DEPARTMENT OF COMMUNICATIONS ENGINEERING

COMMUNICATIONS NETWORKS I (521340S) – Implementation

LABORATORY WORK
2023

GENERAL

The purpose of this laboratory work is to get familiar with popular virtualization technologies and implement different tasks related to exercises. In Exercise 2, Virtualbox is used but students can use any similar tool e.g., Vmware (or any other).

The laboratory work is designed to be carried out in **groups of three students**. A student can also do the laboratory work alone.

A written report should be submitted in the thesis format by **18th of December 2023 (23:59:59)** in pdf format through the **Moodle** portal. Thesis report format can be found at:

- **https://www.oulu.fi/en/for-students/thesis-and-graduation/graduation-masters-degree/masters-thesis#175** [look at -- 6. Thesis work and writing the thesis]

No report will be accepted if it is submitted by e-mail. However, the written report should contain all the answers asked in each exercise with the appropriate figures. The reports will be graded as fail or pass. Students are required to pass the theory exam and the lab work (**either implementation or simulation**) separately to get credit from this course.

The submissions will be reviewed as soon as possible, and a feedback e-mail will be sent to all group members. A list of things will be included with detail instruction if the report is not in the acceptable quality.

Submissions will be tested with a plagiarism detection tool. A student caught with a plagiarized report will lose all accumulated points for the course. Cases of plagiarism may also be reported to the head of the department for further action.

For further information/assistance please contact:

Assistant:
Johirul Islam
e-mail: johirul.islam@oulu.fi

Introduction

To understand the benefits of this lab work, students are requested to read the introduction lecture of the implementation part of Communications Network I (CN1) during lecture 4. This lab work is required to have some basic theoretical knowledge on:

- Cloud, Fog (Edge) and Mist computing,
- Internet of Things (IoT) and traditional data processing mechanisms and protocols, and
- Some architectures regarding monolith, microservices and nanoservies.

In this lab work, students are also required to have basic programming knowledge and a little hands-on experience in command prompt or terminal to execute commands that are required to run an application on their own machine. In some cases, students may require administrative privileges to run applications/services into their computers. However, the whole project is designed to:

(1) get the dummy sensor data (e.g., temperature and humidity),
(2) send the collected data to a time-series database,
(3) store the dummy data into the time-series database, and
(4) display them into graph.

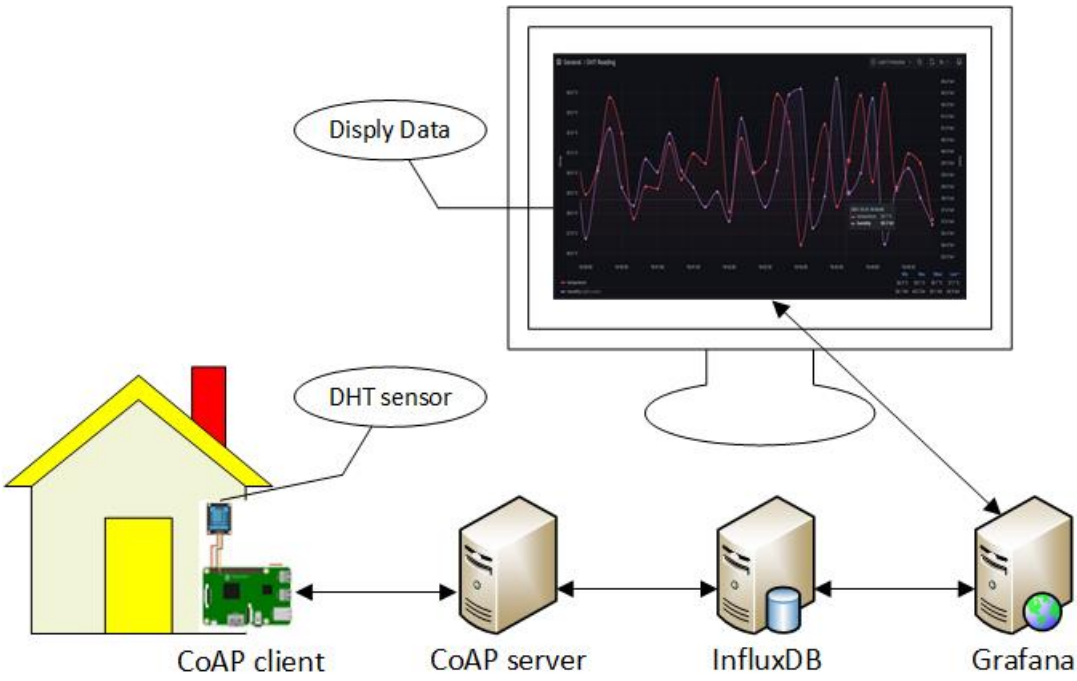

Figure 1. Implementation scenario.

Here, everything will be done locally i.e., on the student's own computer. The client program is going to generate some random/dummy temperature and humidity data instead of using the real DHT sensor. Once the client reads data from the sensor, then the client will send the data to the local server. In this case, Constrained Application Protocol (CoAP) is used in the client and the server side to process the sensed data. The entire application is broken four separate sub-domains commonly known as services:

I. ***CoAP client*** -- is a *CoAP* client-side application which will be used to collect the dummy data and sends to the server,
II. ***CoAP server*** -- is a *CoAP* server-side application which will be used to acquire the dummy data from the client and sends to database,
III. ***influxdb*** -- is a time-series data and will be used to store dummy data into for future reference / use,
IV. ***grafana*** -- is a graph application and will be used to show the dummy data in a graph format.

The local deployment process is divided into three phases:

a. ***Exercise 1*** -- direct deployment on student's PC (bare metal),
b. ***Exercise 2*** -- deployment on hosted-hypervisor (virtual machine), and
c. ***Exercise 3*** -- deployment on containerized system (Docker).

Students' feedback is highly appreciated for future improvement of this lab work. Therefore, students are requested to attach their opinion on the discussion section in their report.

PART I

Deploying Application in Local Server

Exercise 1

1. INTRODUCTION

In this exercise, a student will run/deploy the four services onto your personal computer (PC). Through this exercise, a student is required to have basic understanding of **Python (version 3.10.12)** along with some hands-on experience in **Linux** (preferably, **version 22.04**). Both the *client-side* and *server-side* CoAP applications are developed with ***txThings*** IoT RESTful framework. A student also required to install the ***influxdb*** and ***grafana***.

First, download ***cn1-impl-opt-coap*** as a zip file from github [1] and unzip it. Open your *Terminal/Command Prompt* and make sure you are in ***cn1-impl-opt-coap*** directory (folder) like as follows –

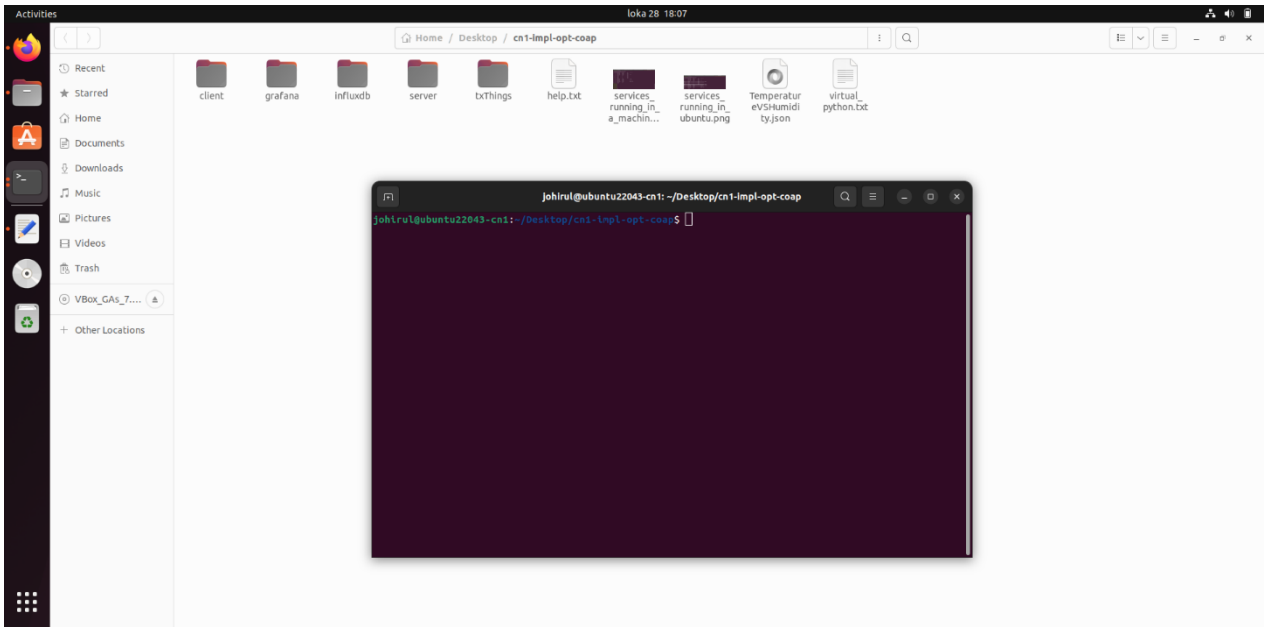

Figure 2. Services to be deployed into local machine.

1.1. Installation:

For a successful deployment, students are required to follow many steps which are described in ***help.txt*** file at the root directory (***cn1-impl-opt-coap***) of the project.

1.1.1 Installing InfluxDB:

In this exercise, InfluxDB 1.8.10 will be used as a time series database. Please install InfluxDB on your host OS. To install it, follow the instructions carefully at InfluxDB's documentation (find -- **Are you interested in InfluxDB 1.x Open Source?** -- section)

- Choose **InfluxDB v1.8.10** (from the **Version** dropdown menu)
- Choose anyone that satisfy your host OS system (from the Platform dropdown menu)

For instance, for Ubuntu OS

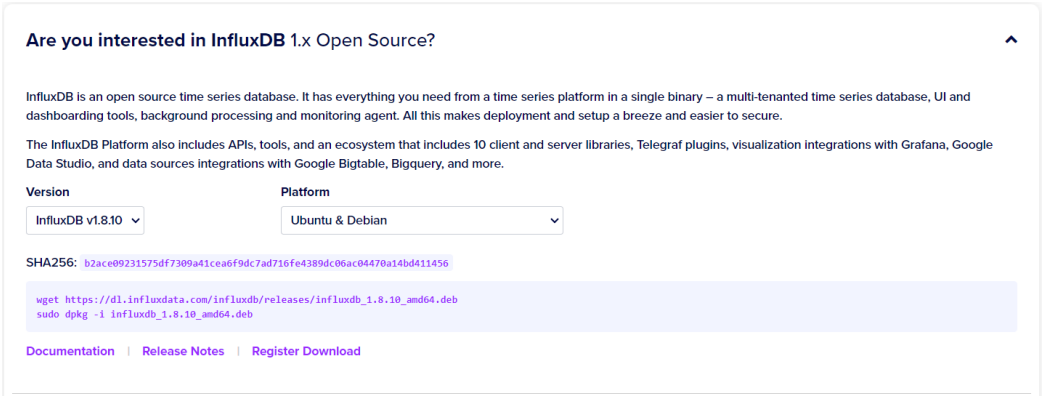

Figure 3. InfluxDB 1.x.

```
wget https://dl.influxdata.com/influxdb/releases/influxdb_1.8.10_amd64.deb
sudo dpkg -i influxdb_1.8.10_amd64.deb

# Run the influxDB daemon as follows
sudo influxd
```

By default, port 8086 is allocated for ***influxdb***

sudo lsof -nP -i4TCP:8086| grep LISTEN

```
netstat -ptuln | grep 8086
```

Now execute influxdb commands as instructed in (make sure you are in 'influxdb' directory)

(a) **new/*1_create_user_db.txt***; and
(b) **new/*2_influxdb_queries.txt***.

You can check the expected output in *the* ***new/help*** directory under ***influxdb***.

1.1.2 Installing Grafana:

Now install Grafana 10.2.0 to visualize data from InfluxDB. Select the package according to your host OS like as follows --

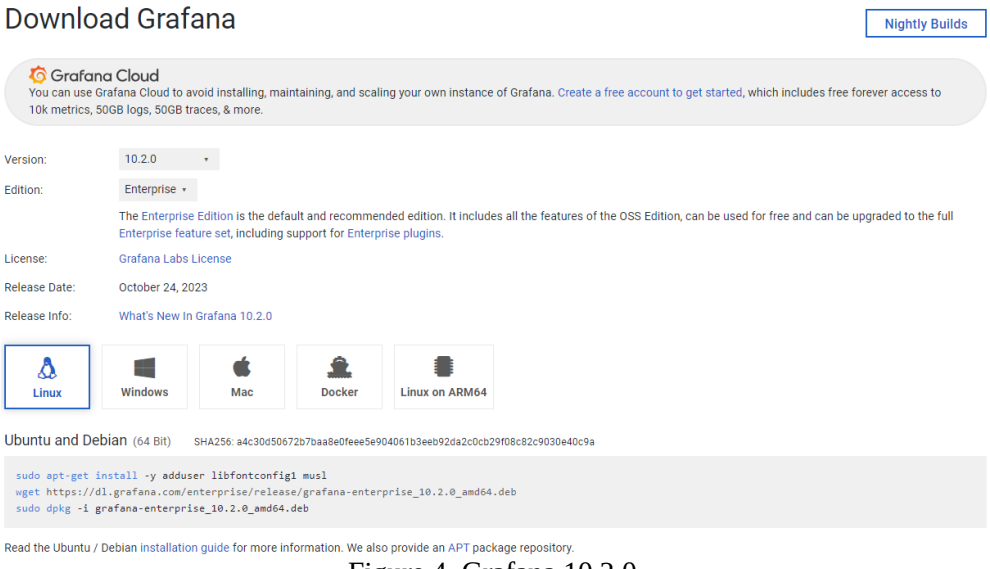

Figure 4. Grafana 10.2.0.

Now move to ***grafana*** directory and execute following commands (you can check installation guide if you have some other OS):

```
sudo apt-get install -y adduser libfontconfig1 musl
wget https://dl.grafana.com/oss/release/grafana_10.2.0_amd64.deb
sudo dpkg -i grafana_10.2.0_amd64.deb

# Start Grafana server
sudo service grafana-server start
```

You can check the status of the service as follows:

```
service grafana-server status
```

You can get output like as ***1.1_check_status_of_grafana_server.png*** (check ***help*** directory)**.** If the grafana-server is up and running, then the status should be ***Active: active (running)***. Normally grafana-server runs at port ***3000***.

```
netstat -ptuln | grep 3000
```

Open your browser and login to the grafana server at http://localhost:3000 (or 127.0.0.1:3000).

1.1.3 CoAP server-client setup:

- System setup:
  In this exercise, both the server and client require python3 (**3.10.12**) to run the services as the server/client. Following instruction will help you to install it into your host OS --
  - Installing python and pip:
    - If you already have ***python 3*** running on your pc, please skip this section.
      - You can run the command mentioned in version checking part to verify which ***python 3*** you are currently using.
    - If you do not have ***python 3*** on your pc, then follow the instructions to install it.
      - Linux User: Run the following command on your Terminal window.

```
sudo apt-get install python3 python3-pip python3-dev
```

      - Mac and Windows User: this guideline [2] will help you to get install ***python 3*** on Mac and Windows.
  - Version checking:
    Check your ***python*** and ***pip*** (python package installer). Run the following commands --
    - ✓ *python*:

```
python3 --version
```

    - ✓ *pip:*

```
pip3 --version
```

- Running the CoAP server:

Now move to the ***server*** directory and install CoAP server's requirements with pip:

```
pip3 install -r requirements.txt
```

pip will install *influxdb*, *service_identity* and *twisted* into host OS. To run the server run following command:

```
python3 server.py
```

By default, CoAP server exposes port 5683. Now, check if it is open or not:

```
netstat -ptuln | grep 5683
```

- Running the CoAP client:

Make sure you have activated the virtual environment under ***cn1-impl-opt-coap*** directory. Now move to the ***client*** directory and install CoAP client's requirements with pip:

```
pip install -r requirements.txt
```

pip will install *ipaddress* and *twisted* under python. To run the server run following command:

```
Python3 client.py
```

By default, CoAP client expose/open the port 61616. Now, check if it is open or not:

```
netstat -ptuln
```

1.2. Monitoring:

To initiate the monitoring process, your terminal and run following command:

```
netstat -ptuln
```

You will something as follows:

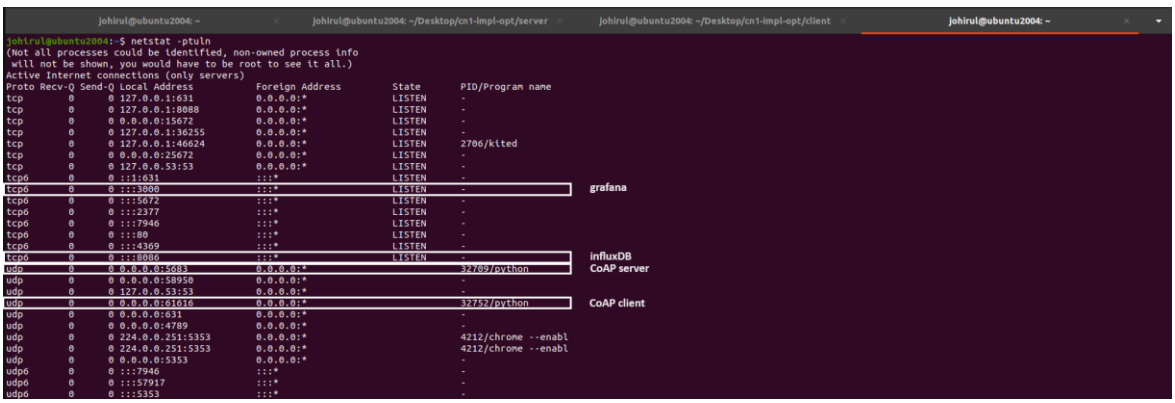

Figure 5. Exposed ports by different services in a local machine.

Now open a browser on your development PC and try to explore the grafana-server with --

```
http://127.0.0.1:3000/
or,
http://localhost:3000/
```

You will get a UI like as:

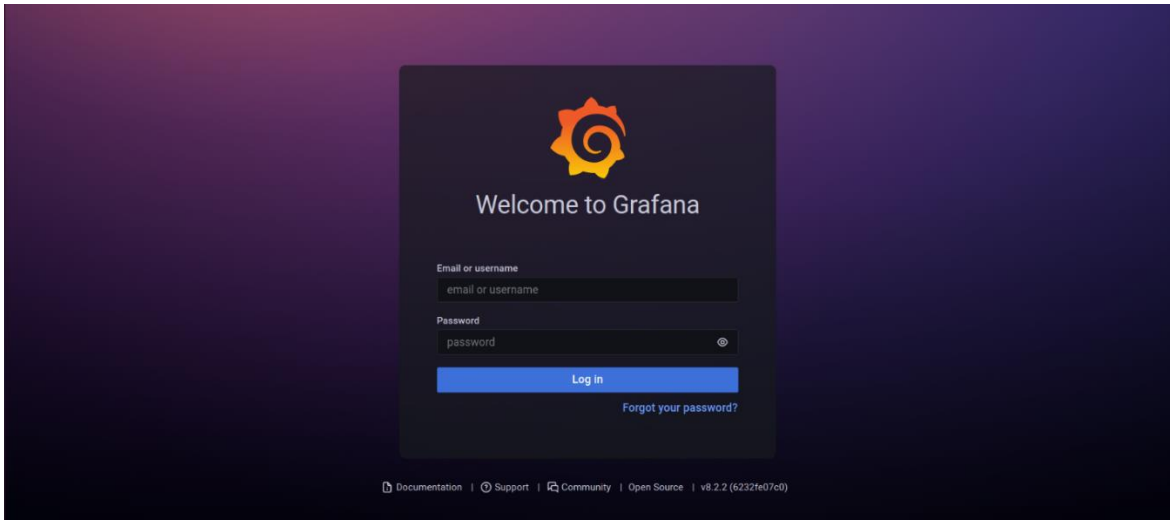

Figure 6. Gafana login screen.

The default ***username*** and ***password*** are ***admin*** and ***admin***. Grafana will suggest you to change the default password once you logged in successfully which you can skip for now. Now follow the instructions (available at **grafana/new** directory) to add the InfluxDB as a data source --

- Click on menu beside **Home** (3 bar icon on the left side) and click on **Connections** (look at ***3.1-new.png***).
- Click on Add New connection (from left side) and search for ***InfluxDB*** at the search bar. Click on **InfluxDB** as the **Data sources** (look at ***3.2-new.png***).
- Click on the **Add new data source** (from the right as shown in ***3.3-new.png***)
- In the **Name** field, type **DHT** and put http://localhost:8086 (or http://127.0.0.1:8086) at the ***URL*** (as shown in ***3.4-new.png***).

- Now, in the InfluxDB Details section put ***dht***, ***influxDBuser*** and ***influxDBpass*** at the Database, User and Password fields respectively and click on **Save & test** button to save the configuration (as shown in ***3.5-new.png***).

You will see a confirmation message as shown in ***3.6-new.png***. Anyway, to add a dashboard follow the instruction as follows --

- Click on **3 bars** beside **Home** and then click on **Dashboards** (like as ***4.0-new.png***).
- Click on **New** dropdown menu and then click on **Import** (as shown in ***4.1-new.png***).
- Click on **Upload dashboard JSON** file (as shown in ***4.2-new.png***).
- Choose **TemperatureVSHumicity.json** and then click on **Open** to upload it (as shown in ***4.3-1-new.png***).
- Choose **DHT** from the dropdown menu and click **Import** (as shown in ***4.3-2-new.png***).

You will get a live graph like as follows which will refresh every 5 seconds --

Figure 7. DHT dummy data.

1.3. Tasks

With a fully functioning system, go ahead and perform the following tasks:

(1) Take a screenshot and add to your report by showing the dummy temperature and humidity data.

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

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

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

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

PART II

Hypervisor-based Virtualization (Hosted Hypervisor)

Exercise 2

2. INTRODUCTION

Software Development Life Cycle (SDLC) is usually composed of numerous steps from planning to maintenance and follows a strategic rule during the entire development life cycle.

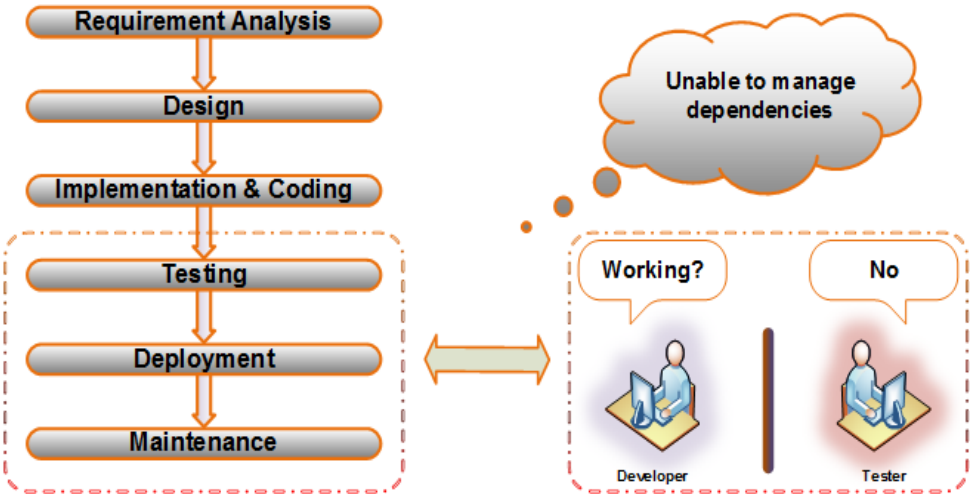

Figure 8. Typical problem space in SDLC.

In SDLC, it is common to encounter problems in between the testing and maintenance phases. To realize this problem, let's assume that you need to have a quick change in the server or the client code. Or you need move all the four services that made for ***Exercise 1*** into some other computer. In this case, you may not be able to run these directly into the new computer. This is mainly because of the new computer is not aware of the requirements of each service that you deployed at ***Exercise 1***. To run these services into the new computer, you need to do everything from the very beginning that you did at ***1.1*** in the ***Exercise 1***.

Software personnel need a novel solution that can mitigate this problem. Virtualization has been introduced in the software industry to overcome these problems. It allows a user to separate the operating system from the underlying hardware [3]. It creates a virtual version of a resource such as a server, storage device, network or even an operating system. There are many virtualization techniques that exist e.g., network virtualization, server virtualization and so on. But for now, you will focus on server virtualization. ***Hypervisor-based*** virtualization and ***container-based*** virtualization are the two most popular techniques for server virtualization.

Hypervisor-based virtualization requires a hypervisor (a management software such as VirtualBox). It allows a user to install a new operating system (known as guest operating system) on top of a hardware with or without a host operating system. It has two types: (1) ***Type 1*** or Bare Metal Hypervisor (without a host OS); and (2) ***Type 2*** or Hosted Hypervisor (with a host OS). In this part you will focus on Type 2 or Hosted Hypervisor-based virtualization.

2.1. Installing Hypervisor:

A virtual guest OS could be a solution to the problem that addressed in the introduction. In this case, you can move this virtual OS to any other system having hypervisor and run the application without any hassle. For this lab exercise we prefer to install ***Oracle VirtualBox (version 7.0)***.

You can install as many guest OS as you need on that. Let's install Oracle VirtualBox on top of your host operating system –

- Download VirtualBox 7.0 (7.0.4 - released November 18, 2022) from here [4].
- Install VirtualBox 7.0

When you run the VirtualBox into your local PC you will get something like this –

Figure 9. User Interface of Oracle VirtualBox 7.0.

2.2. Installing Guest OS:

To run a virtual machine, you need a guest OS. In this lab, you can use Ubuntu 18.10. Now download from the following link and install it into VirtualBox –

- Download Ubuntu 22.0 ***iso*** file from here [5].

When you start the newly created machine you will get something similar to this:

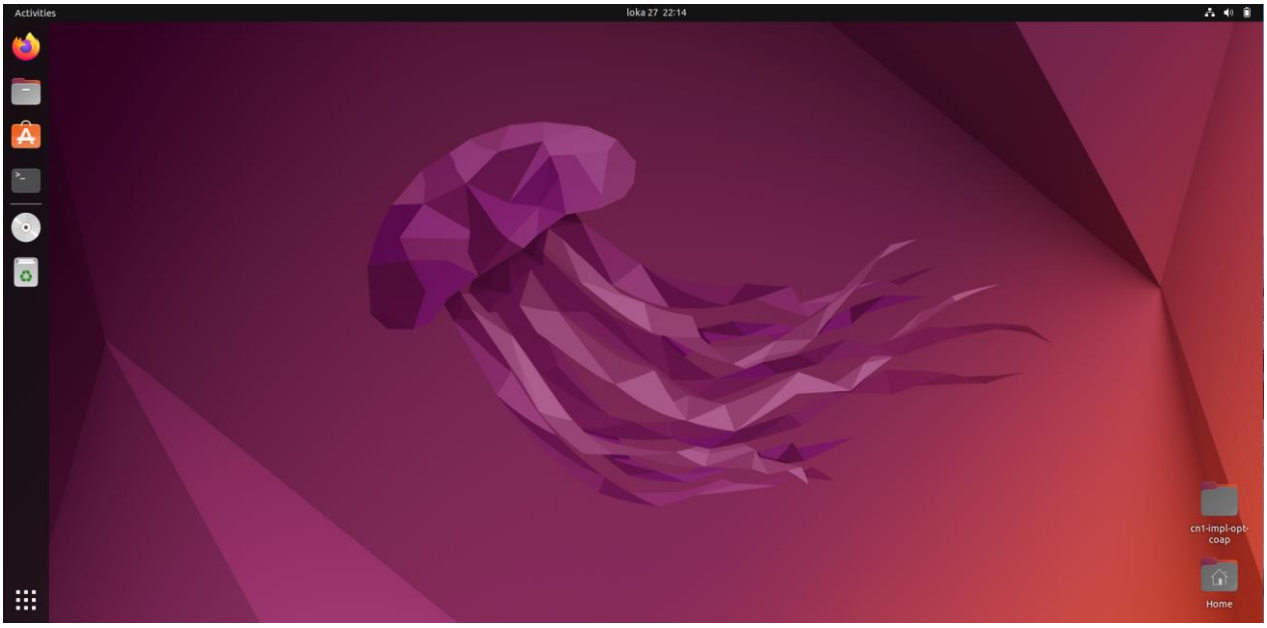

Figure 10. Ubuntu 22.04 OS.

2.3. Running Application:

Before running all the 4 services, you need to install influxdb, grafana-server and python 3, pip3 & other CoAP related libraries (required by server & client) again into your virtual/guest OS. Repeat instructions to install InfluxDB, Grafana and Python3 & Pip3 for CoAP in guest OS --

- ***influxdb*** – see subsection 1.1.1
- ***grafana*** – see subsection 1.1.2
- ***python3, pip3 and CoAP libraries for server & client*** – see subsection 1.1.3

If the requirements are there inside the new virtual/guest OS, then run 4 services with the command that mentioned in section 1.1. Finally, monitor the application on your guest OS as mentioned in section 1.2.

2.4. Task?

Your task is to export the guest OS in ***ova*** format along with the application so that it could be run from any other hypervisor on any machine.

Question 4: Explain pros and cons of guest OS?

Question 5: Differentiate Type 1 and Type 2 hypervisor with necessary figures?

Question 6: What is shell scripting and why it is important? Make a shell script to run the 4 services into your guest OS and explain the commands in a few sentences.

Question 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.

PART III

Container-based Virtualization

Exercise 3

3. INTRODUCTION

In exercise 2, you have deployed all 4 services into Ubuntu 22.04 guest OS. Normally, an OS requires some hardware (HW) resources such as CPU, memory (RAM), storage, network interface etc. During the installation process, you must specify the amount of these HW resources to run OS. However, a virtual/guest OS shares these HW resources with host OS and acts like a virtual machine (VM). An Ubuntu VM requires at least 2GB memory and 1 CPU apart from a huge free storage (more than 5GB). Since a VM shares HW resources with a physical/host machine, the host machine may get stuck easily when a VM starts running on it. Thus, a VM is mostly resource consuming.

Containerization is another virtualization technique which is less resource consuming. In this technique, it wraps up an application/service with its dependencies to run it into a system requiring less resources. Docker is the most widely used containerization technology to build, run and deploy containerized applications. In this exercise, you will use some built-in docker containerized images for ***influxdb*** and ***grafana-server*** offered by InfluxDB and Grafana communities respectively. However, you will make container version of the CoAP server and CoAP client services with their dependencies (***server/requirements.txt*** & ***client/requirements.txt***) that you installed with ***pip3*** in the ***Exercise 1*** and ***Exercise 2***.

3.1. Installing Docker:

There are a few guidelines to install Docker for several machines. Run the following commands to install docker in Linux-based (Ubuntu) machine and reboot the system (once the installation completed) –

```
# Add Docker's official GPG key:
sudo apt-get update
sudo apt-get install ca-certificates curl gnupg
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
/etc/apt/keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg

# Add the repository to Apt sources:
echo \
  "deb [arch="$(dpkg --print-architecture)" signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/ubuntu \
  "$(. /etc/os-release && echo "$VERSION_CODENAME")" stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt-get update

# Install docker
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin
docker-compose-plugin -y

# add $USER to docker group
sudo usermod -aG docker $USER
```

You can find the installation guide for Windows and Mac as well. However, after a successful installation, let's check the docker version with –

```
docker --version
```

3.2. Pulling influxdb and grafana images:

Docker Hub is the docker images repository where you can pull or push your docker images. Normally, you can pull any docker image from docker hub with ***docker pull <username>/<image_name>:version*** command. In some cases, username can be omitted.

3.2.1 Pulling InfluxDB:

In ***Exercise 1***, you have used influxdb 1.8.10. So, to pull the ***influxdb*** of version 1.8.10, run the following command (read influxdb for instruction and tags tab for version):

```
docker pull influxdb:1.8.10
```

Check if it is pulled or not:

```
docker images
```

3.2.2 Pulling Grafana:

During the ***Exercise 1***, you have used the grafana-server version 10.2.0 (check the version with: ***grafana-cli -h*** command). Therefore, to pull grafana-server 10.2.0 from docker hub run following command (check grafana for instruction and tags tab for version):

```
docker pull grafana/grafana:10.2.0
```

Check if it is pulled or not:

```
docker images
```

3.3. Running influxdb and grafana images:

In this exercise you will have access the ***influxdb*** and ***grafana*** from your host machine through their default ports i.e., ***8086*** and ***3000*** respectively. Before running these two services, make sure that the aforementioned ports are free to use. Please check if they are free or not with following command:

```
netstat -ptuln
```

You can also find by piping through ***grep*** with “3000\|8086”:

```
netstat -ptuln | grep “3000\|8086”
```

However, if there is any line containing ***3000*** then ***grafana*** (or some other e.g., nodejs) is running on ***3000***. Similarly, if there is any line containing ***8086*** then ***influxdb*** (or some other) is running on ***8086***. Check which service is listening on a specific port. If these ports are occupied, then free them first before running the ***influxdb*** or ***grafana*** through docker container. However, if you know which service is occupied with port ***3000*** and ***8086***, then stop them ***service*** or ***systemctl*** or ***kill*** command. In Exercise 1 and Exercise 2, ***3000*** is occupied by ***grafana-server*** service whereas ***8086*** is occupied by ***influxdb*** service. Let’s stop them with service command:

```
sudo service grafana-server stop
sudo service influxdb stop
```

Make sure that ***grafana-server*** and ***influxdb*** services are down i.e., status will be ***inactive (dead)***. Please look at the line that starts with ***Active***. You can check the statuses with:

```
service grafana-server status
service influxdb status
```

Also make sure that there is no line having 3000 & 8086 for the following command:

```
netstat -ptuln | grep "3000\|8086"
```

3.3.1 Running InfluxDB with docker image:

A docker image can be run through docker run command. Let's get help from docker:

```
docker run --help
```

Now, check if port ***8086*** is free or not (find command in section 3.3). If it is free, then run following single command to start the ***influxdb*** as a service (add -d for detach mode before -p):

```
docker run -p 8086:8086 -e INFLUXDB_DB='dht' -e INFLUXDB_ADMIN_USER='influxDBuser' -e
INFLUXDB_ADMIN_PASSWORD='influxDBpass' -e INFLUXDB_HTTP_AUTH_ENABLED='true'
influxdb:1.8.10
```

Here,

- ***docker run*** -- is required to run a docker container from an image (here, ***influxdb:1.8.0***).
- ***-p 8086:8086*** -- will expose host's ***8086*** port and redirect incoming request on host's port ***8086*** to container's port ***8086*** (in the form of <***host_port***>:<***container_port***>).
- ***-e INFLUXDB_DB='dht'*** -- will creates a database named ***dht*** in ***influxdb*** database system.
- ***-e INFLUXDB_ADMIN_USER='influxDBuser'*** -- will create a user named ***inflxDBuser*** in ***influxdb*** database system
- ***-e INFLUXDB_ADMIN_PASSWORD='influxDBpass'*** -- will set password for ***inflxDBuser*** user in ***influxdb*** database system.
- ***-e INFLUXDB_HTTP_AUTH_ENABLED*** -- will enable the ***influxdb*** database system so that influx queries can be executed through http protocol. This is required because ***grafana-server*** basically sends query requests to ***influxdb*** server in http query format.
- ***influxdb:1.8.10*** -- the image (last argument) that is going to run with ***docker run*** command.

You can check if port ***8086*** is exposed or not on your host:

```
netstat -ptuln | grep -w ":8086"
```

If you now, get a line containing ***:8086*** then port ***8086*** is exposed and occupied with ***influxdb*** container. However, you will not get it through ***service influxdb status*** command because 8086 is not exposed through a host service. You can get it from the running docker containers and port mapping:

```
docker ps
```

If you run above command, then you find details about currently running docker images. Now, look at the IMAGE and PORTS portions. You will see that ***influxdb:1.8.10*** and ***0.0.0.0:8086->8086/tcp, :::8086->8086/tcp*** (port mapping is in the form of ***HOST_PORT->CONTAINER_PORT***) are in aligned with IMAGE and PORTS tabs.

Run following command to enter into the influxdb shell (look at github if you face any problem):

```
docker exec -it <CONTAINER_ID_or_NAME> /bin/bash
```

Replace ***<CONTAINER_ID_or_NAME>*** with real container ID or container name of the ***influxdb*** that got from above ***docker ps*** command. Now, run the following command inside ***influxdb*** container terminal:

```
influx -username influxDBuser -password influxDBpass
help
show databases
show users
exit
```

You get ***dht*** and ***influxDBuser*** for the above ***show databases*** and ***show users*** commands respectively. Now, exit from the ***influxdb*** terminal:

```
exit
```

3.3.2 Running Grafana with docker image:

Check if port ***3000*** is free or not (find command in section 3.3). If it is free, then run following single command to start the ***grafana-server*** as a service (add -d for detach mode before -p):

```
docker run -p 3000:3000 grafana/grafana:10.2.0
```

Here,

- ***-p 3000:3000*** – will expose host's ***3000*** port and redirect incoming request on host's port ***3000*** to container's port ***3000*** (in the form of ***<host_port>:<container_port>***).

You can check if port ***3000*** is exposed or not on your host:

```
netstat -ptuln | grep -w ":3000"
```

If you now get a line containing ***:3000*** then port ***3000*** is exposed and occupied with some service. However, you will not get it through ***service grafana-server status*** command because ***3000*** is not exposed through a host service. You can get it from the running docker containers and port mapping:

```
docker ps
```

If you find ***grafana/grafana:10.2.0*** and ***0.0.0.0:3000->3000/tcp, :::3000->3000/tcp*** under IMAGE and PORTS, then it's mean that ***grafana*** will be accessing with your local machine through port ***3000***. Now open your browser and access the grafana UI (look at section 1.2).

3.4. Creating Dockerfiles for CoAP server and CoAP client containers:

So far, you use prebuilt docker images provided by InfluxDB and Grafana. Now, it is time to make a virtual version of the CoAP server and the CoAP client with Docker.

3.4.1 Containerizing CoAP server with docker:

Make sure you are in the ***server*** directory which is located under the project's root directory i.e., ***cn1-impl-opt-coap***. To make the CoAP server image, make a file named ***Dockerfile*** (without any extension) and put following things and then save it –

```
FROM python:3.10.12
WORKDIR /server
COPY . /server
RUN pip3 install --upgrade cython && \
    pip3 install -r requirements.txt
EXPOSE 5683
CMD ["python3", "server.py"]
```

Description:

- ***FROM*** -- declares the base image (it should be the very first line for every image for each custom image). The custom image of your CoAP server should start with ***python:3.10.12*** since the CoAP server was built with python 3.10.12.
- ***WORKDIR*** -- declares the working directory of the CoAP server as **/server** into the container. Once you login to the container (CoAP server) it will directly locate this directory.
- ***COPY*** -- declares to copy everything from the current directory (where Dockerfile is located) to **/server** location of the application into the container.
- ***RUN*** -- tells docker to use shell and run the command that you put in console/terminal to execute something. In this case, it is performing ***pip3 install -r requirements.txt*** (which you did in your shell) that means it is installing the requirements of the CoAP server that mentioned in the ***requirements.txt*** file.
- ***EXPOSE*** -- declares to open the container port. Here, 5683 is the container port. You cannot talk to your container from another container if you do not expose it.
- ***CMD*** -- declares the command to run the application once the application is up. You could put also –

```
CMD python3 server.py
```

All the above docker commands i.e., FROM, WORKDIR, COPY, RUN, EXPOSE and CMD etc. are case insensitive. Therefore, **from** and **FROM** are equal in ***Dockerfile***.

3.4.2 Containerizing CoAP client with docker:

Make sure you are in the ***client*** directory under the project's root directory i.e., ***cn1-impl-opt-coap***. To make the CoAP client image, make a file named ***Dockerfile*** (without any extension) and put following things and then save it –

```
FROM python:3.10.12
WORKDIR /client
COPY . /client
RUN pip3 install --upgrade cython && \
    pip3 install -r requirements.txt
EXPOSE 61616
CMD ["python3", "client.py"]
```

Description: Please have a look at the description of subsection 3.4.1 for more in detail.

3.5. Building Container:

First let's get help from docker:

```
docker build --help
```

Now, use the following command which is basically used to an image build from a ***Dockerfile*** –

```
docker build -t <USERNAME>/<IMAGE_NAME>:<VERSION> -f <DOCKERFILE_NAME> <PATH>
or,
docker build -t <IMAGE_NAME> .
```

Here,

- ***-t*** -- declares the tag of the image,
- <***USERNAME***> -- docker hub username. This is optional for local cluster but required when the image is needed to upload/download to/from docker hub with ***docker push/pull*** command,
- <***IMAGE_NAME***> -- the name of image (a meaningful name based on the application),
- <***VERSION***> -- the version of the image (this is optional). Put a version if you have many of the same image,
- ***-f*** -- used to specify a ***Dockerfile*** name (skip flag and associated argument i.e., ***-f*** <***DOCKERFILE_NAME***> if it is ***Dockerfile***),
- <***PATH***> -- defines the location of the ***Dockerfile*** (use . i.e., a dot if the ***Dockerfile*** is in the current working directory).

3.5.1 Building CoAP server with docker:

Make sure you are in the ***server*** directory which is located under the project's root directory i.e., ***cn1-impl-opt-coap***. Run the following command to build the CoAP server image (make sure ***Dockerfile*** for the ***server*** image is inside the ***server*** directory) –

```
docker build -t dht_coap_server:1.0 .
```

Now, check if the image is built or not (with **IMAGE** *dht_coap_server* and **TAG** *1.0*):

```
docker images
```

3.5.2 Building CoAP client with docker:

Make sure you are in the ***client*** directory which is located under the project's root directory i.e., ***cn1-impl-opt-coap***. Run the following command to build the CoAP client image (make sure ***Dockerfile*** for the ***client*** image is inside the ***client*** directory) –

```
docker build -t dht_coap_client:1.0 .
```

Now, check if the image is built or not (with **IMAGE** *dht_coap_client* and **TAG** *1.0*):

```
docker images
```

3.6. Running Container:

First, let's get help from docker:

```
docker run --help
```

3.5.1 Running CoAP server with docker:

Run the following command to run the CoAP server image –

```
docker run -p 5683:5683 --net='host' dht_coap_server:1.0
```

docker run -p 61616:61616 --net= bridge dht_coap_client:1.0

Now, open a new terminal and check if the image is running or not:

```
docker ps
```

3.5.2 Running CoAP client with docker:

Run the following command to run the CoAP client image –

```
docker run -p 61616:61616 --net='host' dht_coap_client:1.0
```

Now, open a new terminal and check if the image is built or not:

```
docker ps
```

3.6. Monitoring:

Open a new console / terminal without closing the previous one. Now list out the running containers –

```
docker ps
```

Now, check if each of ***influxdb:1.8.0***, ***grafana/grafana:8.2.3***, ***dht_coap_server:1.0*** and ***dht_coap_client:1.0*** containers are running or not.

```
netstat --ptuln | grep "3000\|5683\|8086\|61616"
```

Observe if all ports 3000 (for ***influxdb***), 5683 (for ***dht_coap_server***), 8086 (for ***grafana***) and 61616 (for ***dht_coap_client***) are exposed or not.

3.7. Testing:

Now open your browser and explore http://localhost:3000 and then login with default username and password i.e., ***admin*** and ***admin*** respectively. Now add data source and make a dashboard to show the dummy temperature and humidity data by pick the data from the database (look at the section 1.2 in Exercise 1).

3.8. Task?

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

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

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

Question 9: Why an image starts with ***FROM*** instruction?

Question 10: How a host machine can talk to a container?

Question 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**.

DISCUSSION

Students are requested to discuss the learning outcome from this lab work including the difficulty level. Also include the suggestions for future versions of the work.

REFERENCES

[1] "cwc-ns/cn1-impl-opt-coap", GitHub, 2023. [Online]. Available: https://github.com/cwc-ns/cn1-impl-opt-coap. [Accessed: 15- Oct- 2023].

[2] “Python tutorial for Beginners 1: Install and setup for Mac and windows,” YouTube, 17-May-2017. [Online]. Available: https://www.youtube.com/watch?v=YYXdXT2l-Gg&t=85s. [Accessed: 15-Oct-2023].

[3] "Introduction to Virtualization", YouTube, 2023. [Online]. Available: https://www.youtube.com/watch?v=zLJbP6vBk2M&t=112s. [Accessed: 15- Oct- 2023].

[4] “Python tutorial for Beginners 1: Install and setup for Mac and windows,” YouTube, 17-May-2017. [Online]. Available: https://www.youtube.com/watch?v=YYXdXT2l-Gg&t=85s. [Accessed: 15-Oct-2023].

[5] Download ubuntu desktop | download | ubuntu. [Online]. Available: https://ubuntu.com/download/desktop. [Accessed: 27-Oct-2023].