Assignment-1

Use VirtualBox to Create Multiple VMs, Connect These VMs, and Host One Microservice-Based Application.

-- Prof. Sumit Kalra

Submitted By: G24AI1009
Name: Nitin Awasthi

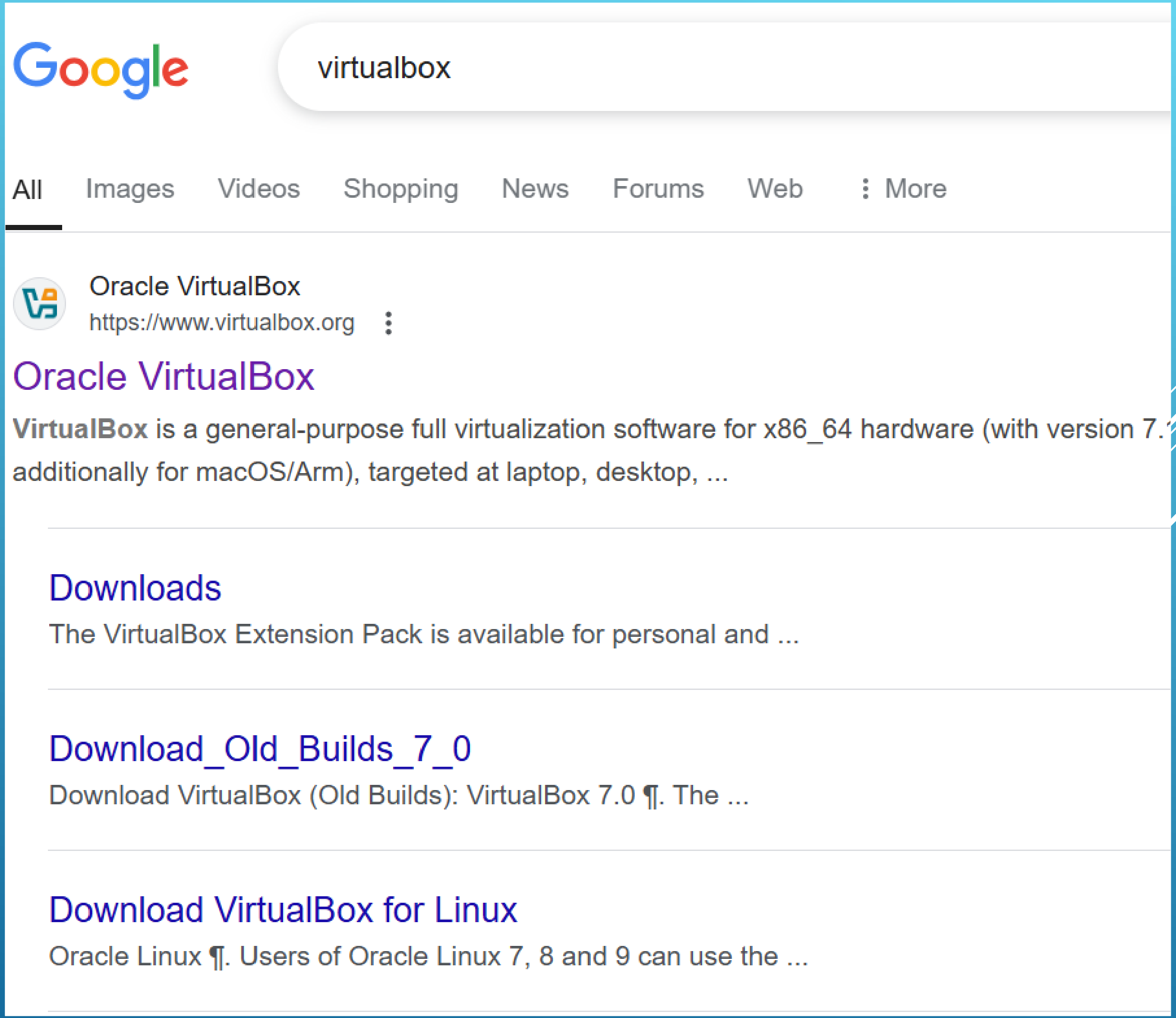
Index

1. Objective

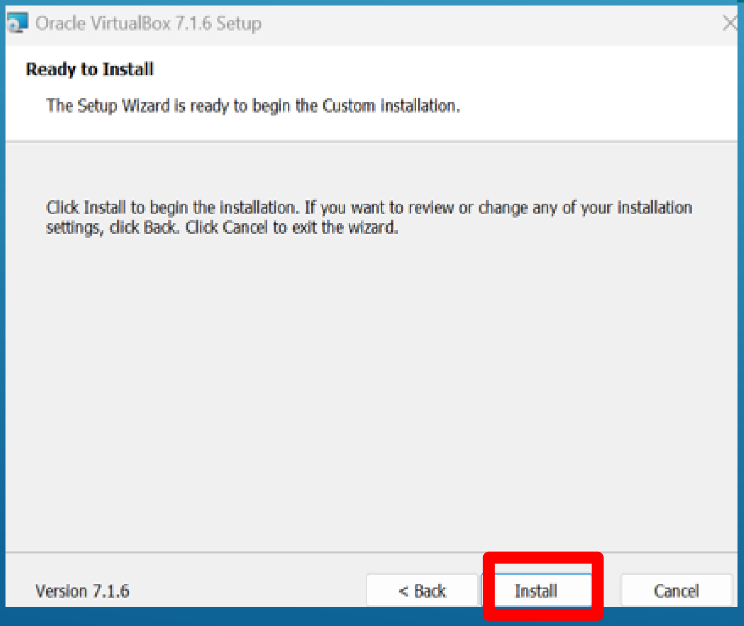
1. Create and configure multiple Virtual Machines (VMs) using VirtualBox,
2. Establish a network between them.
3. Deploy a microservice-based application across the connected VMs.

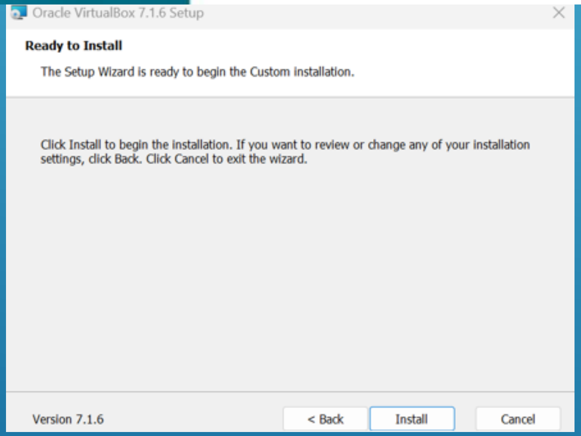
2. Installation of VirtualBox – 1/2

1. Visit the official website: https://www.virtualbox.org/
2. Click on the **"Downloads"** section.
3. Choose the appropriate version for your operating system (Windows, macOS, or Linux).
4. Run the downloaded installer file (.exe for Windows, .dmg for macOS, .deb or .rpm for Linux)
5. Follow the on-screen instructions in the setup wizard.
6. Accept the default settings unless you need custom configurations.
7. Various steps involved in the installation are shown in the next slide.



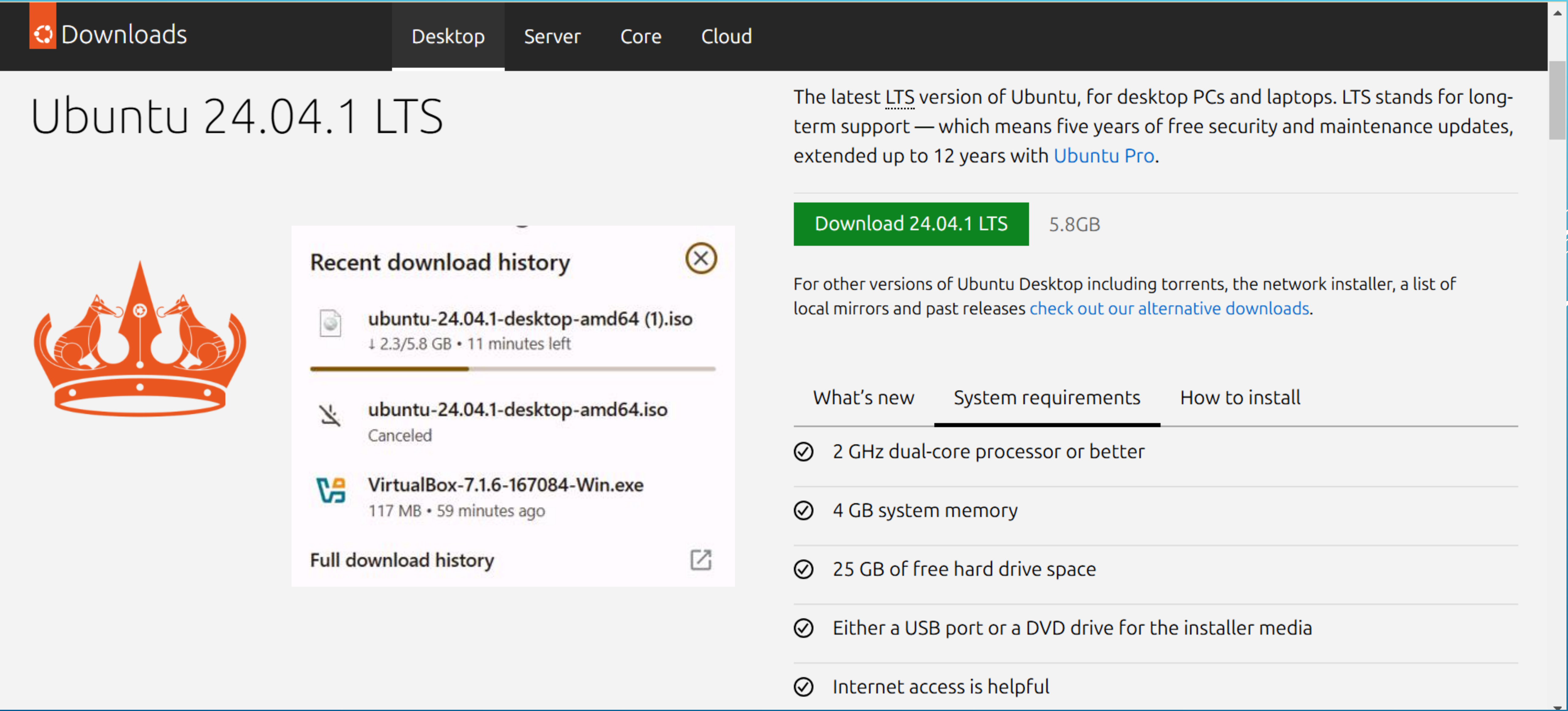
2.1 Various Steps Involved During Installation





3.Downloading Ubuntu 24.04.1 LTS File For Installation

The Ubuntu 24.04.1 LTS ISO file is needed to install and boot the OS inside the VM.



4. Creation of Virtual Machine-1

This page is the **welcome screen of Oracle VirtualBox Manager**, which provides an introduction and guidance for first-time users. It includes the following key elements:

1.Introduction to VirtualBox

1. Explains that VirtualBox is a tool for managing virtual machines.
2. Mentions that users can **import, add, and create new VMs** using toolbar buttons.

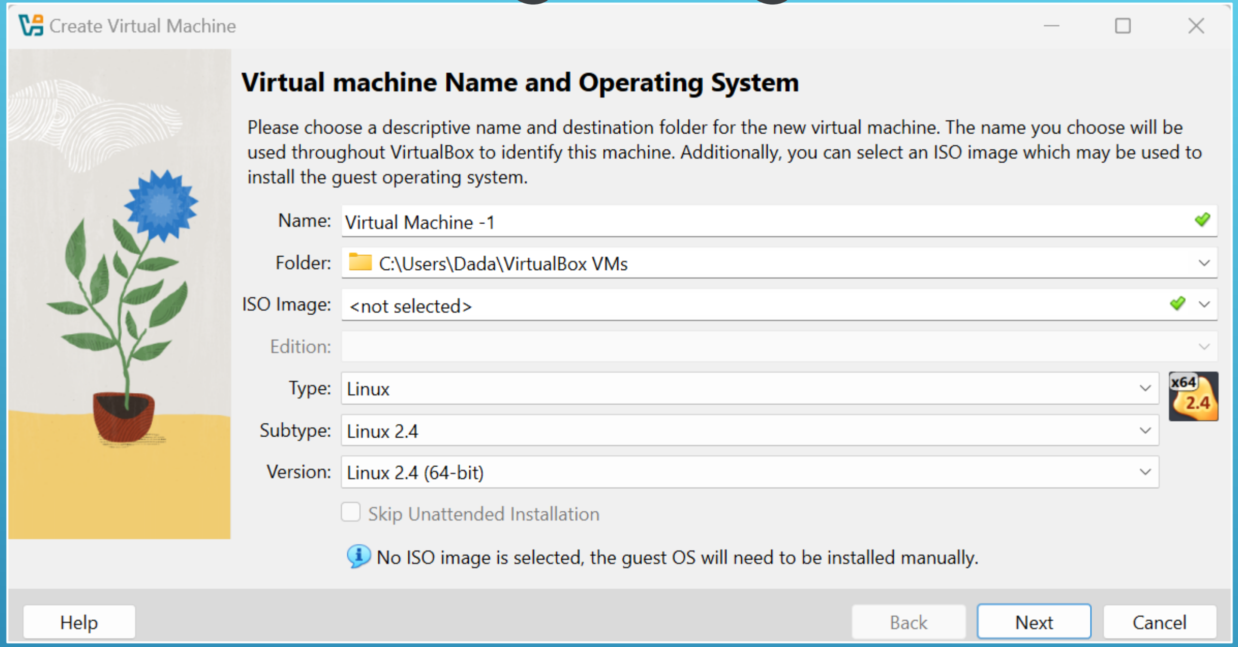
2.Experience Mode Selection

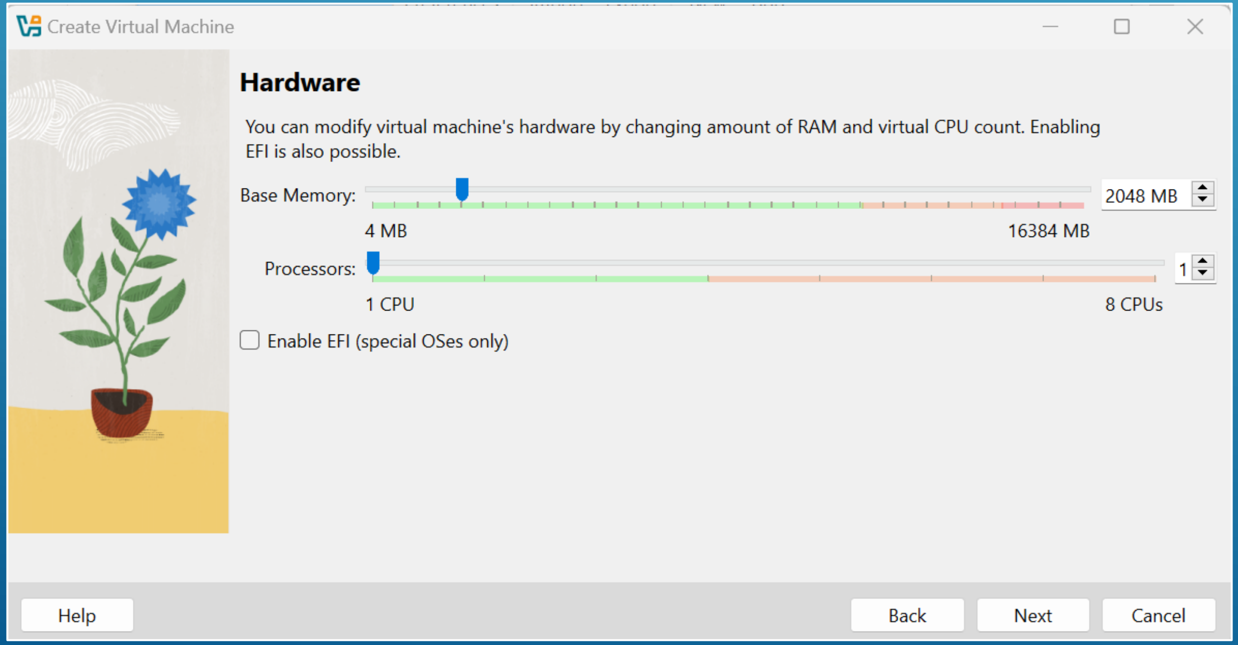
1. Offers two modes for users:
    1. **Basic Mode**: A simplified interface with fewer options.
    2. **Expert Mode**: A more advanced interface with full functionality.

3.Toolbar Options

1. Provides options like **Preferences, Import, Export, New, and Add** for VM management.

4.1 Configuring Virtual Machine – 1/3

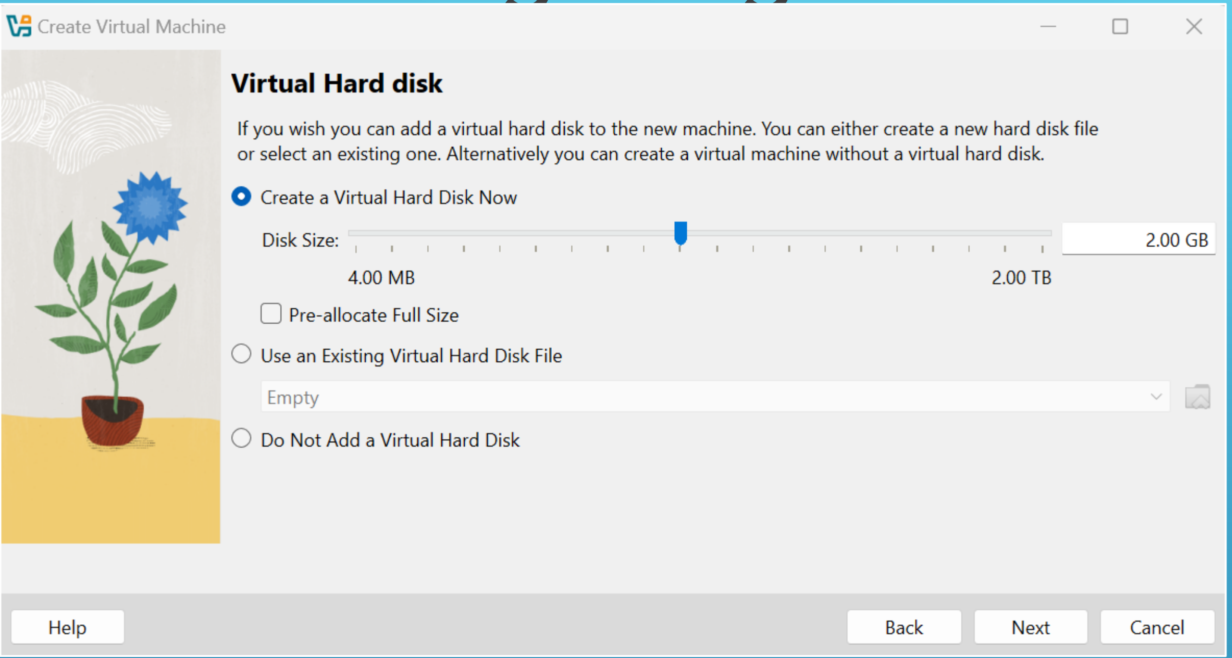


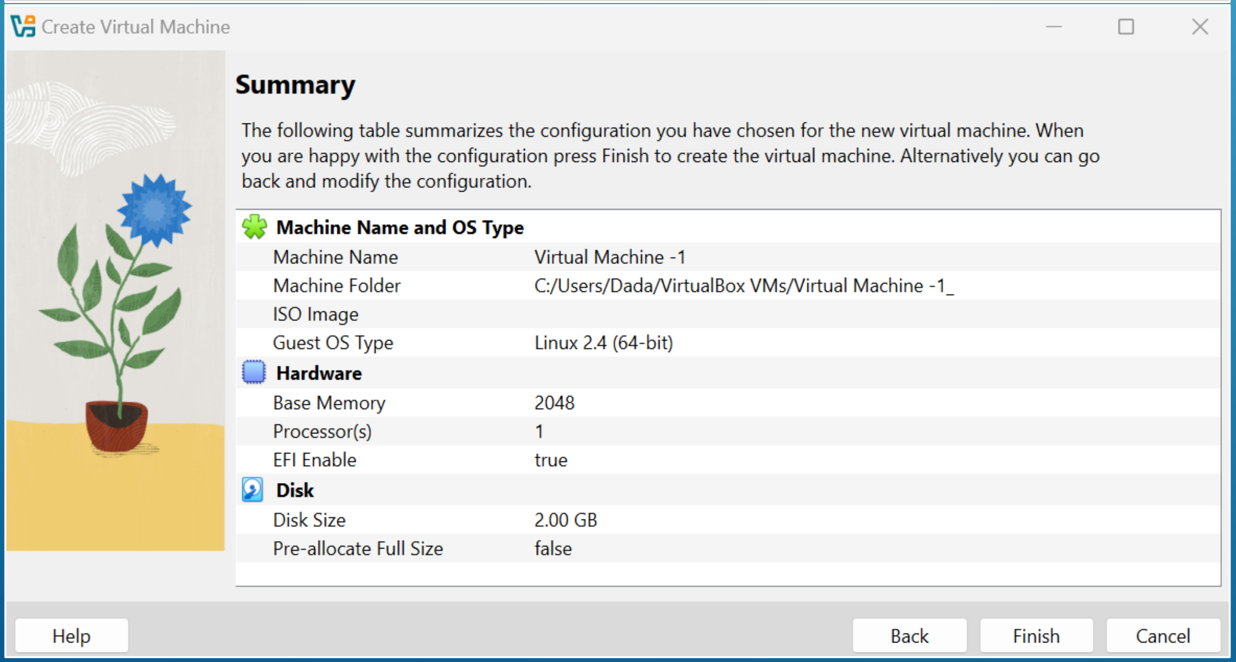


Steps to Create a Virtual Machine in VirtualBox (Based on Image)

1. Enter VM Name – Provide a name (e.g., "Virtual Machine-1").
2. Select default or dedicated folder as save location.
3. Choose an ISO image of Ubuntu 24.04.1 LTS
4. Select OS Type – Choose Linux as the Type.
5. Choose OS Version – Select Ubuntu 64-bit (or the appropriate Linux version).
6. Click "Next" – Proceed to the next configuration step.
7. Select Base Memory "4096 MB" for Ubuntu OS.
8. Consider Default Processor settings in my case.

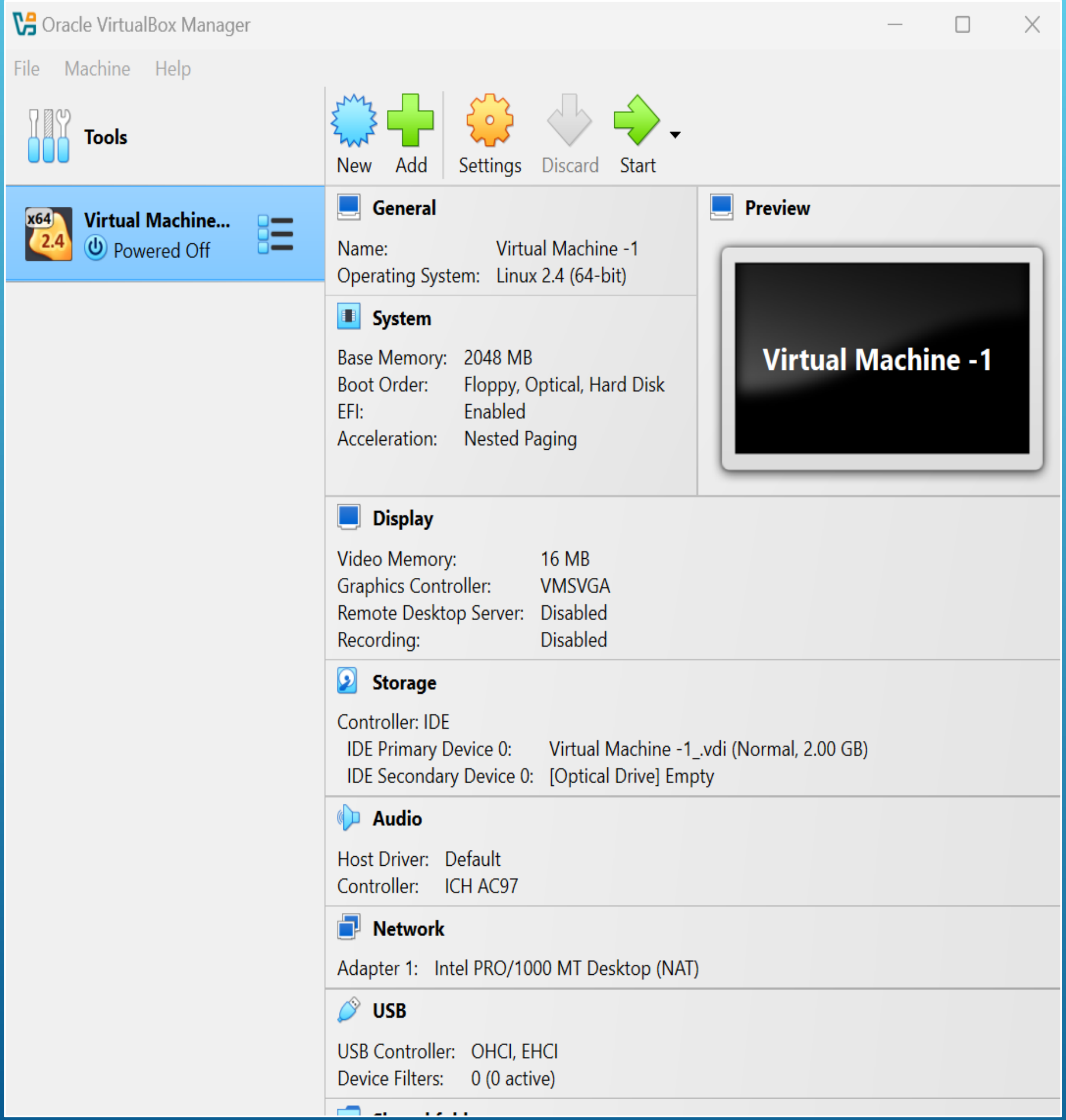
4.2 Configuring Virtual Machine – 2/3





1. Select "Create a Virtual Hard Disk Now" – This will create a new virtual disk for the VM.
2. Set Disk Size – Adjust the disk size (e.g., 25GB recommended, but at least 2GB as shown in the image).
3. Choose Pre-allocation (Optional) – Tick "Pre-allocate Full Size" for better performance or leave it unchecked for dynamic allocation.
4. Click "Next" – Confirm the disk settings and proceed.
5. Complete VM Creation – Click "Finish" to finalize the Virtual Machine setup

4.3 Configuring Virtual Machine – 3/3

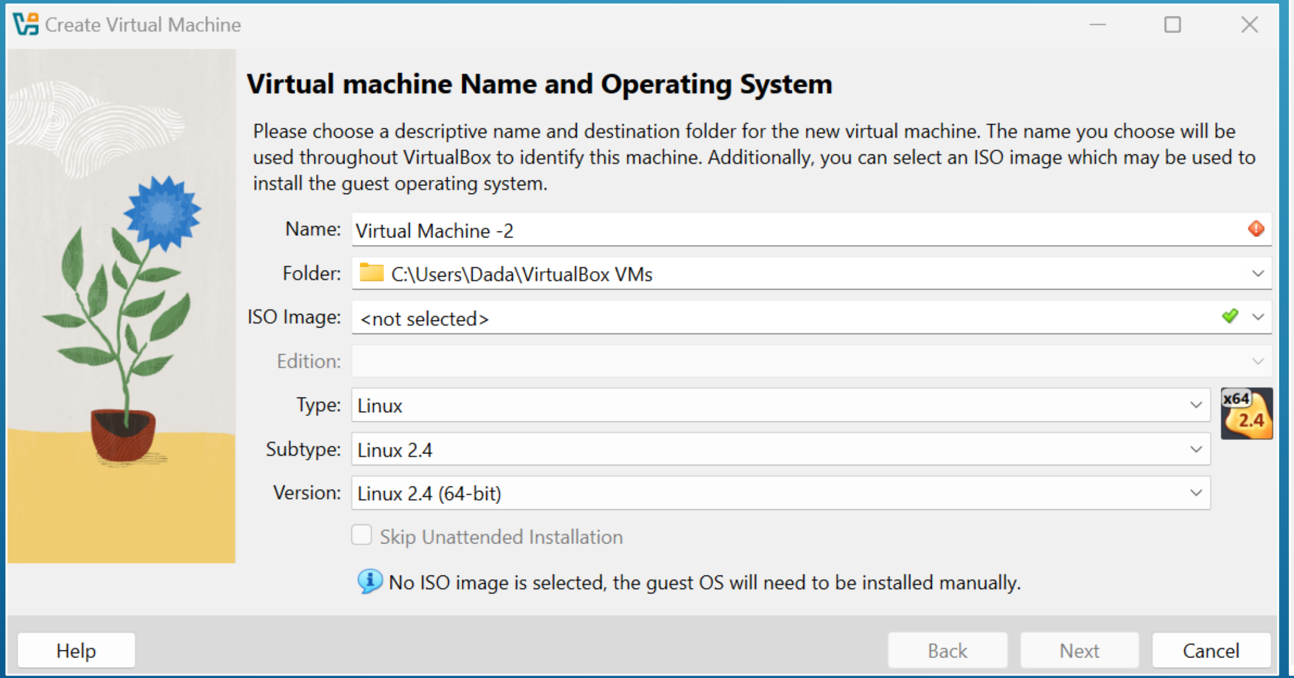


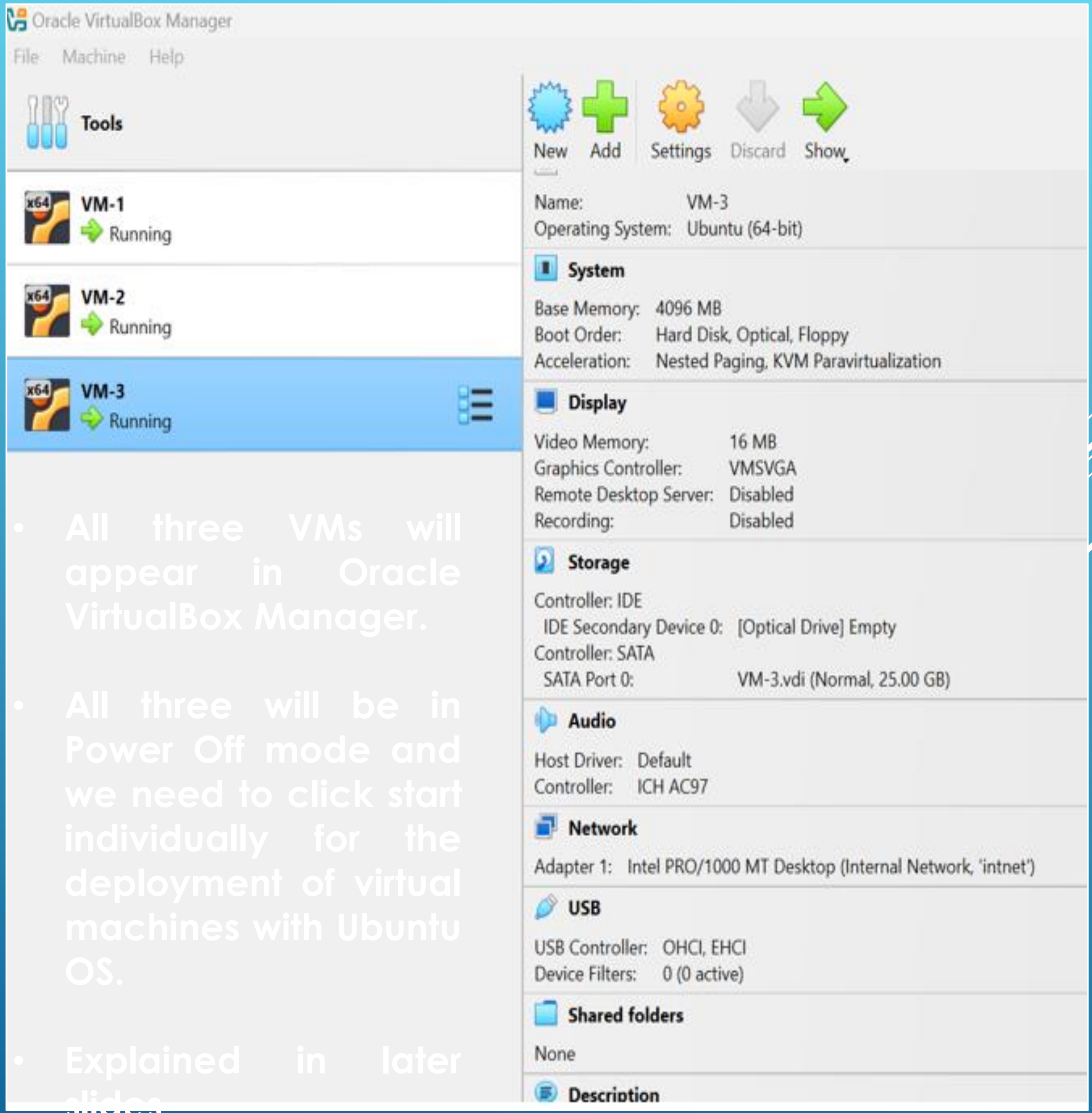
This page is the **VirtualBox Manager interface**, showing the details of a created but currently powered-off virtual machine (**Virtual Machine -1**). It provides an overview of the VM's configuration, including:

- **VM Name:** Virtual Machine -1
- **OS Type:** Linux 2.4 (64-bit)
- **Memory Allocation:** 2048 MB (2GB) RAM
- **Boot Order:** Floppy, Optical, Hard Disk
- **Storage:** 2GB Virtual Disk (VDI) with an empty optical drive
- **Display:** 16MB Video Memory, VMSVGA Graphics Controller
- **Network:** Intel PRO/1000 MT Desktop (NAT mode)
- **Audio:** ICH AC97 Controller (Default)
- **USB Support:** OHCI, EHCI (0 active filters)
- **Available Actions:** New, Add, Settings, Discard, Start

4.4 Configuring Virtual Machine – VM2 and VM3

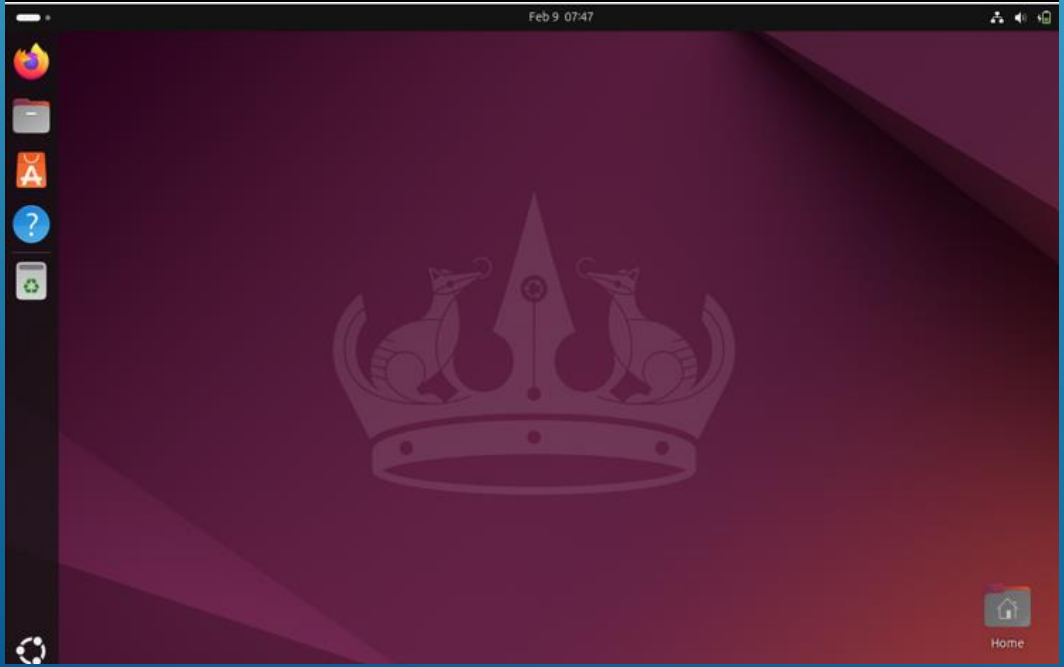
1. Deploying 2 another Virtual Machines – 2 and 3 using VirtualBox.
2. Using same Ubuntu 24.04.1 LTS ISO image file for all three VMs for deploying OS.
3. Configure RAM (4GB or more), Storage (25GB or more), and Ubuntu (64-bit) settings.





- All three VMs will appear in Oracle VirtualBox Manager.
- All three will be in Power Off mode and we need to click start individually for the deployment of virtual machines with Ubuntu OS.
- Explained in later slides

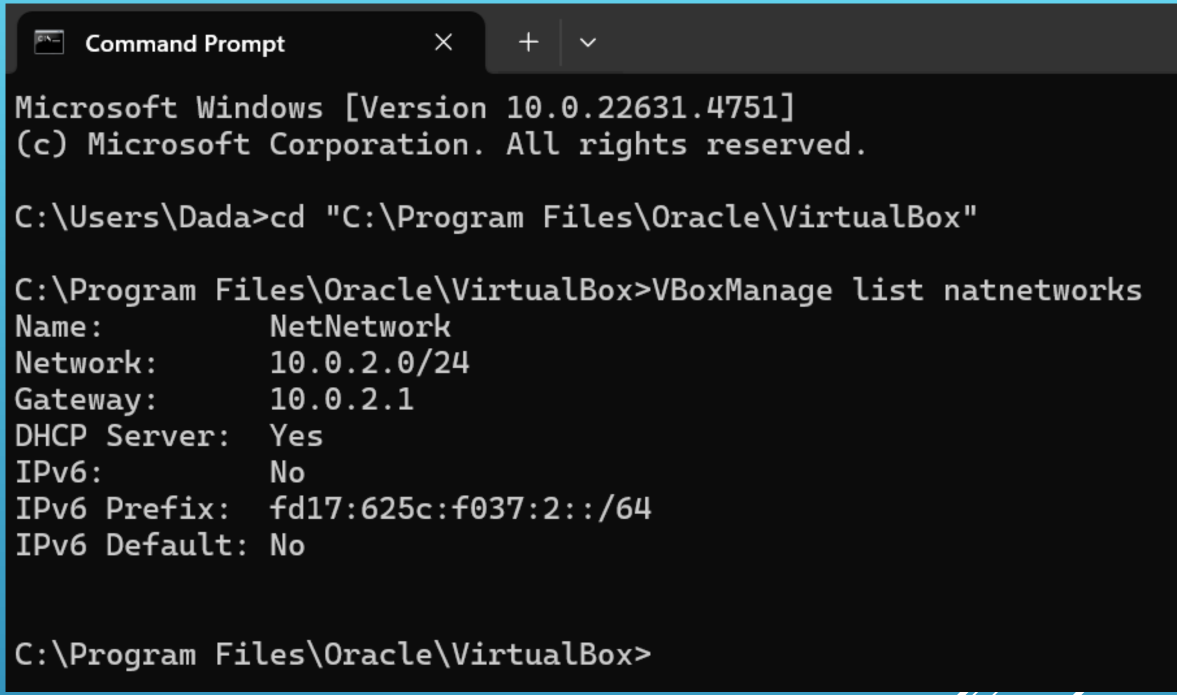
5. Starting Deployment of Virtual Machines with



1. **Power On VM** – Boot process starts.
2. **GRUB Loads** – Ubuntu kernel initializes.
3. **System Services Start** – Essential processes run.
4. **Login Screen Appears** – GUI or terminal ready.
5. **Pre-installed Apps Available** – Default Ubuntu tools ready.
6. **Network Detected** – Interface activation begins.
7. **IP Assigned** – DHCP (dynamic) or static IP applied.
8. **Internet Access Checked** – Connection established if enabled.
9. **User Verifies Connectivity** – ping or curl test.
10. **System Ready** – Fully operational.

Note: Same steps will be followed for all three VMs

6. Internal connectivity of VMs through IPv4 -1/4



```
Command Prompt

Microsoft Windows [Version 10.0.22631.4751]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Dada>cd "C:\Program Files\Oracle\VirtualBox"

C:\Program Files\Oracle\VirtualBox>VBoxManage list natnetworks
Name:         NetNetwork
Network:      10.0.2.0/24
Gateway:      10.0.2.1
DHCP Server:  Yes
IPv6:         No
IPv6 Prefix:  fd17:625c:f037:2::/64
IPv6 Default: No


C:\Program Files\Oracle\VirtualBox>
```

Step-1: Navigate to VirtualBox installation folder

Command changes the current working directory to **VirtualBox installation folder** on a Windows machine.

cd "C:\Program Files\Oracle\VirtualBox“

Step-2: Create a NAT Network

Command creates a NAT Network named NatNetwork.

VBoxManage natnetwork add --netname NatNetwork --network "10.0.2.0/24" --enable --dhcp on

--network "10.0.2.0/24": Defines the subnet (IP range) for the network. This means:

- The network address is 10.0.2.0.
- The subnet mask is /24 (255.255.255.0), allowing 254 usable IPs.

--enable: Enables the NAT network.

dhcp on: Enables DHCP, meaning that any VM attached to this network will automatically receive an IP address from this range.

6. Internal connectivity of VMs through IPv4 -2/4

Step-3: Verify that the NAT Network was created

Below command Lists all configured NAT networks in VirtualBox. If NatNetwork appears in the list, it was successfully created.

VBoxManage list natnetworks

Step-4: Attach your VMs to the NAT Network

VBoxManage modifyvm "VM1" --nic1 natnetwork --nat-network1 "NatNetwork“
VBoxManage modifyvm "VM2" --nic1 natnetwork --nat-network1 "NatNetwork“
VBoxManage modifyvm "VM3" --nic1 natnetwork --nat-network1 "NatNetwork"

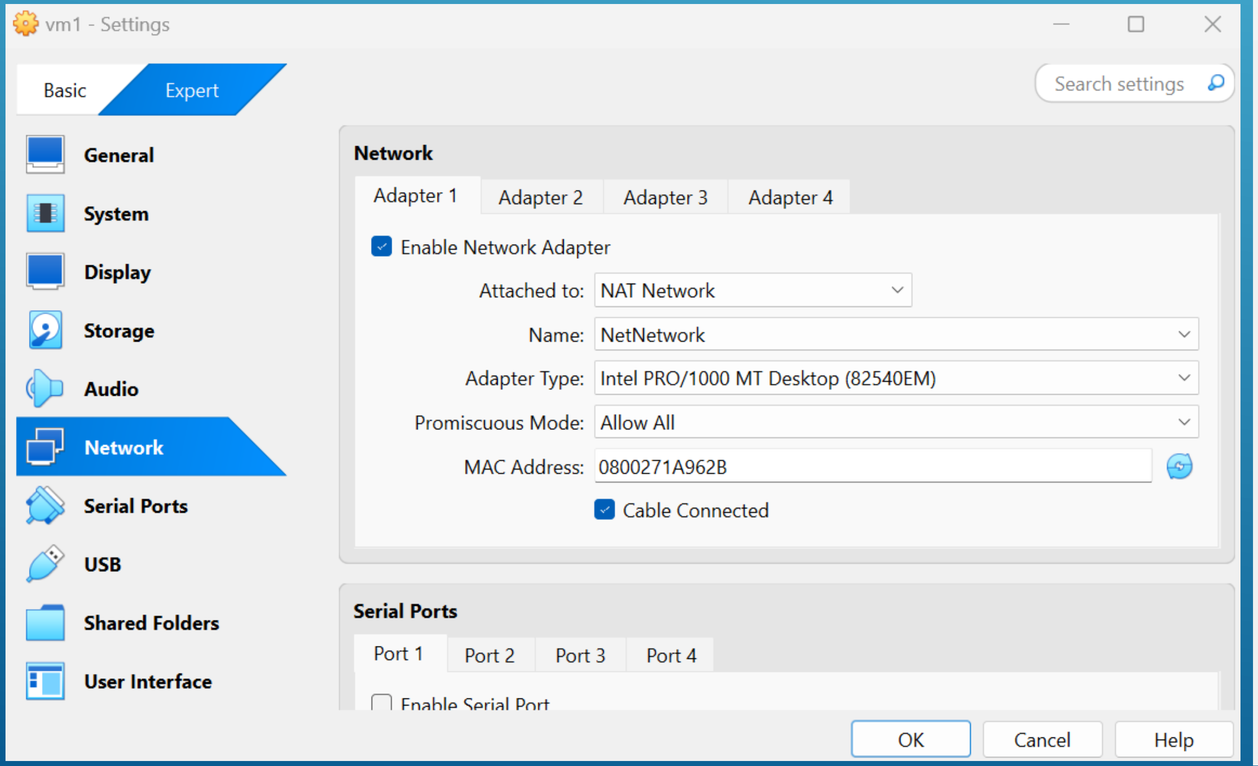
VBoxManage modifyvm "VM_Name": Modifies the settings of the VM named VM_Name.
--nic1 natnetwork: Sets Network Adapter 1 (NIC1) to use a NAT Network.
--nat-network1 "NatNetwork": Specifies that the VM should connect to the NatNetwork created earlier.

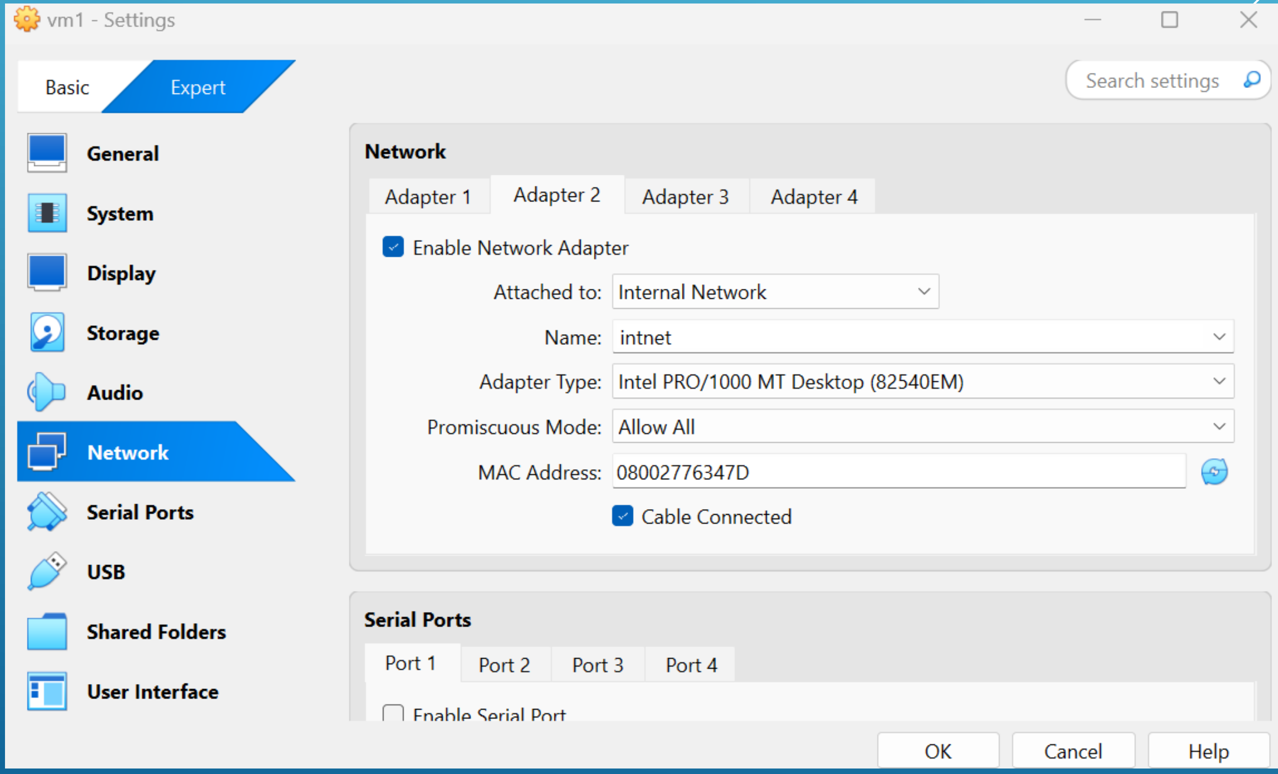
6. Internal connectivity of VMs through IPv4 -3/4

Created two different Network Adapters to allows VMs to communicate while remaining isolated from the host. The Internal Network (intnet) enables secure VM-to-VM communication without internet or host access, useful for private networking.

Adapter-1: VM1 is connected to a NAT Network (NetNetwork), allowing communication with other VMs in the same network while enabling outbound internet access.

Adapter-2: VM1 is connected to an Internal Network (intnet), allowing communication only with other VMs attached to the same internal network without internet access.

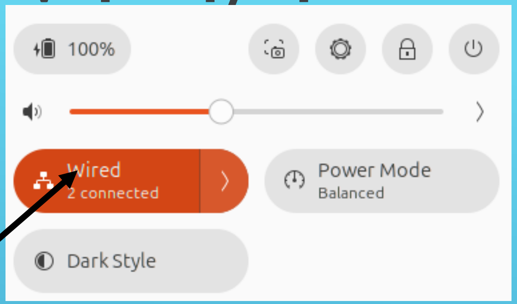


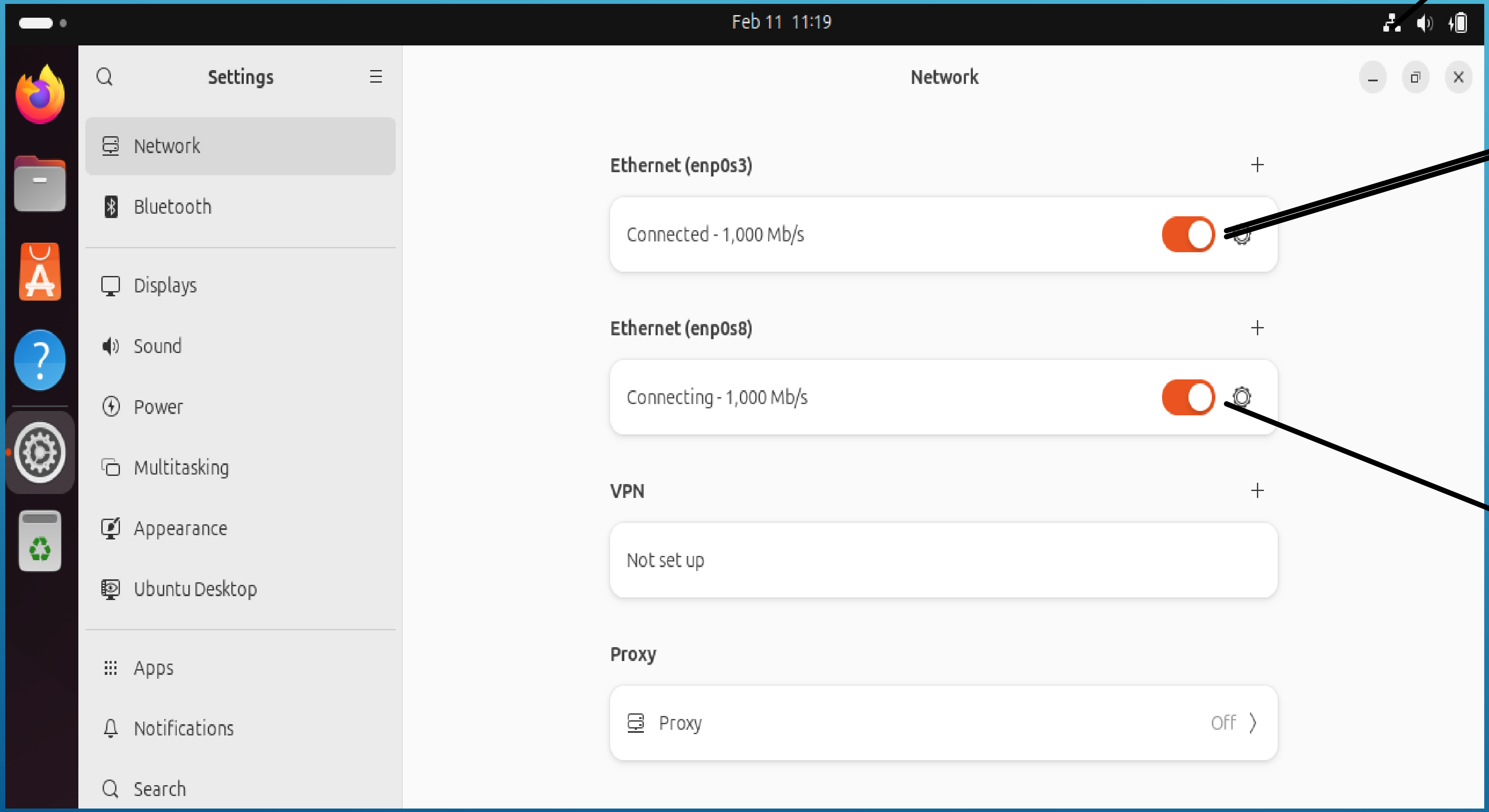


6. Internal connectivity of VMs through IPv4 -4/4

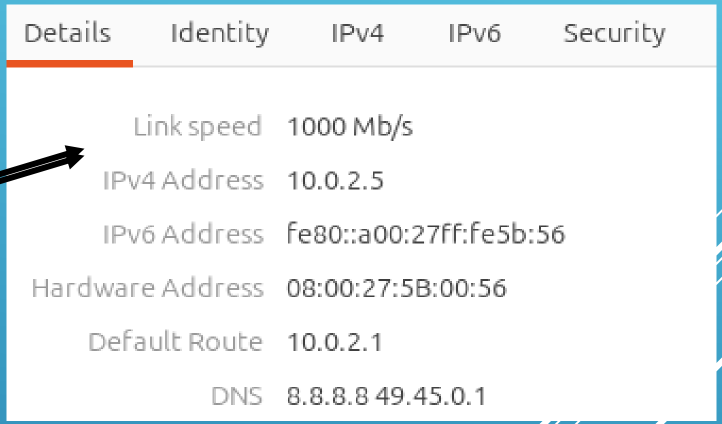
Here, it shows "2 connected", meaning two wired interfaces are currently active.

Network settings/details – Clicking it will show IP addresses, connection speed, and adapter configurations.

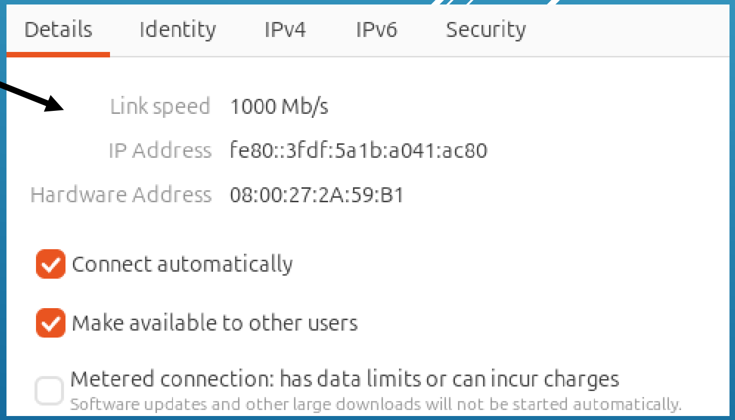




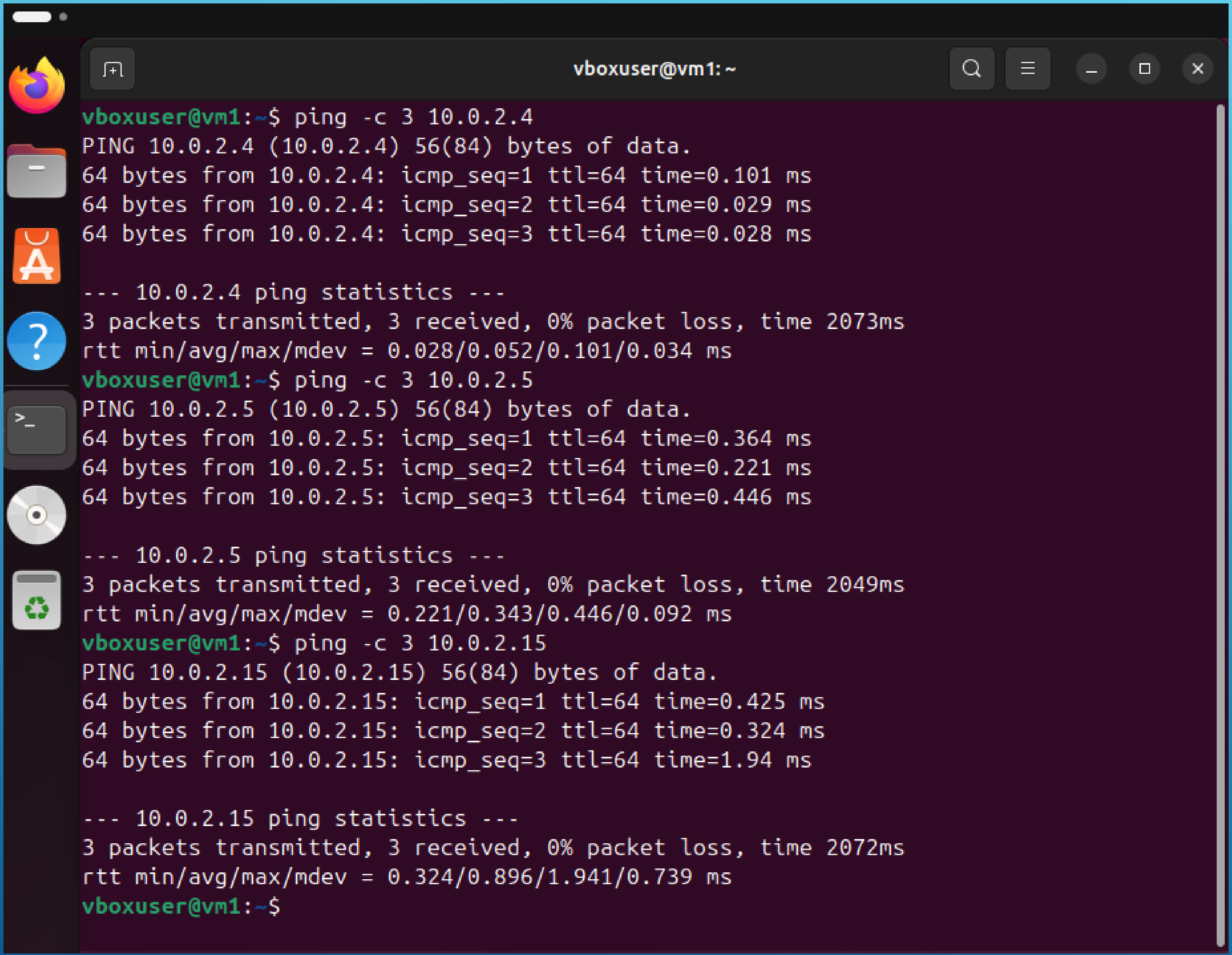
Adapter-1



Adapter-2

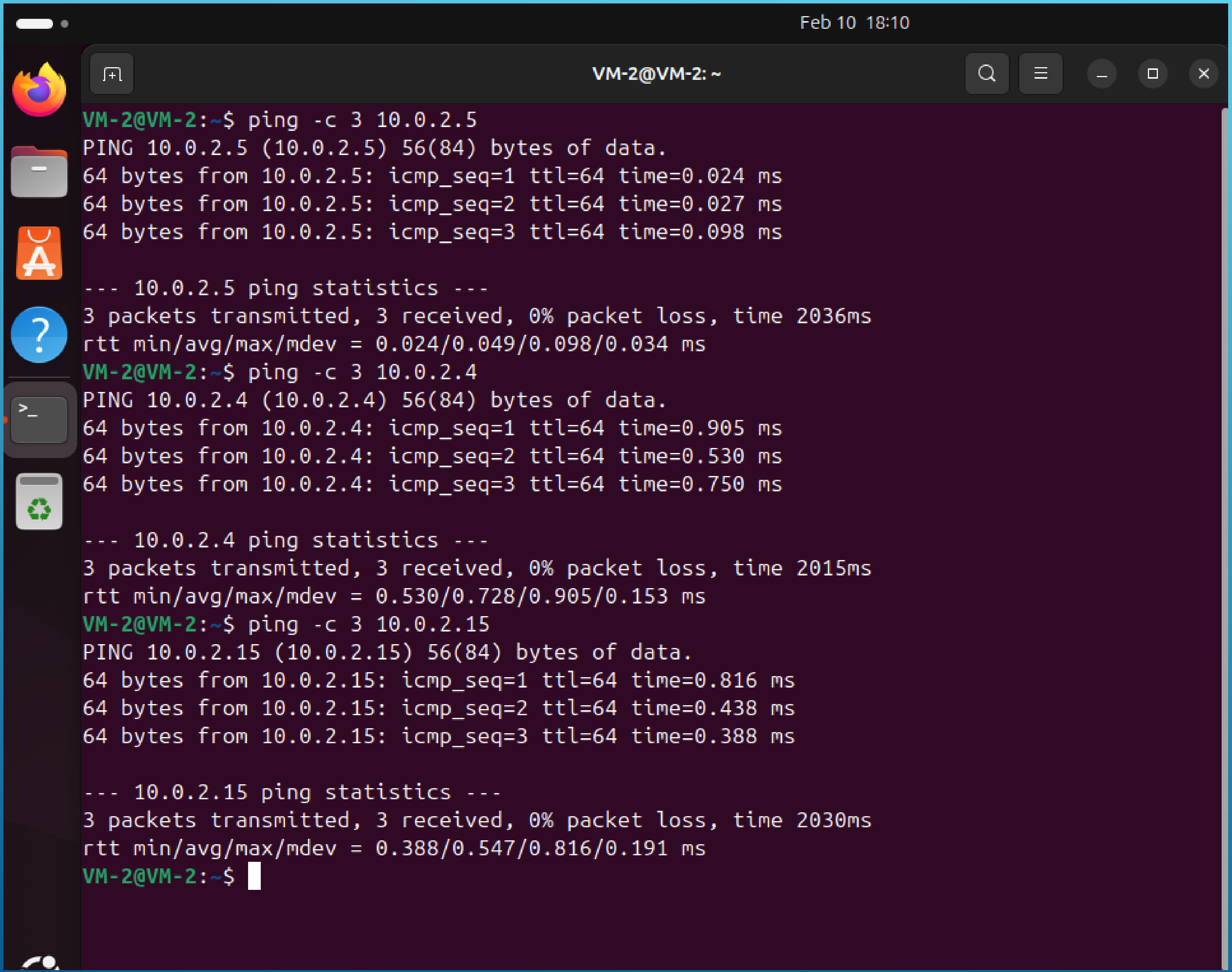


6.1 Ping Results – VM1(Self) → VM2 and VM3



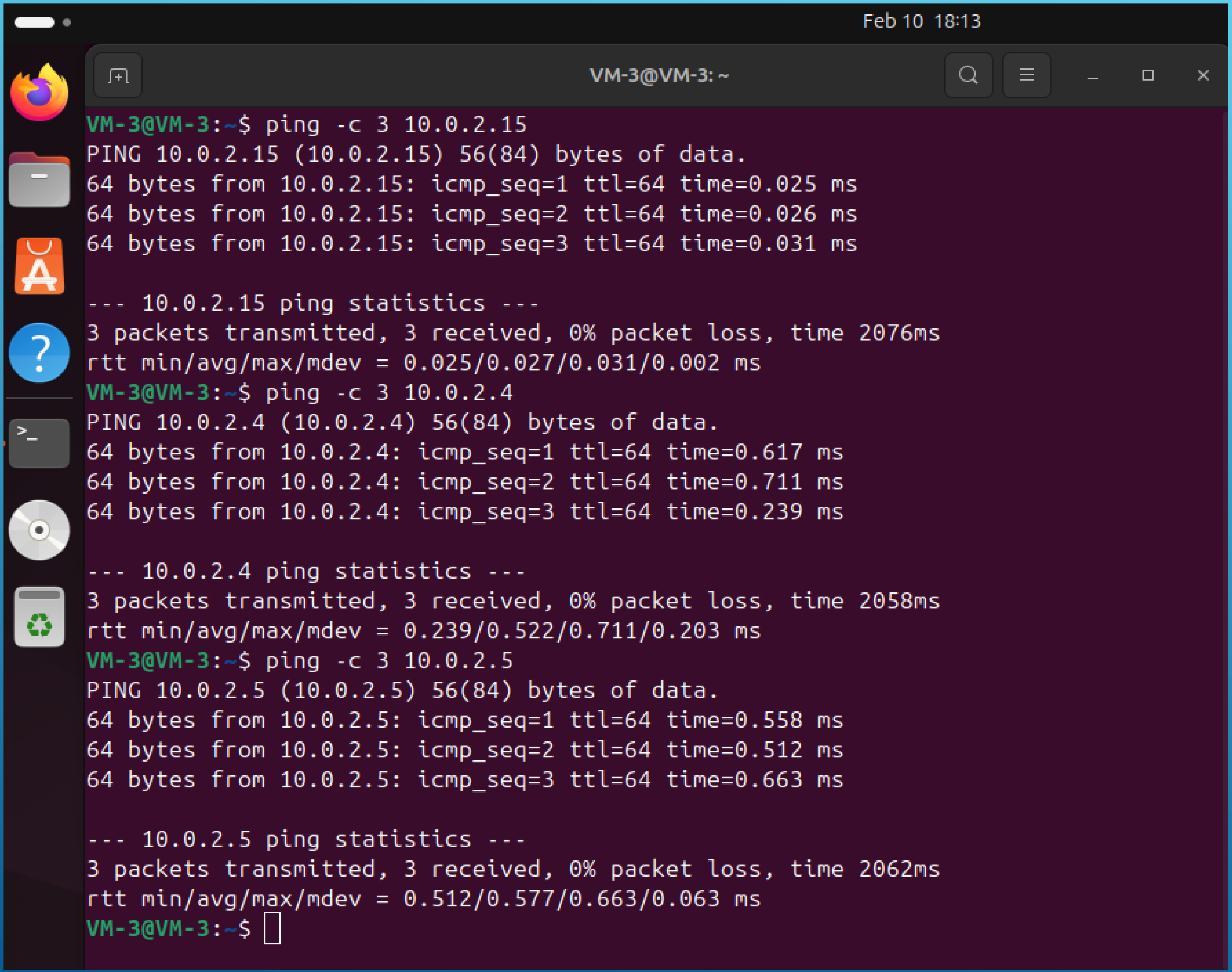
1. The screenshot shows successful ping tests from VM-1 (10.0.2.4) to three other VMs:
2. Ping to 10.0.2.4 (itself) – Successful, indicating the network interface is active.
3. Ping to 10.0.2.5 – Successful, confirming connectivity to VM-2.
4. Ping to 10.0.2.15 – Successful, confirming connectivity to VM-3.
5. All VMs are reachable, and the internal network is properly configured with no packet loss.
6. Latency values are low, showing a healthy local network setup.

6.2 Ping Results – VM2(Self) → VM1 and VM3



1. The screenshot shows successful ping tests from VM-2 (10.0.2.5) to three other VMs:
2. Ping to 10.0.2.5 (itself) – Successful, indicating the network interface is active.
3. Ping to 10.0.2.4 – Successful, confirming connectivity to VM-1.
4. Ping to 10.0.2.15 – Successful, confirming connectivity to VM-3.
5. All VMs are reachable, and the internal network is properly configured with no packet loss.
6. Latency values are low, showing a healthy local network setup.

6.3 Ping Results – VM3(Self) → VM1 and VM3



1. The screenshot shows successful ping tests from VM-3 (10.0.2.15) to three other VMs:
2. Ping to 10.0.2.15 (itself) – Successful, indicating the network interface is active.
3. Ping to 10.0.2.4 – Successful, confirming connectivity to VM-1.
4. Ping to 10.0.2.5 – Successful, confirming connectivity to VM-2.
5. All VMs are reachable, and the internal network is properly configured with no packet loss.
6. Latency values are low, showing a healthy local network setup.

7. Flask-Based Microservice Deployment Across Three VMs

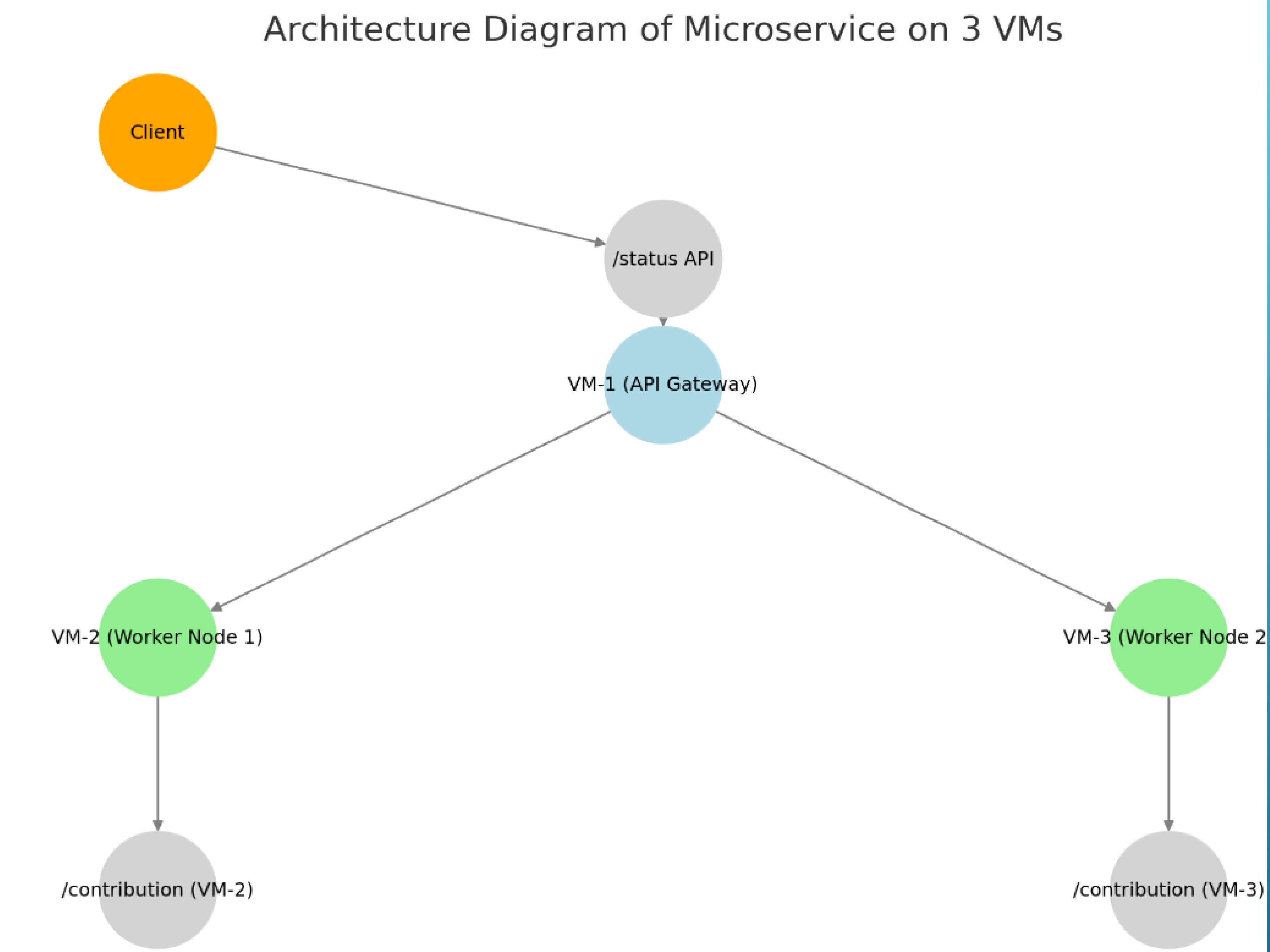
To test the connectivity and functionality of three VMs using a simple microservice,

We will deploy a simple **Flask-based microservice** across three interconnected Ubuntu VMs:

1. **VM-1 (10.0.2.4) → API Gateway** (Main service that aggregates responses from worker VMs)
2. **VM-2 (10.0.2.5) → Worker Node 1** (Provides a data processing service)
3. **VM-3 (10.0.2.15) → Worker Node 2** (Provides a log management service)

Note: After installing three VMs, my laptop became extremely slow and started hanging continuously. Due to this, I decided to install a **small RESTful APIs based microservice** that fulfills the objective of my assignment while also enabling me to understand this topic effectively.

7.1 Architecture Diagram of Microservice on 3 VMs



- Client sends a request to the API Gateway (VM-1) via the /status API.
- VM-1 (API Gateway) forwards requests to the worker nodes.

i. VM-2 (Worker Node 1) handles data processing and responds via /contribution.

ii. VM-3 (Worker Node 2) manages log management and responds via /contribution.

- VM-1 collects responses from VM-2 and VM-3, aggregates them, and returns the combined response to the client.

This setup ensures modularity, fault tolerance, and scalability.

7.2 API Gateway (VM-1 - 10.0.2.4)

The API Gateway is the central point that:

I. Collects responses from Worker Node 1 (VM-2) and Worker Node 2 (VM-3).

II. Aggregates the responses and returns a combined result to the client.

III. Handles failures (e.g., if a worker node is down, it still returns partial results).

<u>Example Workflow for API Gateway</u>

1. A user sends a request to VM-1 (10.0.2.4) on port 5000.
2. The gateway fetches data from:
   I. VM-2 (10.0.2.5:5001) → Data Processing Service
   II. VM-3 (10.0.2.15:5002) → Log Management Service
3. It combines the responses and sends back a unified response.

7.3 API Gateway (VM-1 - 10.0.2.4) – Cont.

Example Request:

curl http://10.0.2.4:5000/status

Example Response:


```
{
    "Main VM": "VM-1",
    "Worker Contributions": [
        {
            "VM": "VM-2",
            "Contribution": "Processing Sensor Data"
        },
        {
            "VM": "VM-3",
            "Contribution": "Managing System Logs"
        }
    ]
}
```

7.4 Worker Node 1 (VM-2 - 10.0.2.5)

This node acts as a **data processor**:

I. It performs **data analysis** (e.g., sensor data processing, real-time calculations).

II. It exposes an API endpoint (/contribution) to return its status.

Example Request to Worker Node 1:

curl http://10.0.2.5:5001/contribution

Example Response:

{ "VM": "VM-2", "Contribution": "Processing Sensor Data" }

This response is collected by the API Gateway and included in the final output.

7.5 Worker Node 2 (VM-3 - 10.0.2.15)

This node acts as a log manager:

I. It handles logging (e.g., system logs, debugging, monitoring).

II. It exposes an (/contribution) to return its status.

Example Request to Worker Node 2:

curl http://10.0.2.15:5002/contribution

Example Response:

{ "VM": "VM-3", "Contribution": "Managing System Logs" }

Like Worker Node 1, this response is sent to the API Gateway for aggregation.

8. How They Work Together

Scenario 1: Everything Works:

A request to VM-1 (10.0.2.4:5000/status) will return contributions from both workers.

Scenario 2: VM-2 (Worker Node 1) Fails:

The API Gateway will detect failure and return:


```
{
    "Main VM": "VM-1",
    "Worker Contributions":
    [
        {
            "VM": "VM-2",
            "Contribution": "Error connecting"
        },
        {
            "VM": "VM-3",
            "Contribution": "Managing System Logs“
        }
    ]
}
```


This setup ensures fault tolerance and scalability in a distributed microservice system

9. Step-by-Step Implementation – 1/6

Step : 1 Install Flask on All VMs

```
sudo apt update && sudo apt install python3-pip –y
pip3 install flask requests
```

Step : 2 Deploy Microservice on VM-2 (10.0.2.5)

On VM-2, create the file worker_vm2.py:

```
nano worker_vm2.py

from flask import Flask, jsonify

app = Flask(__name__)

@app.route('/contribution', methods=['GET'])
def contribution():
    return jsonify({"VM": "VM-2", "Contribution": "Data Processing"})

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5001)
```

Save and exit (CTRL+X, then Y, then Enter).

9. Step-by-Step Implementation – 2/6

Step 3: Run the Microservice by executing the following command:

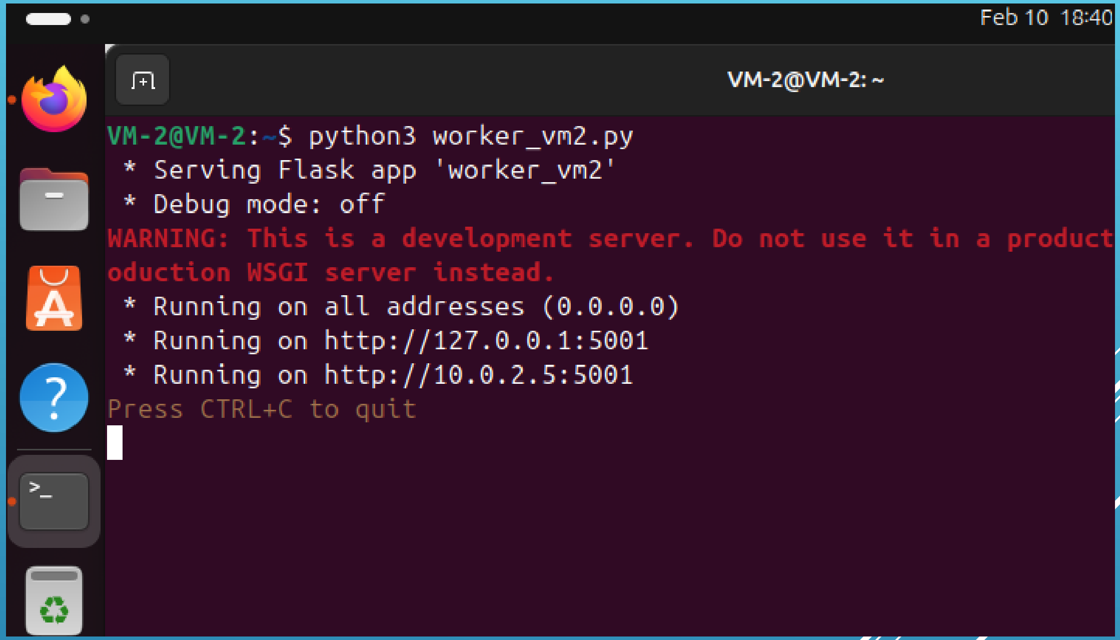
python3 worker_vm2.py

This starts the Flask server on **port 5001**, making it accessible inside the internal netwok.

Step 4: Test the Microservice

From **VM-1** or **VM-3**, test the microservice using:
curl http://10.0.2.5:5001/contribution

{"VM": "VM-2", "Contribution": "Data Processing"}



9. Step-by-Step Implementation – 3/6

Step 5: Create the Microservice File

On VM-3, create the file worker_vm3.py:

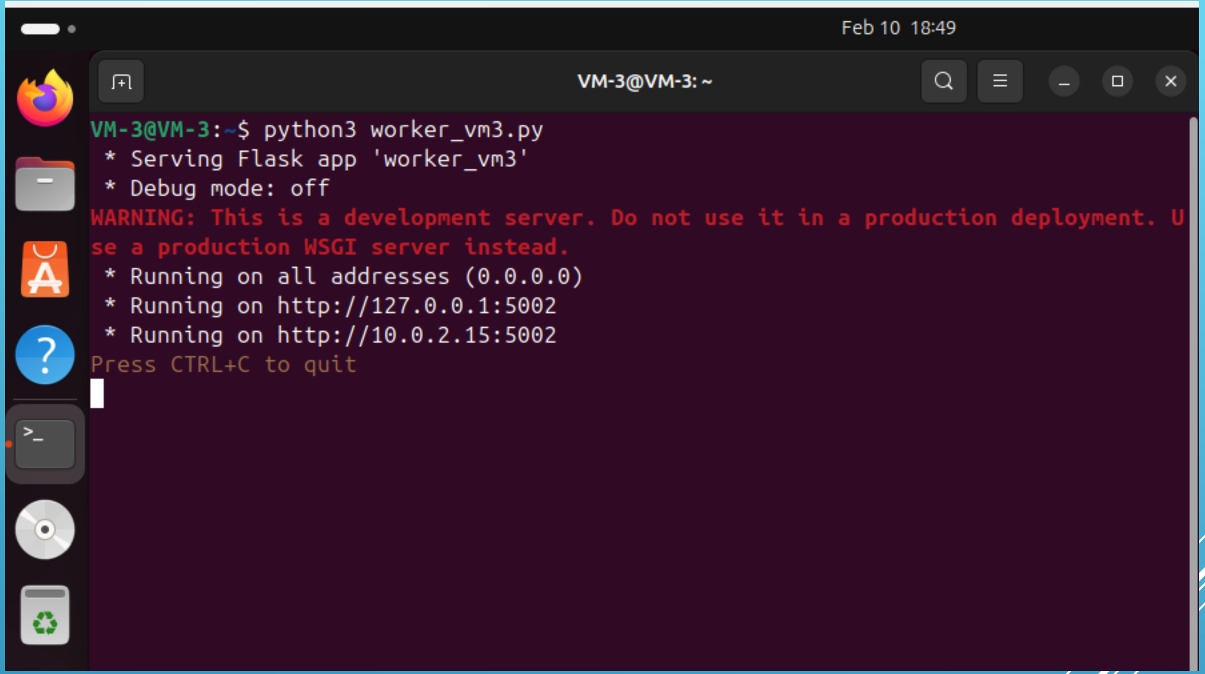
nano worker_vm3.py

```
from flask import Flask, jsonify

app = Flask(__name__)

@app.route('/contribution', methods=['GET'])
def contribution():
    return jsonify({"VM": "VM-3", "Contribution": "Log Management"})

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5002)
```

Save and exit (CTRL+X, then Y, then Enter).

Execute the following command to start the service:

python3 worker_vm3.py

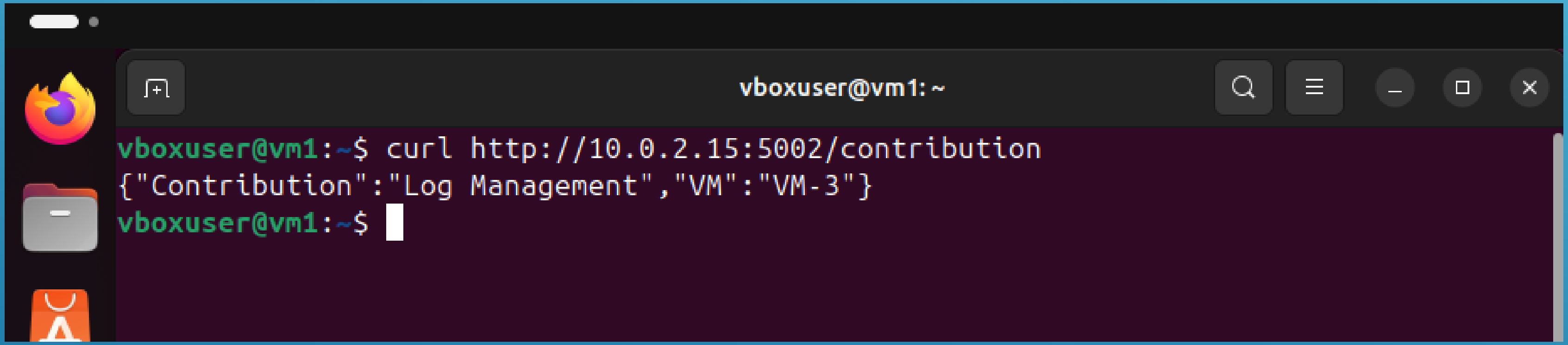


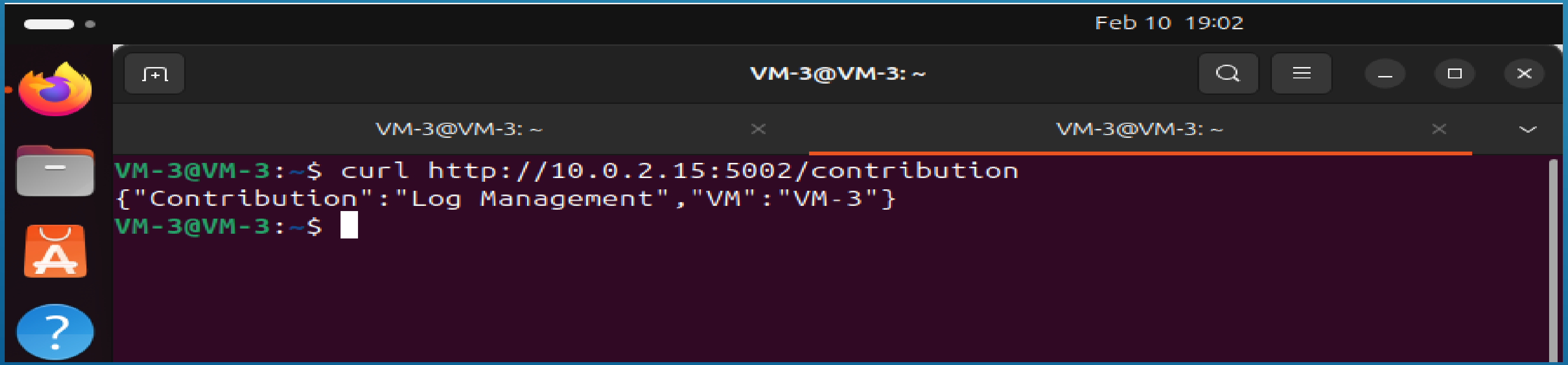
9. Step-by-Step Implementation – 4/6

Step 6: Test the Microservice

From **VM-1** or **VM-2**, run:
curl http://10.0.2.15:5002/contribution

{"VM": "VM-3", "Contribution": "Log Management"}





9. Step-by-Step Implementation – 5/6

Step 7: Deploy API Gateway on VM-1 (Aggregator)

File: main_api_gateway.py

```
from flask import Flask, jsonify
import requests

app = Flask(__name__)

WORKER_NODES = {
    "VM-2": "http://10.0.2.5:5001/contribution",
    "VM-3": "http://10.0.2.15:5002/contribution"
}

@app.route('/status', methods=['GET'])
def status():
    contributions = []
    for vm, url in WORKER_NODES.items():
        try:
            response = requests.get(url, timeout=2)
            if response.status_code == 200:
                contributions.append(response.json())
```

9. Step-by-Step Implementation – 6/6

```
 else:
                contributions.append({"VM": vm,
"Contribution": "Unavailable"})
        except requests.exceptions.RequestException:
            contributions.append({"VM": vm, "Contribution":
"Error connecting"})

    return jsonify({"Main VM": "VM-1", "Worker
Contributions": contributions})

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

Run: python3 main_api_gateway.py