Assignment: Virtualization and Cloud Computing

Student Details

Name: Aryan Baranwal
Roll Number: M25CSE035
Course: M.Tech (CSE)

Objective

Create and configure multiple Virtual Machines (VMs) using VirtualBox, establish networking between them, and deploy a microservice-based application using Node.js, MySQL, and ReactJS across connected VMs.

System Configuration

Host OS: macOS
Virtualization Tool: Oracle VirtualBox
Guest OS: Ubuntu 24.04.3 Desktop (amd64)

VM Architecture

VMUBUNTU-001: MySQL Database Server (192.168.56.2)
VMUBUNTU-002: Node.js Backend (Order Management API) (192.168.56.3)
VMUBUNTU-003: ReactJS Frontend (192.168.56.4)

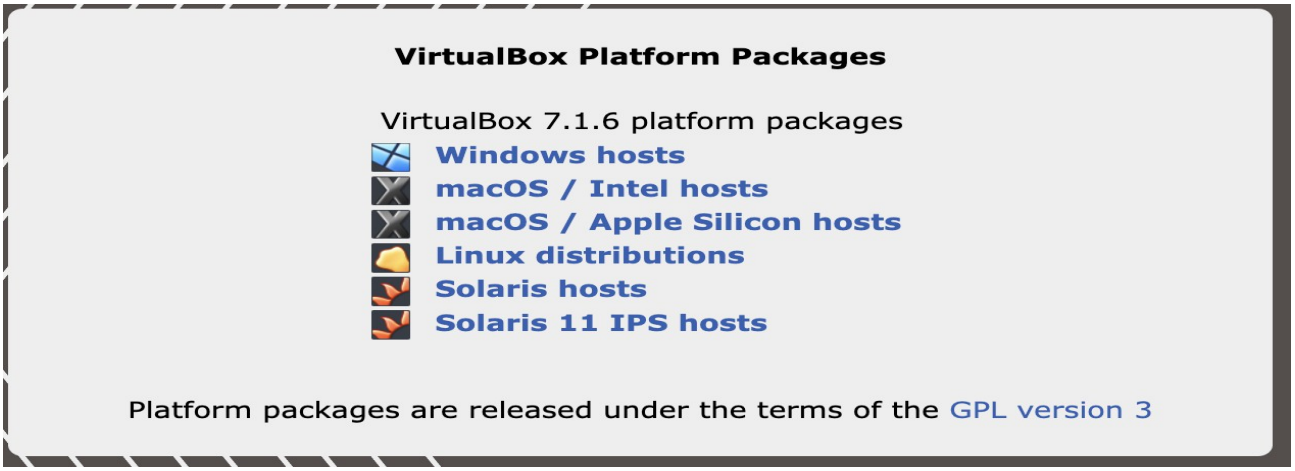
Network Configuration:
Adapter 1: NAT (Internet Access)
Adapter 2: Host-Only (VM-to-VM Communication)

Step-by-Step Instructions for Implementation:

Step 1: Installation of VirtualBox -

Download virtual box from below link - https://www.virtualbox.org/wiki/Downloads

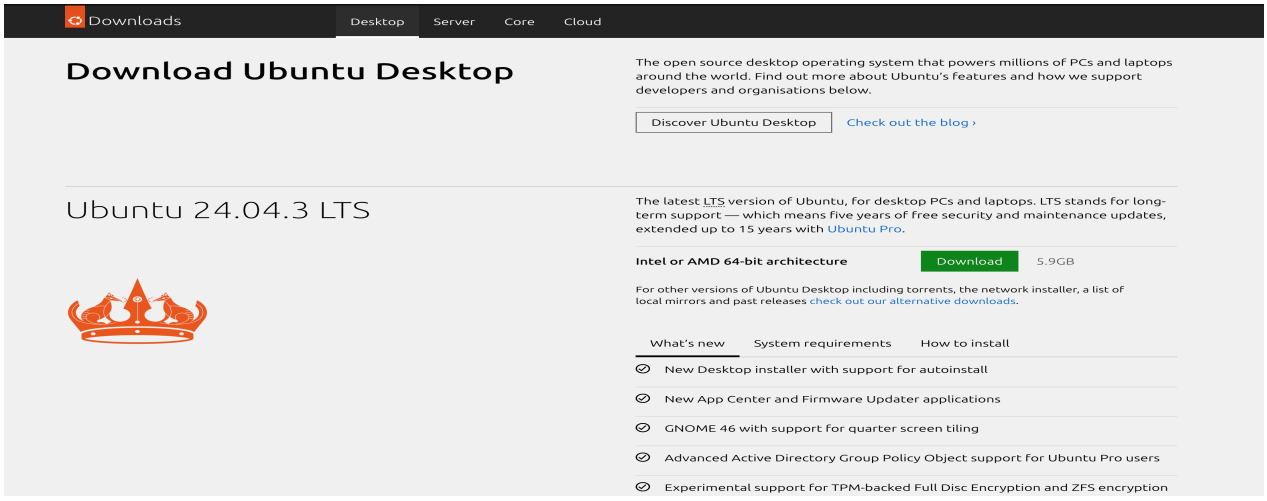
Refer the below screenshot -



Since I am using an Intel chip, I downloaded the **macOS / Intel hosts**. Now, using the above downloaded file, I completed the installation of Virtual box.

Step 2: Download the ISO Image –

For the virtual machine setup, I downloaded and used **Ubuntu 24.04.3 Desktop (AMD64)** with the ISO file ubuntu-24.04.3-desktop-amd64.iso. This ISO was obtained from the official Ubuntu website at https://ubuntu.com/download/desktop and was chosen due to its compatibility with the VM environment and stable support for development and deployment tasks.

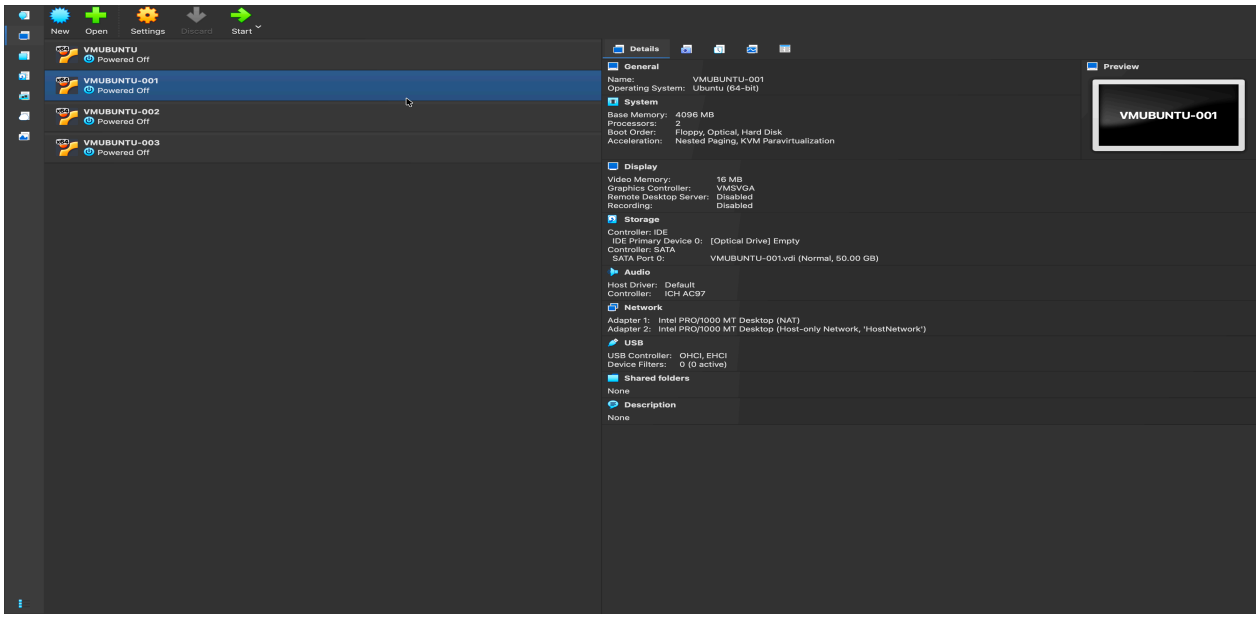


Step 3: Creation of Virtual Machines

After the successful installation of Virtual Box, I created multiple VM using the below configurations -

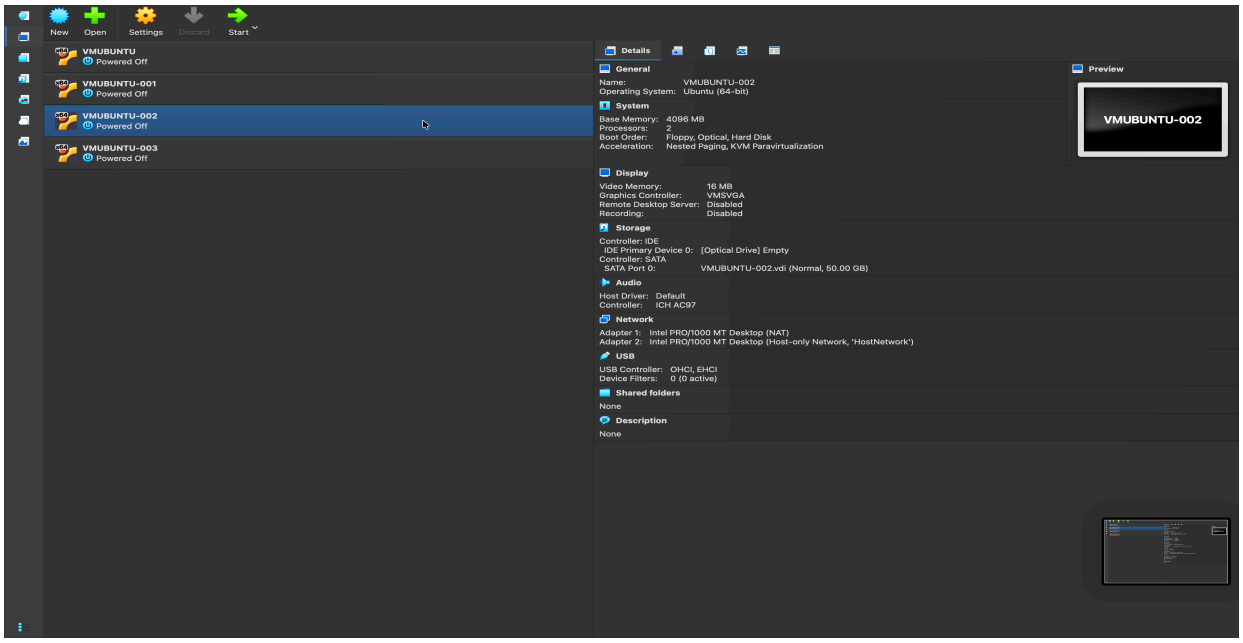
a. VMUBUNTU-001 :

Refer the below screenshot for configurations –



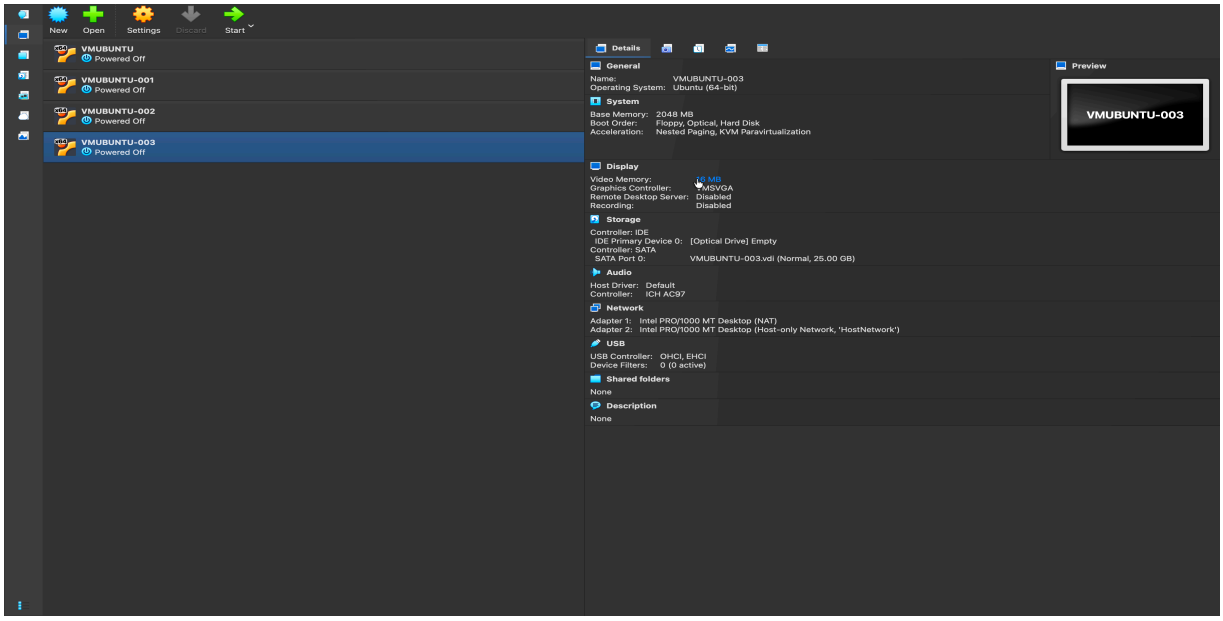
b. VMUBUNTU-002 :

Refer the below screenshot for configurations –



c. VMUBUNTU-003 :

Refer the below screenshot for configurations –

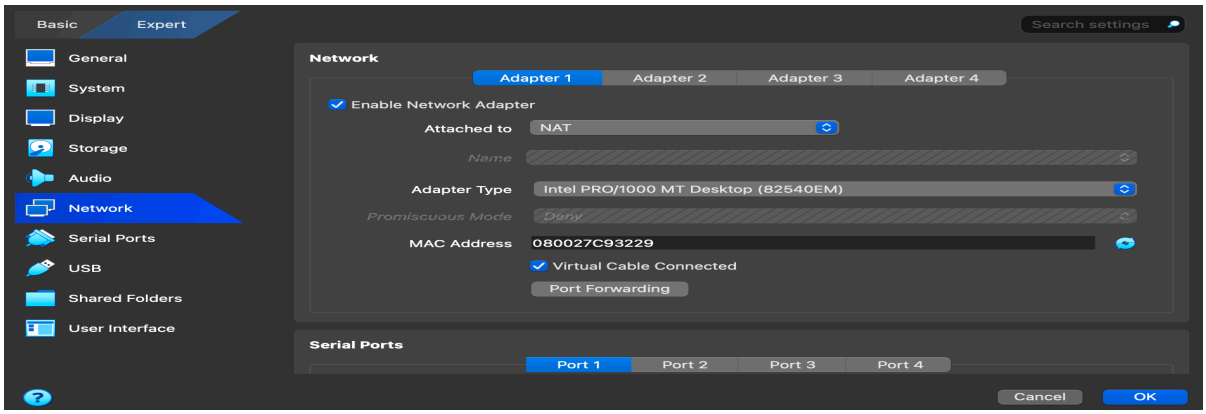


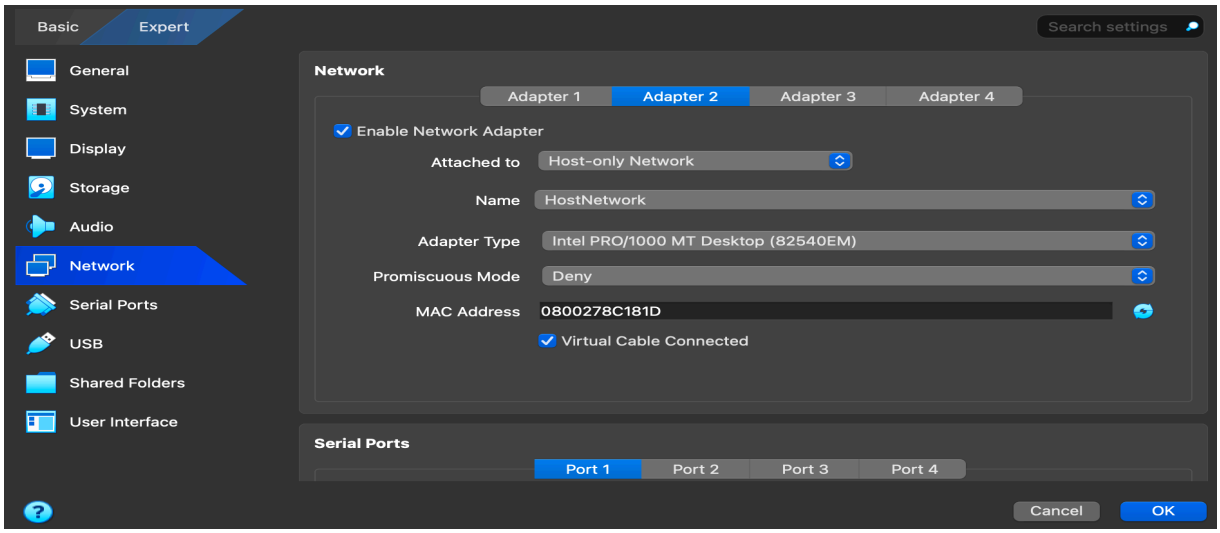
Step 4: Network Configuration

First, a network entry was created in **VirtualBox → Network** settings. After saving the configuration, the network settings were enabled for each virtual machine by navigating to **VM Settings → Network**.

- **Adapter 1** was configured as **NAT** to provide internet access to the virtual machine.
- **Adapter 2** was configured as **Host-Only Network** to allow communication between the host system and other VMs.

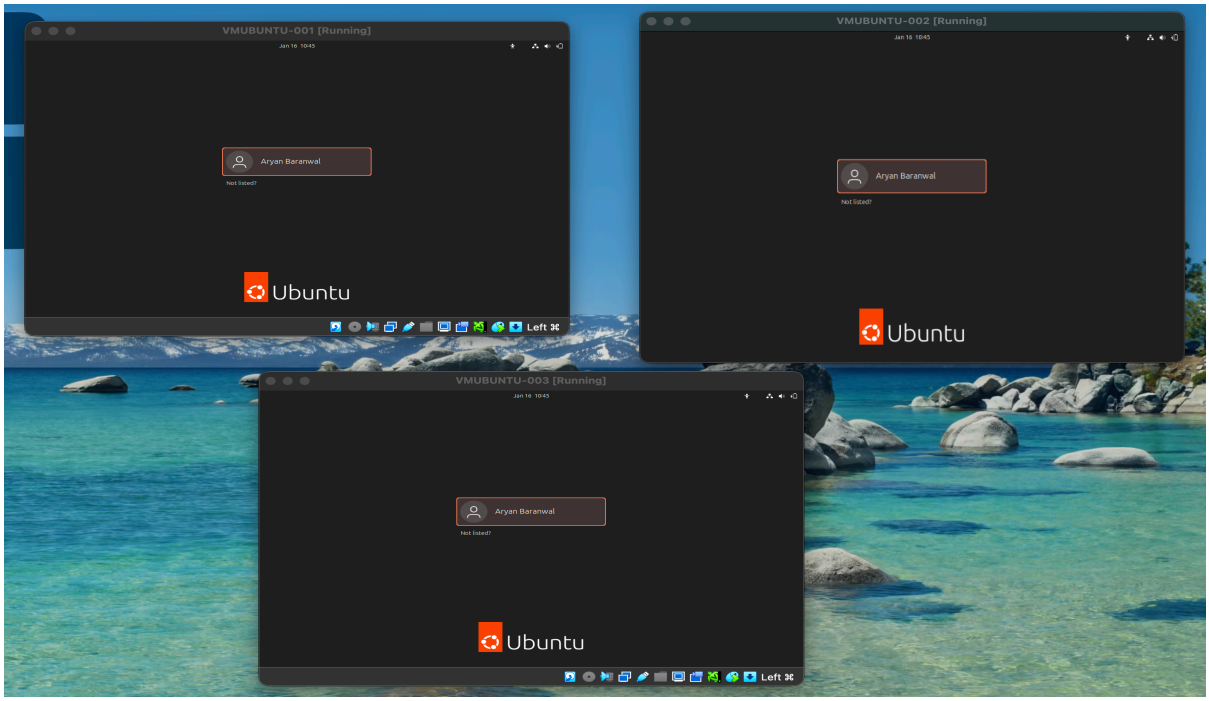
Both adapters were enabled for every VM. Static IP addresses were assigned on the Host-Only network, and connectivity between all virtual machines was verified using the ping command. The screenshots below are provided for reference for all network configurations.





Step 5: Start the VMs –

Start **all three virtual machines** and install the operating system on each of them. After the OS installation is completed, all three VMs should be running successfully, as shown in the screenshot below.



Next, verify network connectivity between the virtual machines. First, check the IP address of each VM using:

→ ip a

Then, test communication between all three virtual machines using the ping command:

- **From VMUBUNTU-001**

  ping 192.168.56.3

  ping 192.168.56.4

- **From VMUBUNTU-002**

  ping 192.168.56.2

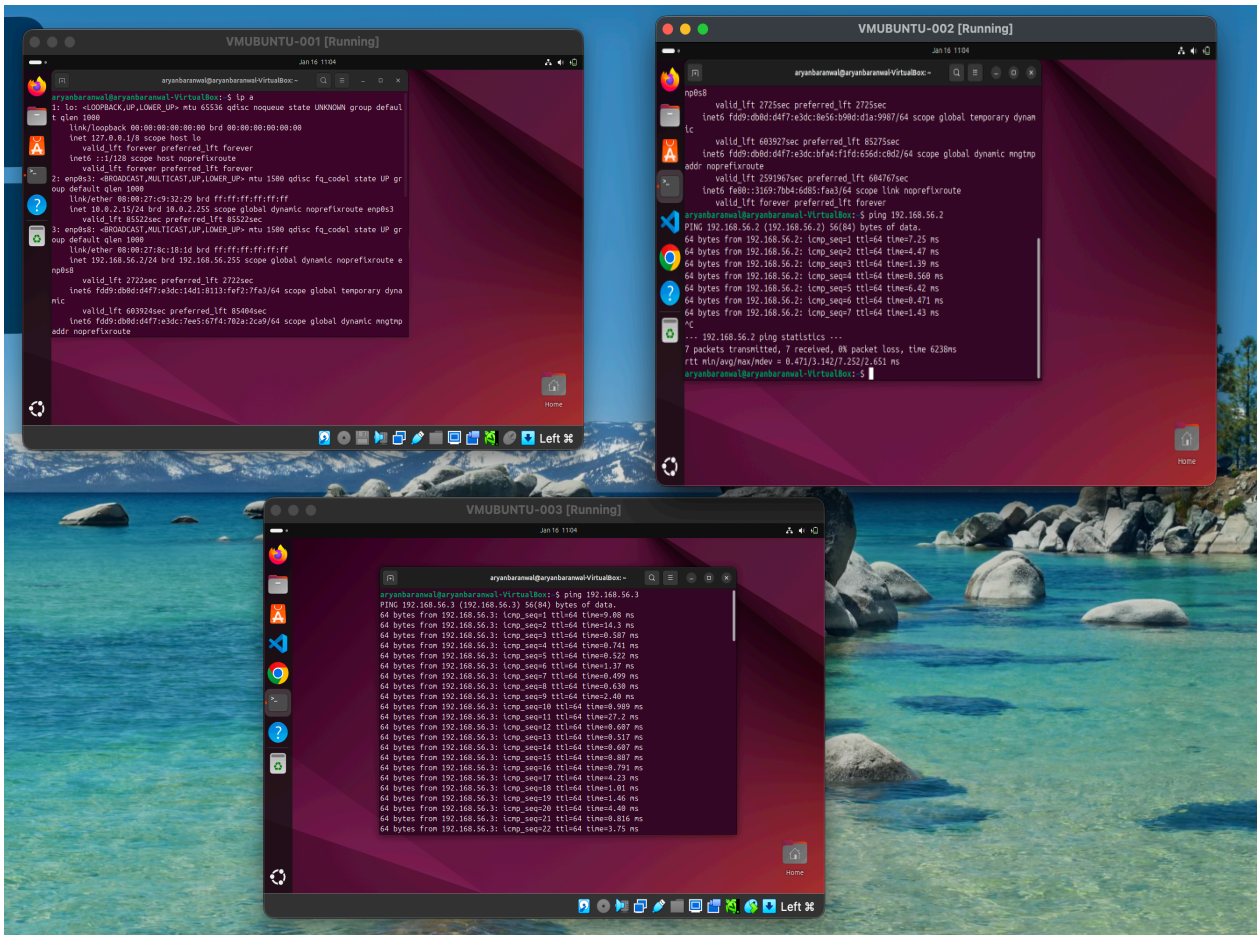
  ping 192.168.56.3

- **From VMUBUNTU-003**

  ping 192.168.56.2

  ping 192.168.56.4

The above ping commands that were successful prove that VMUBUNTU-001, VMUBUNTU-002, and VMUBUNTU-003 are able to communicate with each other. Verification is conducted in the form of screenshots below.



Step 6: Software Installation

As this project is based on the microservice architecture, various software components were deployed on various virtual machines.

- **VMUBUNTU-001:** MySQL Database Server
- **VMUBUNTU-002:** Node.js Backend Microservice
- **VMUBUNTU-003:** React.js Frontend Application

This will provide appropriate separation of database, backend and frontend layers, illustrating a distinct microservice-based implementation within a multiplicity of virtual machines.

Step 7: MySQL Database Setup (VMUBUNTU-001) -

MySQL database had been configured on VMUBUNTU-001 that will be used to perform all the database related activities in the Order Management system.

7.1 Install MySQL

First, the system packages were updated and MySQL server was installed:

→ sudo apt update

→ sudo apt install mysql-server -y

After installation, the MySQL service was started and enabled to run automatically on system boot:

→ sudo systemctl start mysql

→ sudo systemctl enable mysql

7.2 Create Order Management Database

Login to the MySQL console:

→ sudo mysql

Create and select the database:

→ CREATE DATABASE order_management;

→ USE order_management;

Create the required table to store order details:

→ CREATE TABLE orders (

id INT AUTO_INCREMENT PRIMARY KEY,

customer_name VARCHAR(100),

product_name VARCHAR(100),

quantity INT,

price DECIMAL(10,2),

created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP

);

7.3 Enable Remote Database Access

Since the backend microservice runs on a different VM, remote access to MySQL was enabled.

Create a database user for the backend VM and grant required privileges:

→CREATE USER 'order_user'@'192.168.56.102' IDENTIFIED BY 'Order@123';

→ GRANT ALL PRIVILEGES ON order_management.* TO 'order_user'@'192.168.56.102';

→ FLUSH PRIVILEGES;

→ SELECT user, host FROM mysql.user;

Edit the MySQL configuration file to allow external connections:

→ sudo nano /etc/mysql/mysql.conf.d/mysqld.cnf

Update the bind address:

bind-address = 0.0.0.0

Restart MySQL to apply the changes:

→ sudo systemctl restart mysql

After these steps, **VMUBUNTU-002 (backend)** was able to successfully connect to the MySQL database running on **VMUBUNTU-001**.

Step 8: Node.js Backend Setup (VMUBUNTU-002)

The backend microservice was deployed on **VMUBUNTU-002** using **Node.js and Express.js**. This service handles all order-related operations and communicates with the MySQL database hosted on **VMUBUNTU-001**.

First, **Node.js and npm** were installed on the VM. A new backend project was then created, and required dependencies such as Express, MySQL client, CORS, and body-parser were installed.

The backend was configured to connect to the remote MySQL database using the database VM's static IP address. Once the connection was established, RESTful APIs were implemented for creating and fetching orders.

The backend service was started using the command:

→ node server.js

Successful logs confirmed that the Node.js server was running and ready to handle API requests.

Step 9: React Frontend Setup (VMUBUNTU-003)

The frontend application was deployed on **VMUBUNTU-003** using **React.js** to provide a user interface for the Order Management system. After installing Node.js and npm, a new React application was created. Axios was used to enable communication between the frontend and the backend service running on **VMUBUNTU-002**.

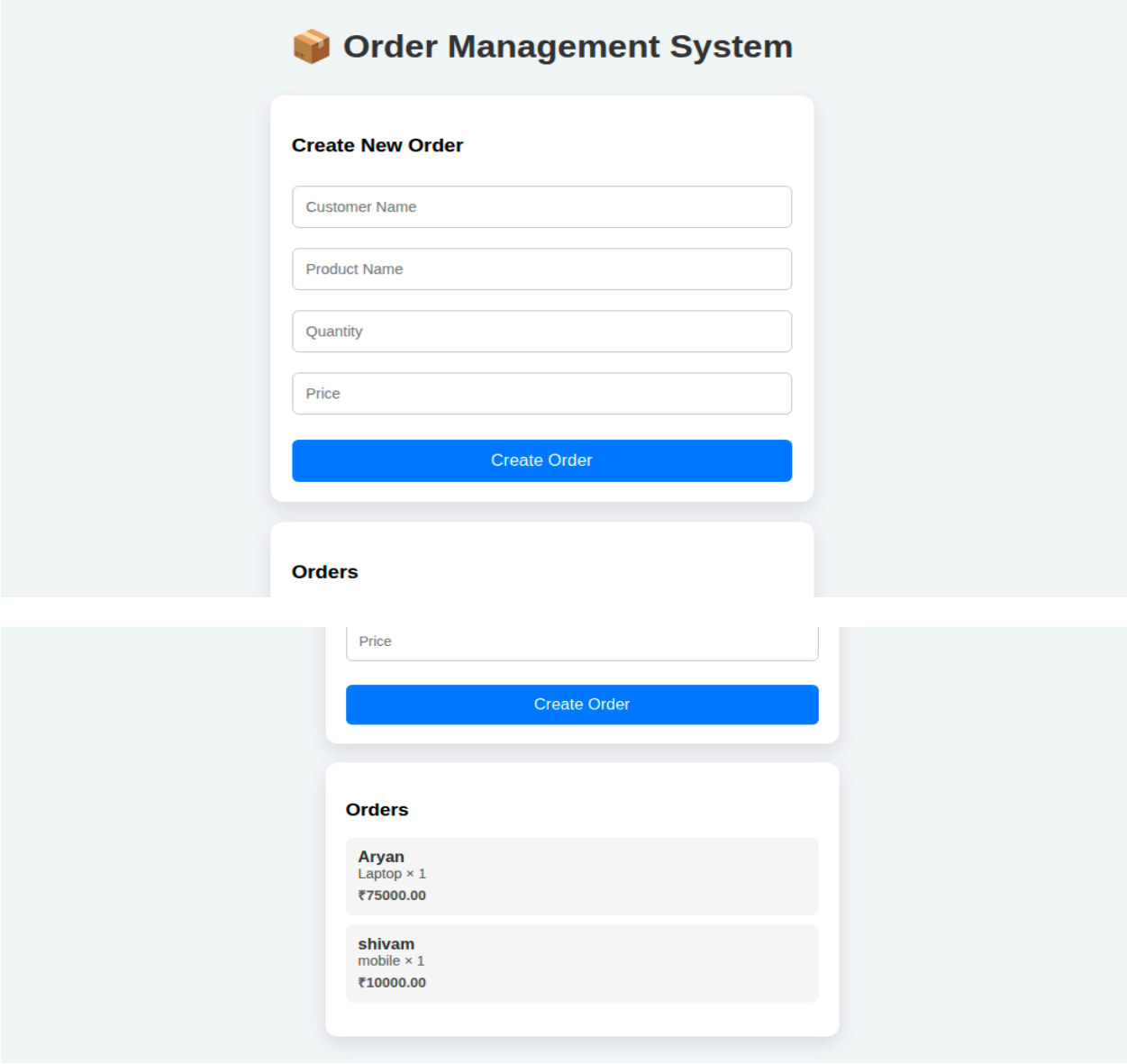
The backend API URL was configured in the frontend to ensure proper connectivity.

The frontend application sends requests to the backend to create and retrieve orders and displays the data on the UI.

The React application was started using:

→ npm start

Once running, the frontend successfully communicated with the backend and displayed data fetched from the MySQL database.



Step 10: API Testing using Postman

For testing the backend microservice, **Postman** was used on the host machine (MacBook). Since the VirtualBox network was configured using **NAT and Host-Only adapters**, the host system was able to communicate directly with the backend service running inside the virtual machine.

Postman was used to verify whether the **Node.js Order Management microservice** deployed on **VMUBUNTU-002** was working correctly and able to communicate with the **MySQL database on VMUBUNTU-001**.

Api Specifications -

1. GET API – Fetch All Orders-

API type	GET
URL	http://192.168.56.3:3000/api/orders

Sample output	[
	{
	"id": 1,
	"customer_name": "Aryan",
	"product_name": "Laptop",
	"quantity": 1,
	"price": "75000.00",
	"created_at": "2026-01-15T13:46:52.000Z"
	},
	{
	"id": 2,
	"customer_name": "shivam",
	"product_name": "mobile",
	"quantity": 1,
	"price": "10000.00",
	"created_at": "2026-01-16T03:00:00.000Z"
	}
	]

2. POST API – Create a New Order -

This API is used to create a new order and store it in the database.

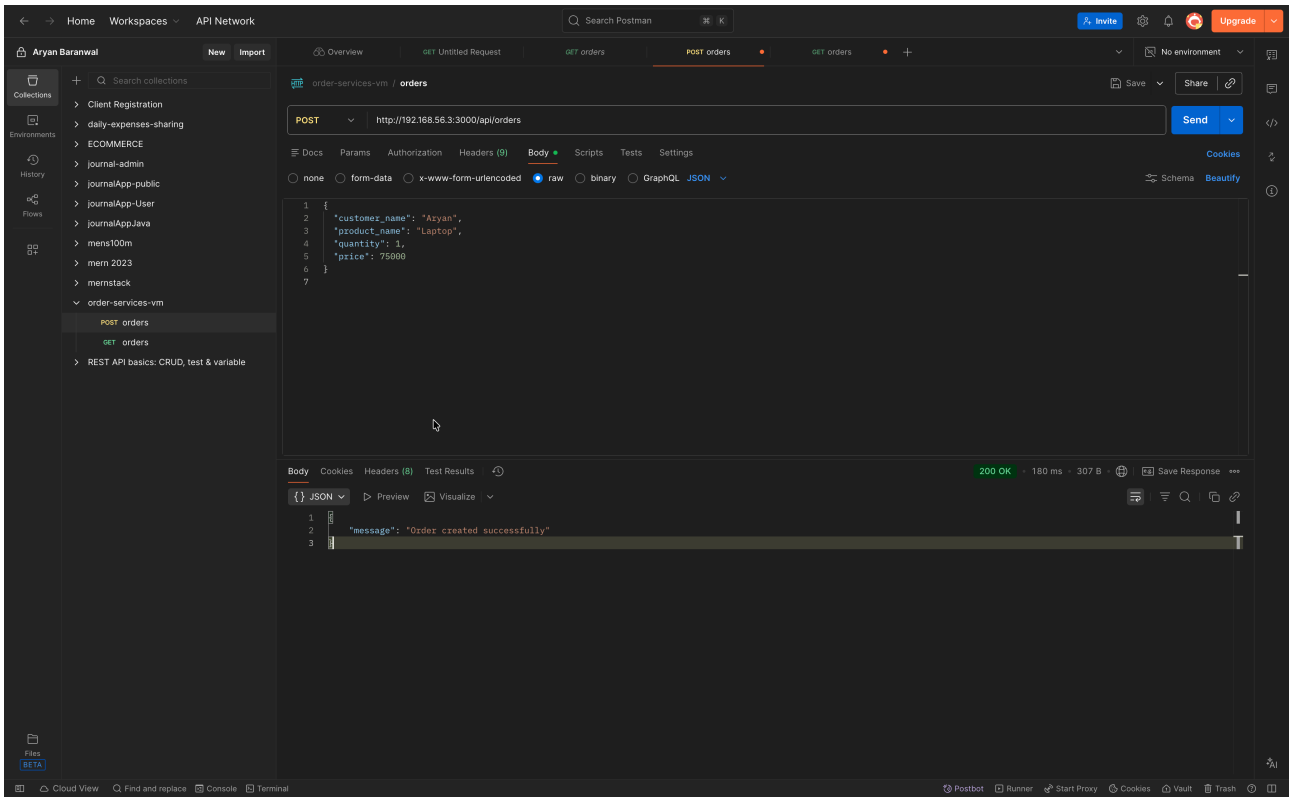
API type	POST
Url	http://192.168.56.3:3000/api/orders
Content-type	application/json

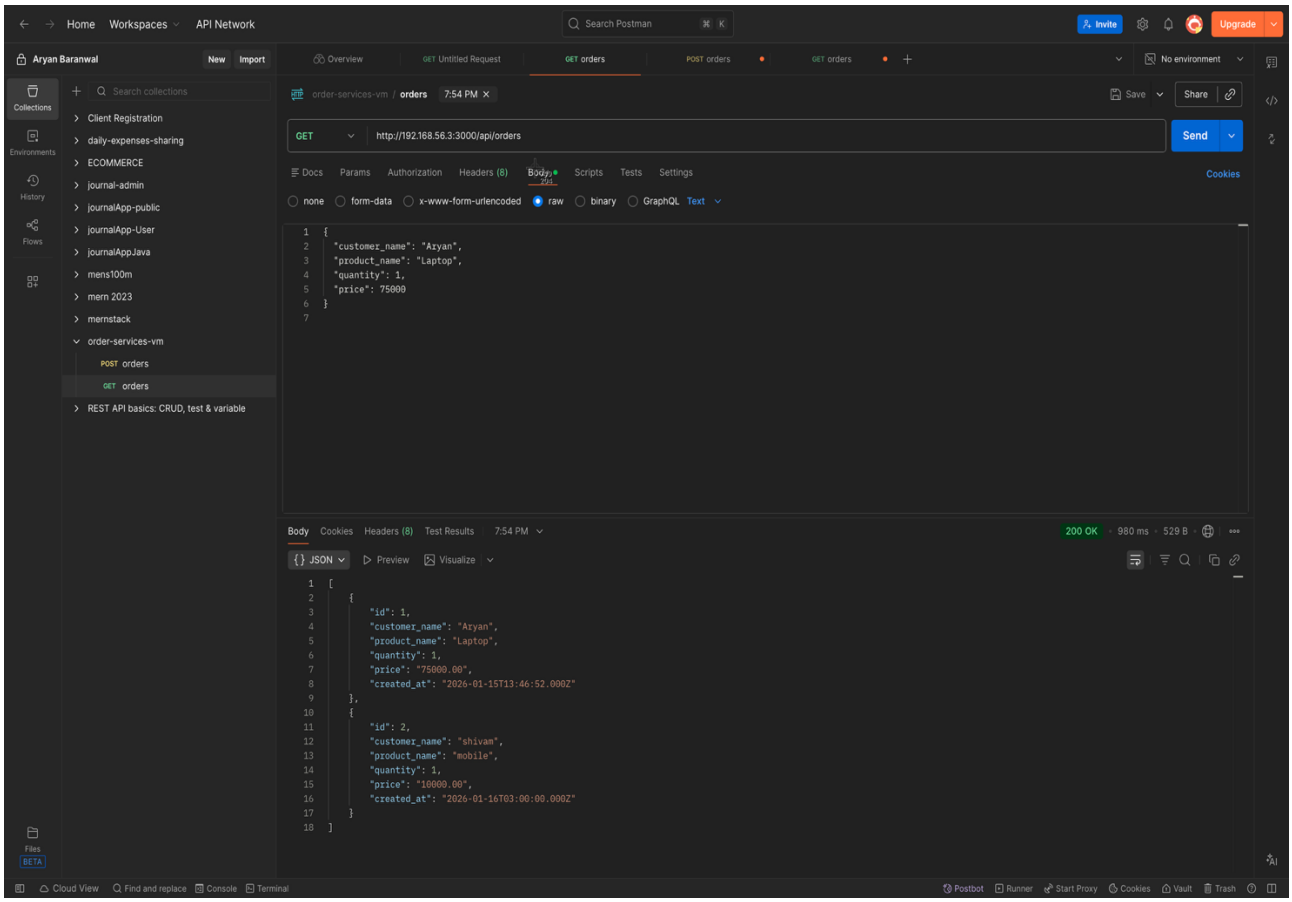
Request body	<pre>{
	  "customer_name": "Aryan",
	  "product_name": "Laptop",
	  "quantity": 1,
	  "price": 75000
	}</pre>

Testing Result

- The **POST API** successfully inserted order details into the MySQL database running on **VMUBUNTU-001**.
- The **GET API** returned the list of all orders, confirming successful data retrieval.
- Logs on the backend VM confirmed that the Node.js service was receiving requests and interacting with the database correctly.

The screenshots below show successful API testing using Postman.


Home
Workspaces
API Network
Search Postman
Invite
Upgrade
Aryan Baranwal
New
Import
Overview
GET Untitled Request
GET orders
POST orders
GET orders
No environment
Collections
Environments
History
Flows
Files
BETA
Search collections
Client Registration
daily-expenses-sharing
ECOMMERCE
journal-admin
journalApp-public
journalApp-User
journalAppJava
mens100m
mern 2023
mernstack
order-services-vm
POST orders
GET orders
REST API basics: CRUD, test & variable
order-services-vm / orders
Save
Share
POST
http://192.168.56.3:3000/api/orders
Send
Docs
Params
Authorization
Headers (9)
Body
Scripts
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Schema
Beautify
{
"customer_name": "Aryan",
"product_name": "Laptop",
"quantity": 1,
"price": 75000
}
Body
Cookies
Headers (8)
Test Results
200 OK
180 ms
307 B
Save Response
{ } JSON
Preview
Visualize
{
"message": "Order created successfully"
}
Cloud View
Find and replace
Console
Terminal
Postbot
Runner
Start Proxy
Cookies
Vault
Trash


Home
Workspaces
API Network
Search Postman
Invite
Upgrade
Aryan Baranwal
New
Import
Overview
GET Untitled Request
GET orders
POST orders
GET orders
No environment
Collections
Environments
History
Flows
Files
BETA
Search collections
Client Registration
daily-expenses-sharing
ECOMMERCE
journal-admin
journalApp-public
journalApp-User
journalAppJava
mens100m
mern 2023
mernstack
order-services-vm
POST orders
GET orders
REST API basics: CRUD, test & variable
order-services-vm / orders 7:54 PM
Save
Share
GET
http://192.168.56.3:3000/api/orders
Send
Docs
Params
Authorization
Headers (8)
Body
Scripts
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
Text
{
"customer_name": "Aryan",
"product_name": "Laptop",
"quantity": 1,
"price": 75000
}
Body
Cookies
Headers (8)
Test Results
7:54 PM
200 OK
980 ms
529 B
{ } JSON
Preview
Visualize
[
{
"id": 1,
"customer_name": "Aryan",
"product_name": "Laptop",
"quantity": 1,
"price": "75000.00",
"created_at": "2026-01-15T13:46:52.000Z"
},
{
"id": 2,
"customer_name": "shivam",
"product_name": "mobile",
"quantity": 1,
"price": "10000.00",
"created_at": "2026-01-16T03:00:00.000Z"
}
]
Cloud View
Find and replace
Console
Terminal
Postbot
Runner
Start Proxy
Cookies
Vault
Trash

Architecture Diagram

It is based on a three-level microservice architecture with each significant part being implemented in its own virtual machine. Such a design provides superior modularity, scaling and responsibility separation.

VMUBUNTU-001: Database Layer (MySQL)

VMUBUNTU-001 is focused on the database layer and it is the host of MySQL server. It holds all the information regarding orders like customer information, product information, quantity and price. Remote access has been allowed whereby only the backend service can communicate with the database using a secure database user and fixed IP address.

VMUBUNTU-002: Backend Layer (Node.js Microservice)

The Node.js Order Management microservice created with the help of Express.js is hosted in VMUBUNTU-002. This layer is used as an intermediary between the front-end and the database.

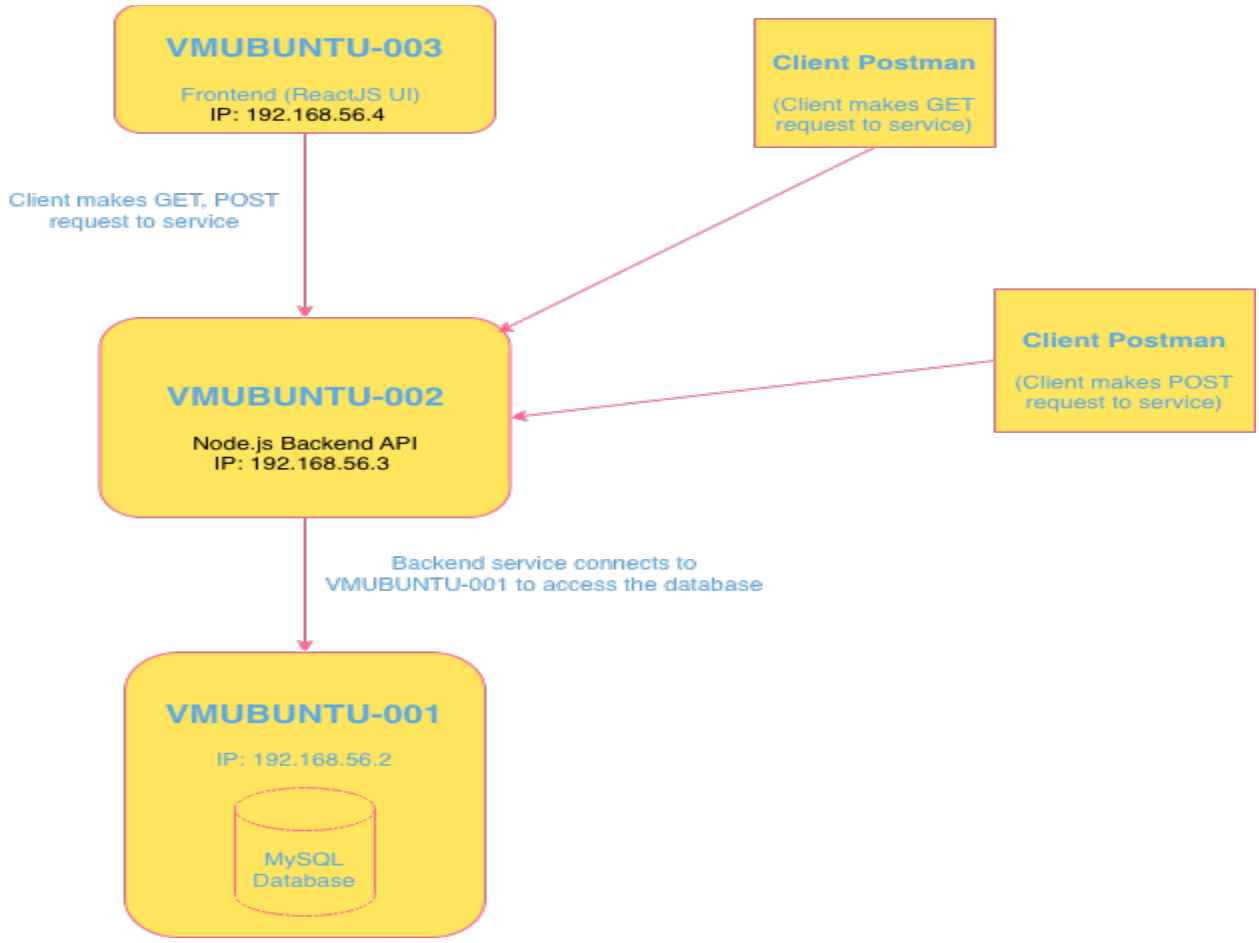
The backend has the following responsibilities:

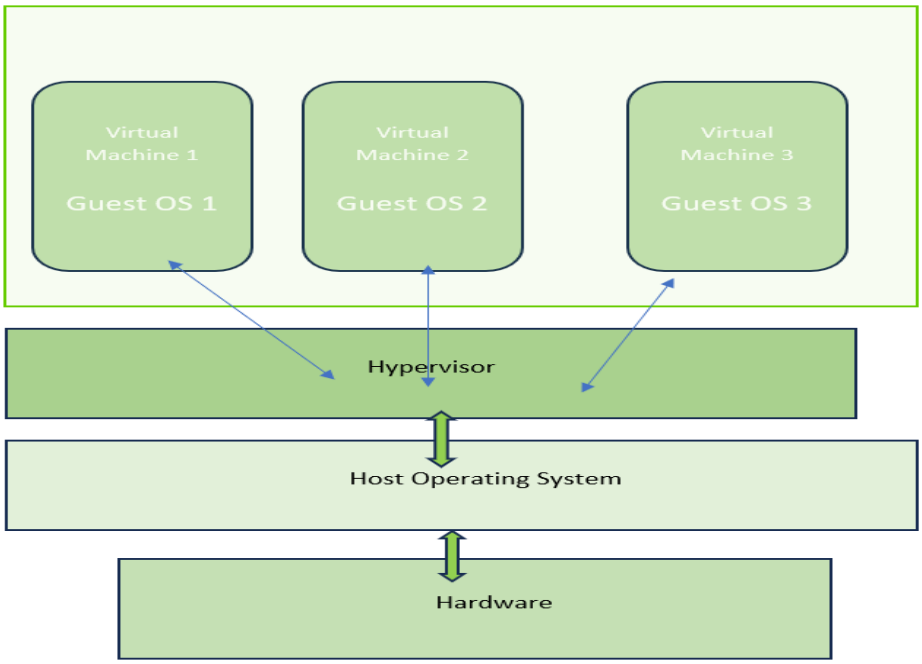
- Exposing RESTful APIs for order creation and retrieval
- Handling business logic
- A secure connection to VMUBUNTU-001 MySQL database.

The backend connects to the database with SQL queries and is able to respond in the form of a JSON to the frontend.

VMUBUNTU-003: Frontend Layer (React.js)

The frontend application of the system is the React.js that is hosted on VMUBUNTU-003 and that has the user interface of the system. This layer interacts with the users to create new orders and to see the existing orders. The frontend makes HTTP requests to the backend microservice in VMUBUNTU-002. It does not connect directly to the database and has the right security and architectural distance.


VMUBUNTU-003
Frontend (ReactJS UI)
IP: 192.168.56.4
Client Postman
(Client makes GET request to service)
Client makes GET, POST request to service
VMUBUNTU-002
Node.js Backend API
IP: 192.168.56.3
Client Postman
(Client makes POST request to service)
Backend service connects to VMUBUNTU-001 to access the database
VMUBUNTU-001
IP: 192.168.56.2
MySQL Database


Virtual Machine 1
Guest OS 1
Virtual Machine 2
Guest OS 2
Virtual Machine 3
Guest OS 3
Hypervisor
Host Operating System
Hardware

Communication Flow

1. The user interacts with the **React frontend (VMUBUNTU-003)**.
2. The frontend sends API requests to the **Node.js backend (VMUBUNTU-002)**.
3. The backend processes the request and interacts with the **MySQL database (VMUBUNTU-001)**.
4. The database returns the result to the backend.
5. The backend sends the response back to the frontend.
6. The frontend displays the data to the user.

Source Code Repository

GitHub Repository Link:
https://github.com/aryan9867bar/vm-order-management-microservices

Video Demonstration

Video Demo Link:
https://drive.google.com/file/d/1ketCheW5Q4qKOwZnTIA0uCUYIEDSxHIw/view?usp=sharing

Conclusion

Overall, this assignment was able to prove the deployment of multi-VMs, networking, and application architecture based on microservices, using Node.js, MySQL and ReactJS.