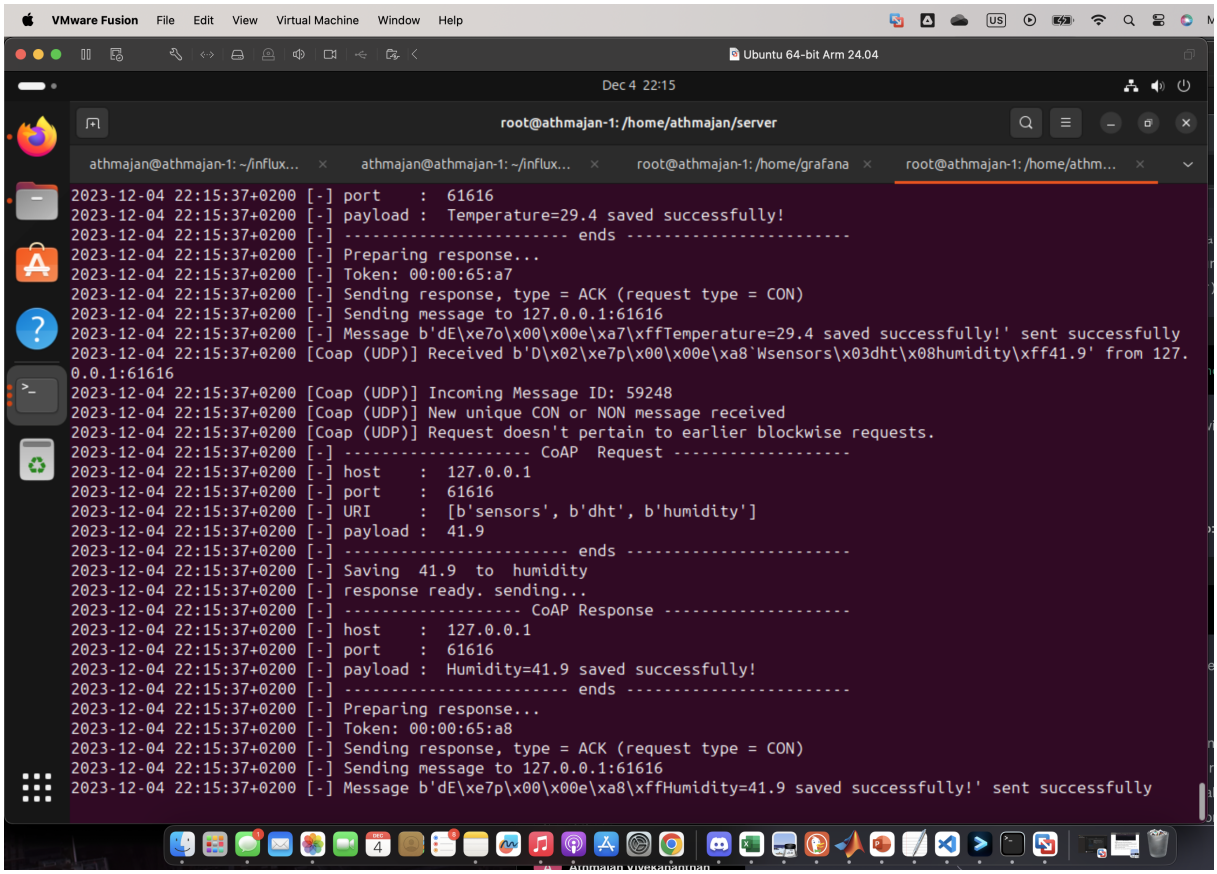


Figure 5.1. Terminal of CoAP Server on Guest OS

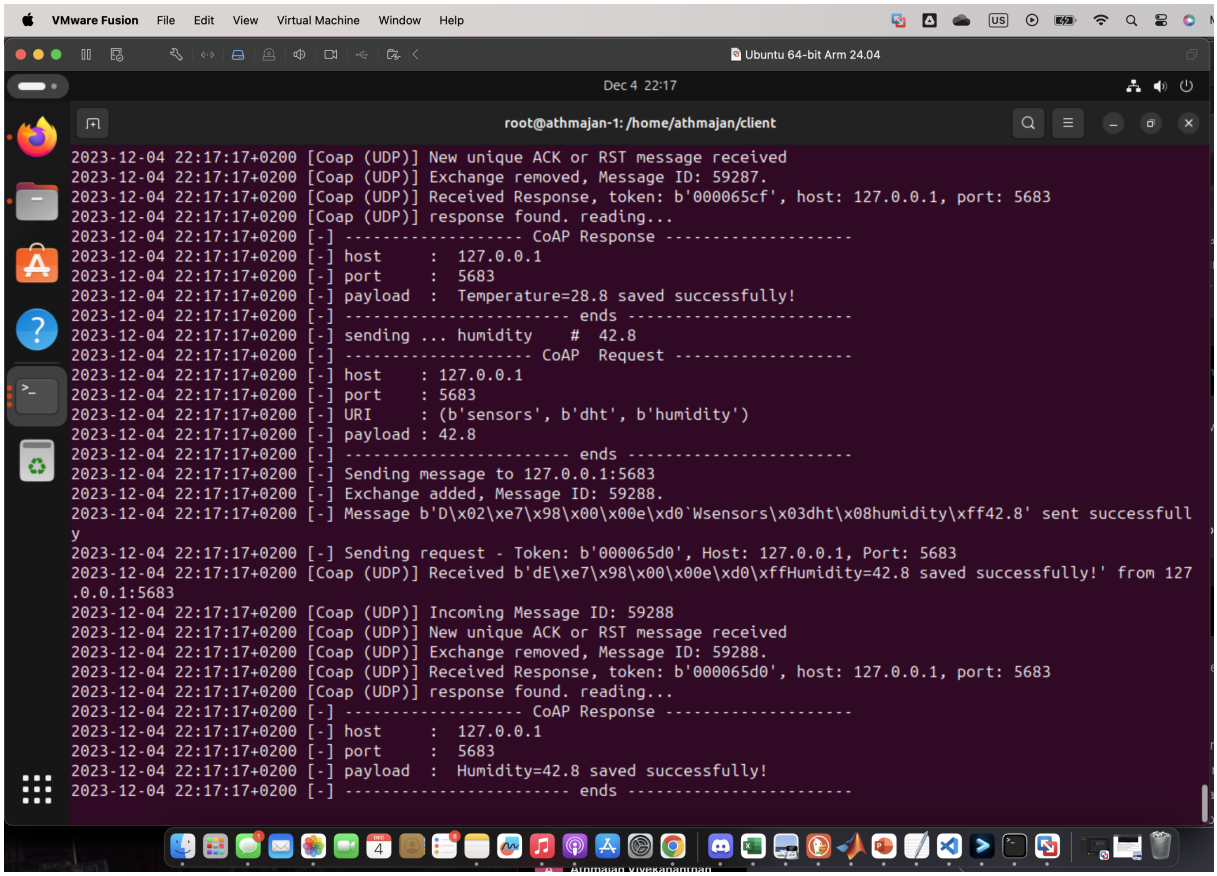


Figure 5.2. Terminal of CoAP Client on Guest OS

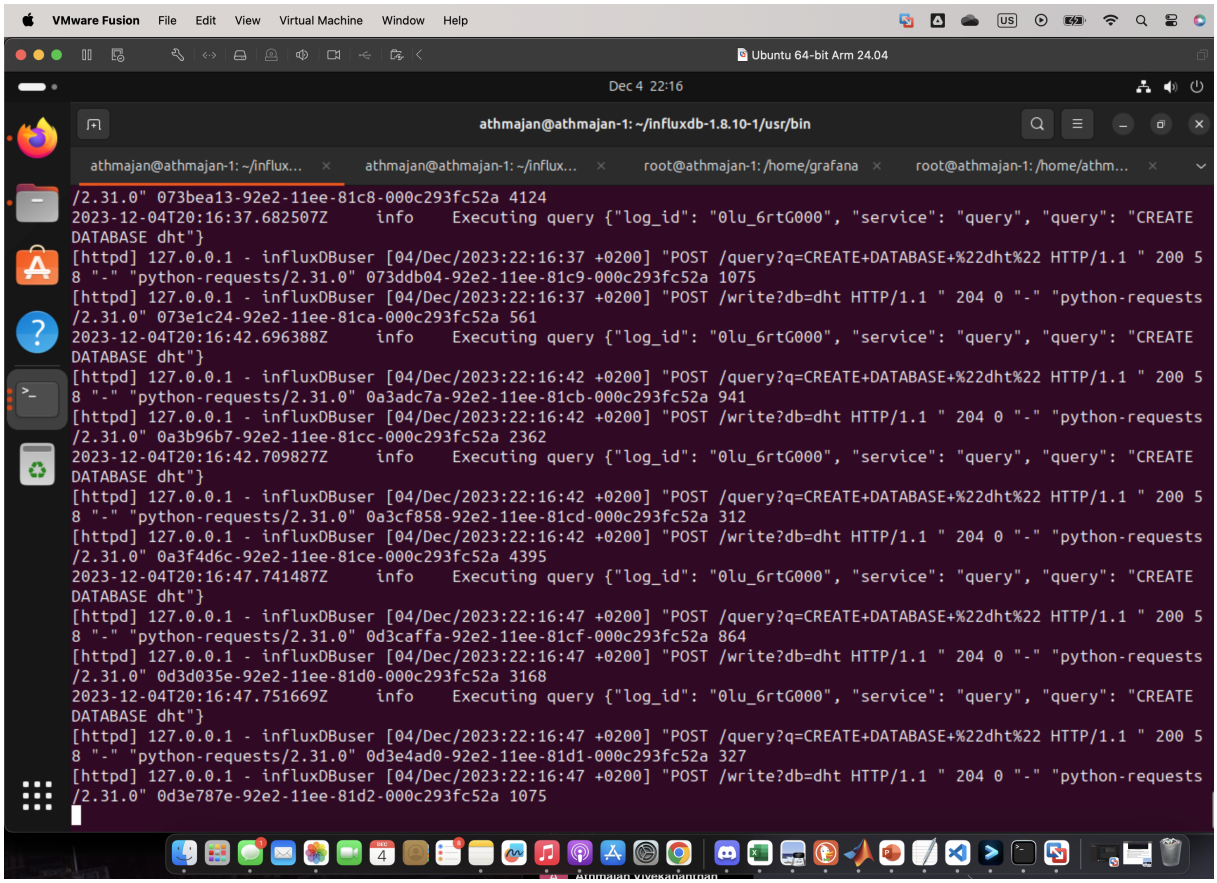


Figure 5.3. Terminal of InfluxDB Server on Guest OS

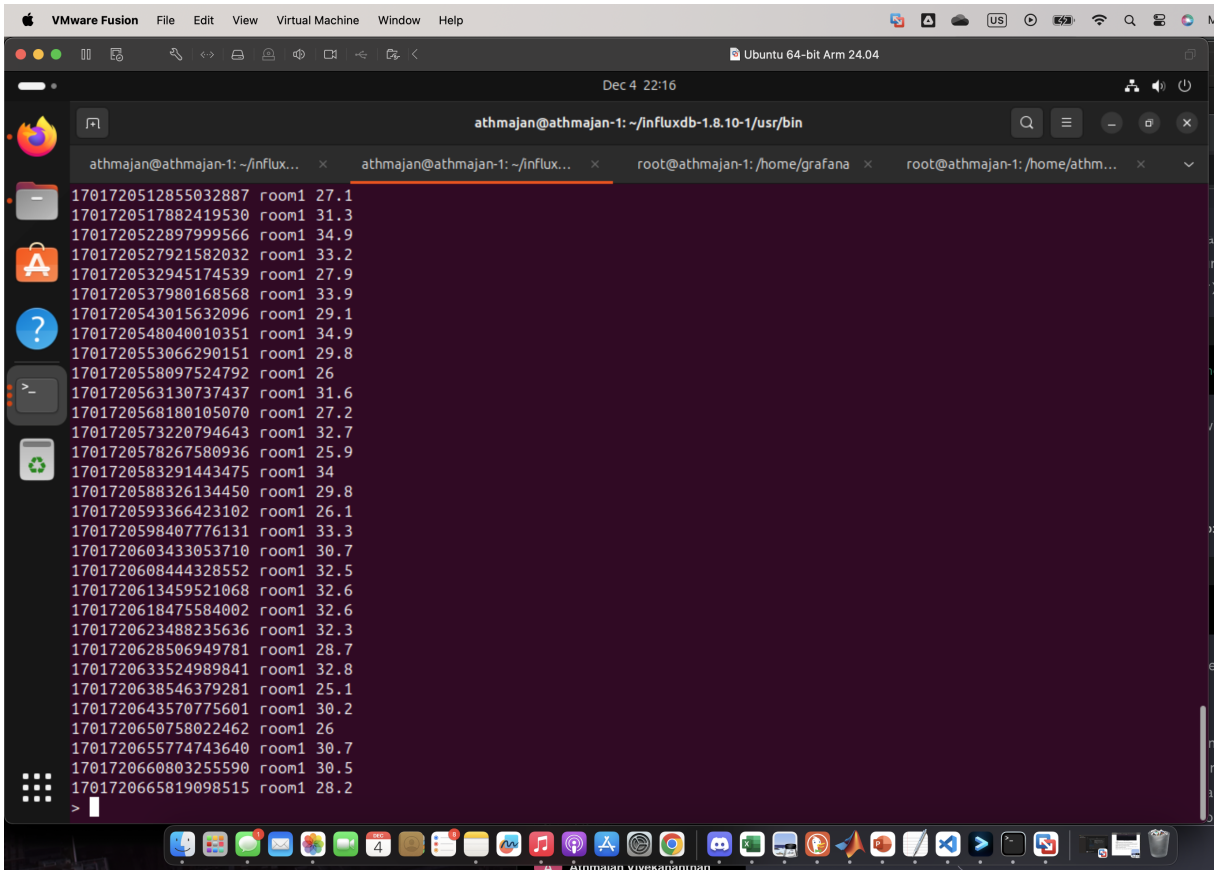


Figure 5.4. Terminal of Influx CLI on Guest OS

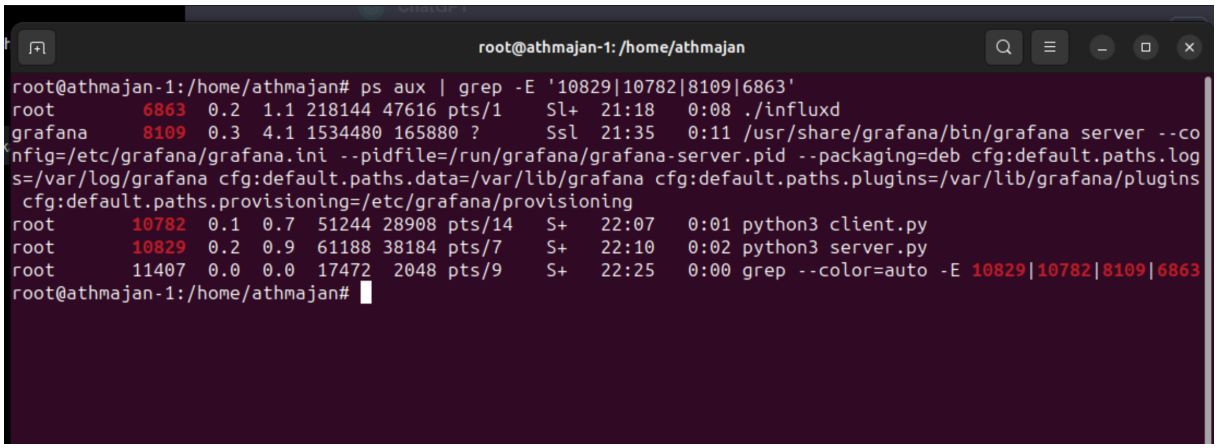


Figure 5.5. Terminal output showing memory usage in Guest OS

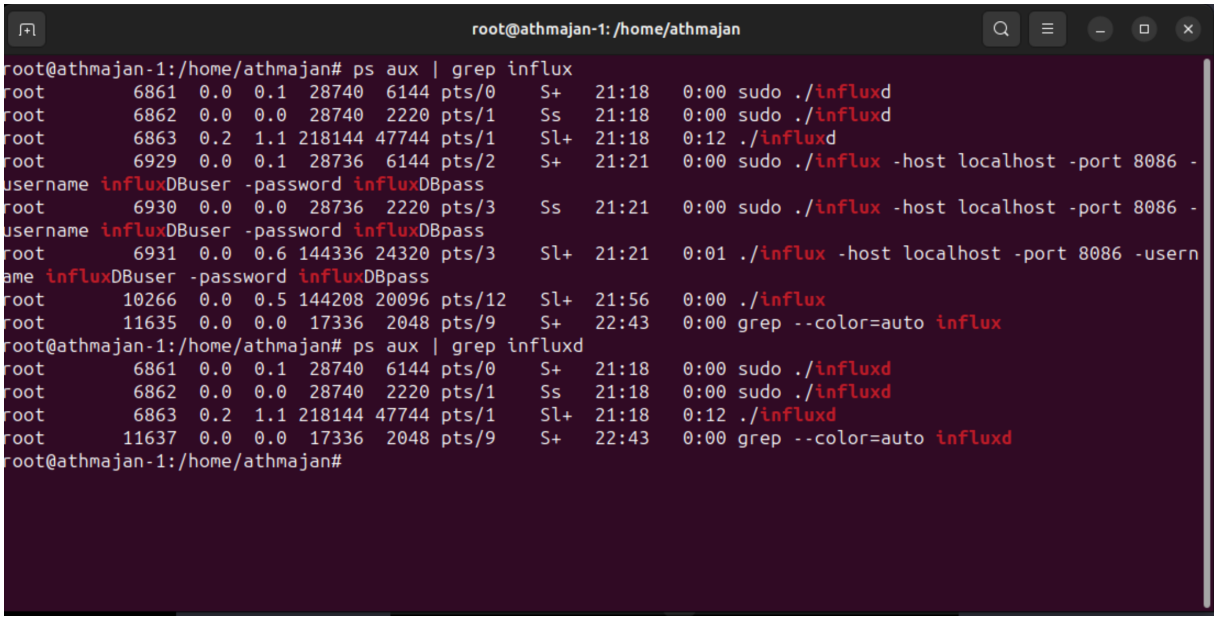


Figure 5.6. Terminal showing memory usage of InfluxDB and dependencies in Guest OS

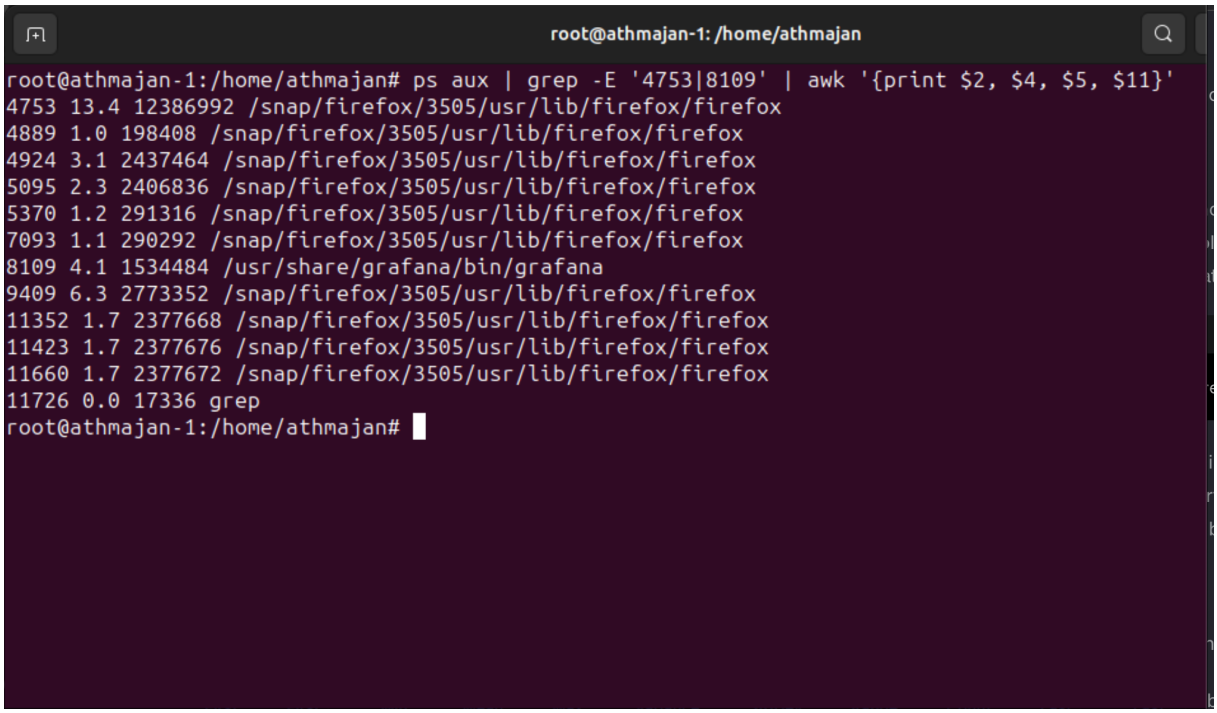


Figure 5.7. Terminal showing memory usage of Grafana and dependencies in Guest OS

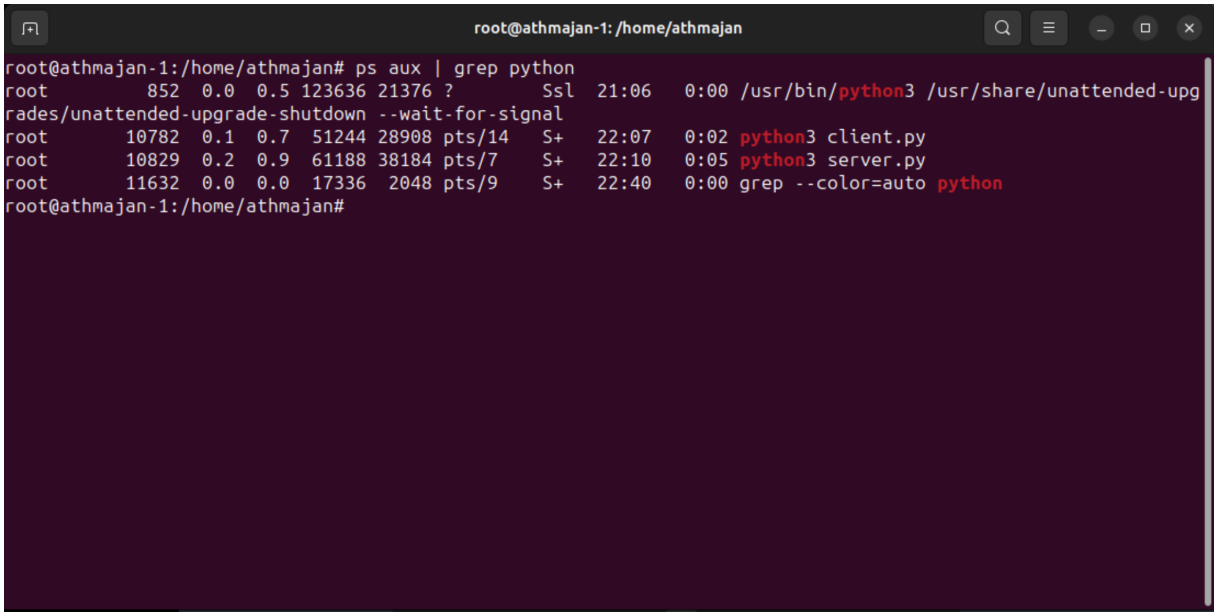


Figure 5.8. Terminal showing memory usage of CoAP Server, Client and dependencies in Guest OS

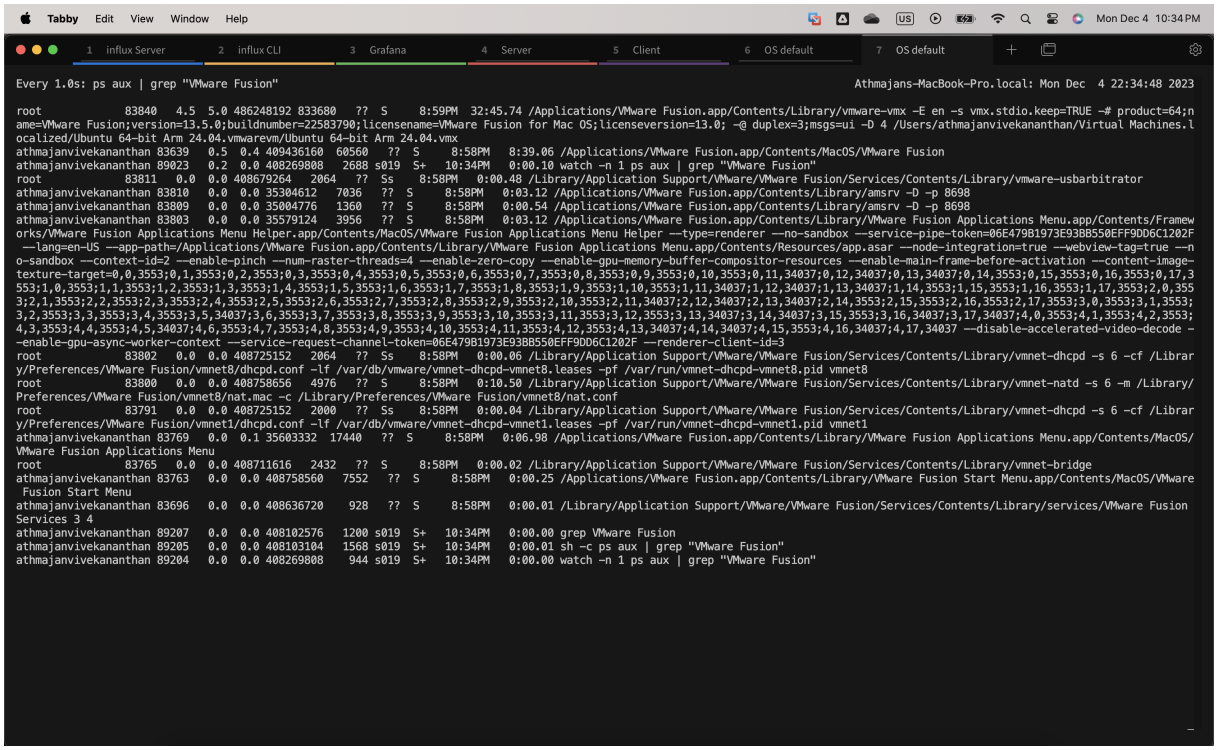


Figure 5.9. Terminal showing memory usage in Host OS while virtualization on hypervisor with Guest OS.

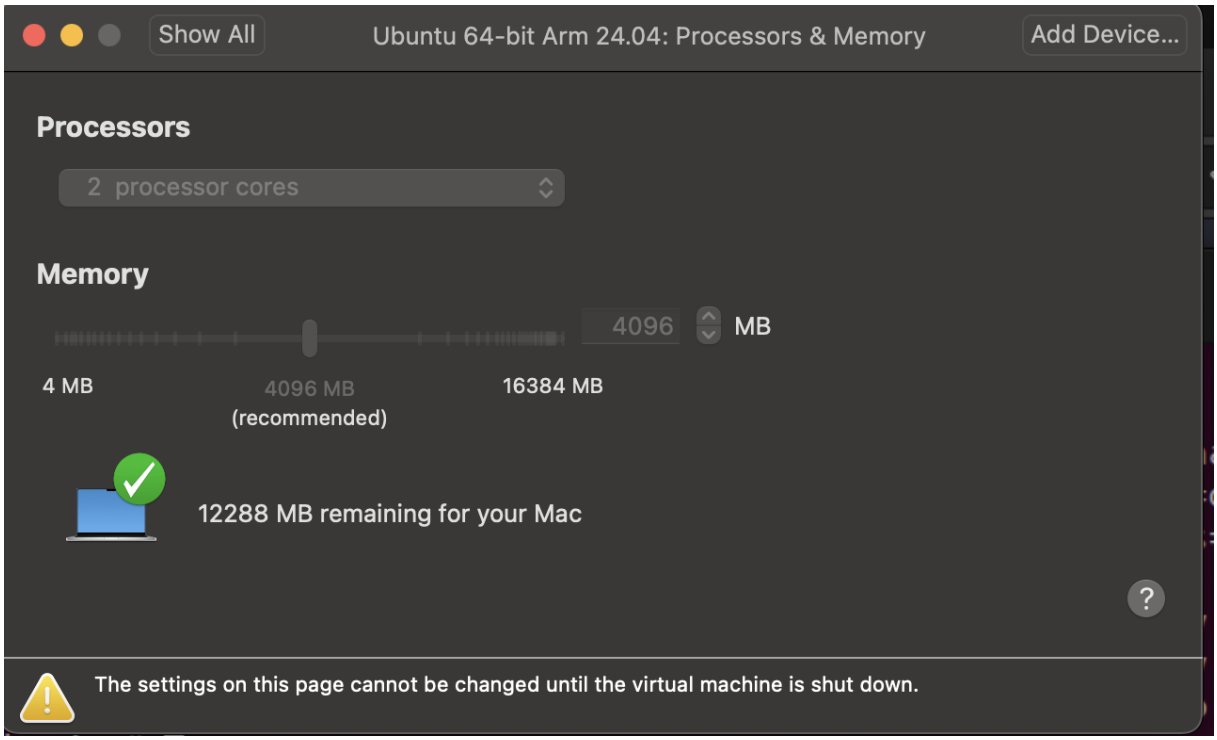


Figure 5.10. Screenshot of VM configuration on CPU and Memory

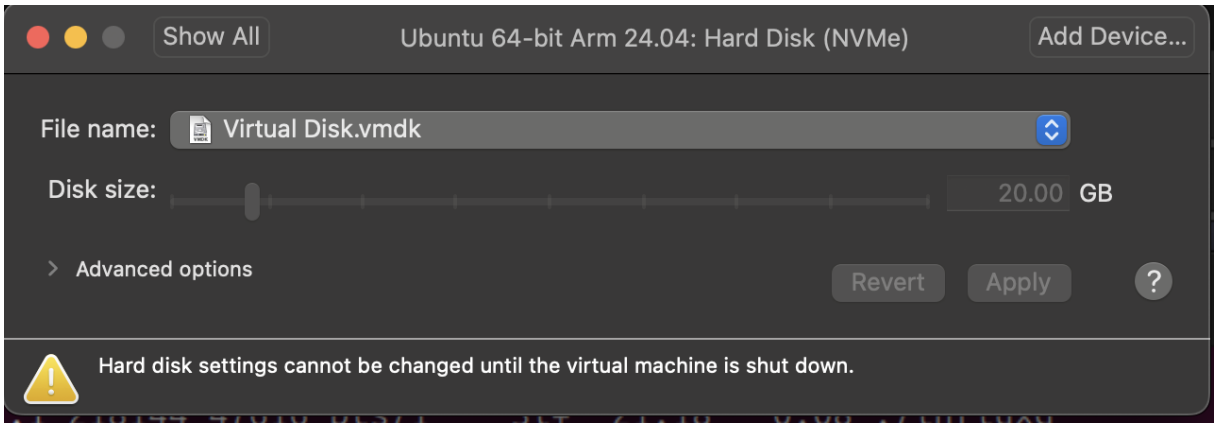


Figure 5.11. Screenshot of VM configuration on Disk Space

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.