Report

on

Use of VirtualBox to Create Multiple Virtual Machines (VMs), Connect these VMs, and Host One Microservice-Based Application

Course Code
CSL7510

submitted by
Ankit Kumar Chauhan (M23CSA509)
M.Tech- Artificial Intelligence (Executive)

Course instructor
Dr. SUMIT KALRA

Department of Computer Science and Engineering

Indian Institute of Technology Jodhpur
NH 65, Nagaur Road, Karwar,
Jodhpur 342030, INDIA

Objective:

To create and configure multiple Virtual Machines (VMs) using VirtualBox, establish a network between them, and deploy a microservice-based application across the connected VMs.

❖ Step-by-Step Instructions for Implementation:

1. Installation of VirtualBox and Creation of VMs

1. Install VirtualBox:

- Downloaded and installed VirtualBox from the official website. As my host machine is windows so I installed the respective release.
- In order to finish the installation, simply double-click and follow the setup wizard.

2. Create Virtual Machines:

- Opened VirtualBox and click **New** to create a new VM.
- Configured the following settings:
  - **Name**: VM1_WebServer
  - **OS Type**: Linux → Lubuntu (64-bit).
  - **Memory Size**: Allocated 2GB.
  - **Processors**: 2 CPU.
  - **Virtual Disk**: Create a VDI file of size 25GB.
  - **Video Memory**: 128 MB.
  - **Graphics Controller**: VMSVGA (with enabled 3D acceleration).
  - **Network**: Bridged Adapter.
  - **Shared Folder**: assignment1.
- After completing the above configuration, start the machine and install the **Lubuntu** by following the setup wizard.
- Shutdown the VM1_Webserver machine.
- Right click the VM1_Webserver and click clone.
  - Configured the settings as
  - **Name**: VM2_ApiServer

- **Full Clone**: Yes
- **Mac Address Policy**: Generate new Mac addresses for all network adapters.

- Right click the VM1_Webserver and click clone.
  - Configured the settings as
  - **Name**: VM3_DBServer
  - **Full Clone**: Yes
  - **Mac Address Policy**: Generate new Mac addresses for all network adapters.
- The below Fig. 1 shows the virtual box with all three machines created and configured.

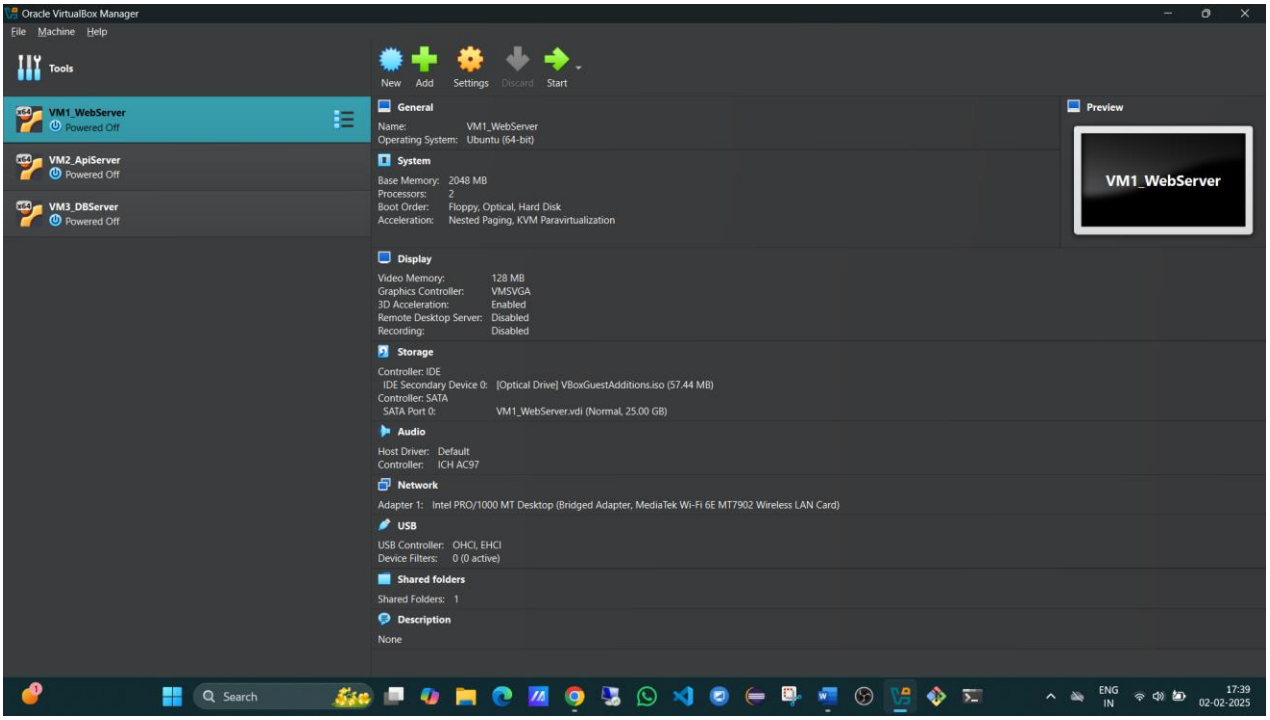

Fig. 1. Successful Creation and Configuration of Three Virtual Machines

2. Configuration of Network Settings to Connect VMs

- Start VM1_WebServer and manually assigned the IP details as-

  IP: 192.168.29.131

Netmask: 255.255.255.0

Gateway: 192.168.29.1

- Start VM2_ApiServer Machine and manually assigned the IP details as-

  IP: 192.168.29.132

  Netmask: 255.255.255.0

  Gateway: 192.168.29.1

- Start VM3_DBServer Machine and manually assigned the IP details as-

  IP: 192.168.29.133

  Netmask: 255.255.255.0

  Gateway: 192.168.29.1

- The screenshots of below Fig.2 demonstrate the proper communication between all four machines, including VM1, VM2, VM3, and the host machine.

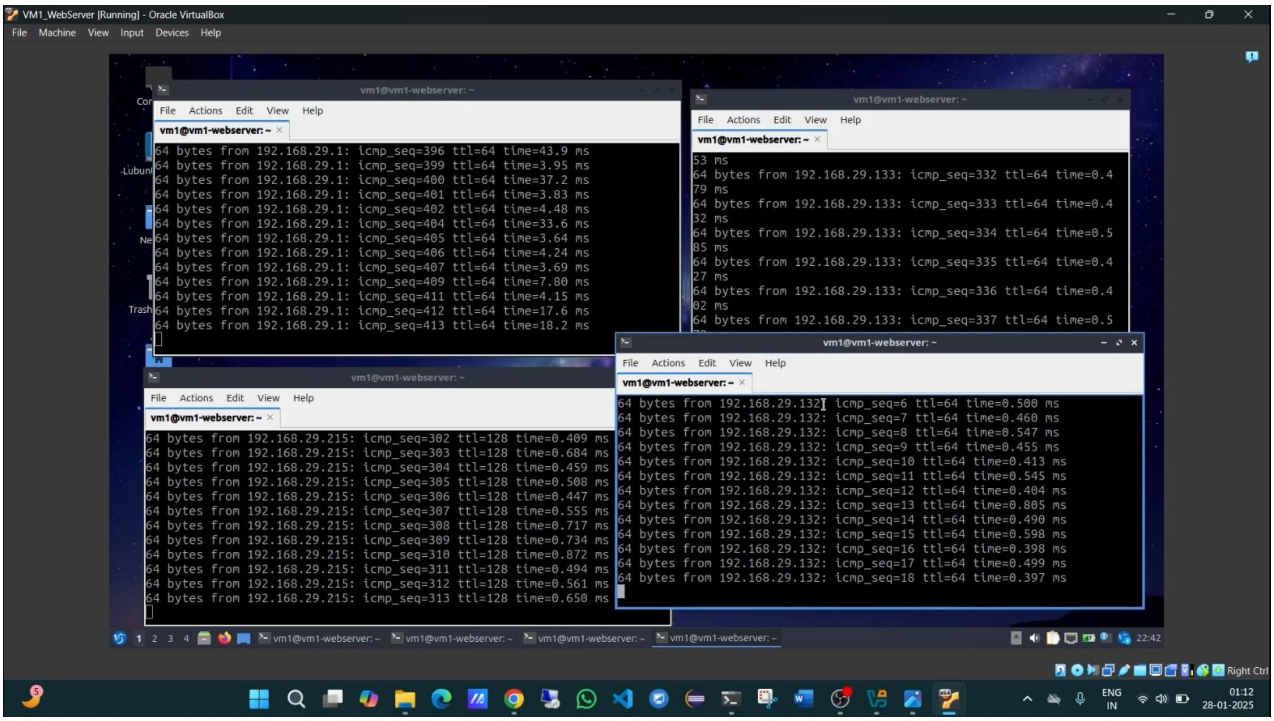

Fig. 2. VM1 is pinging Router(192.168.29.1), VM2(192.168.29.132), VM3(192.168.29.133) and Host(192.168.29.215)

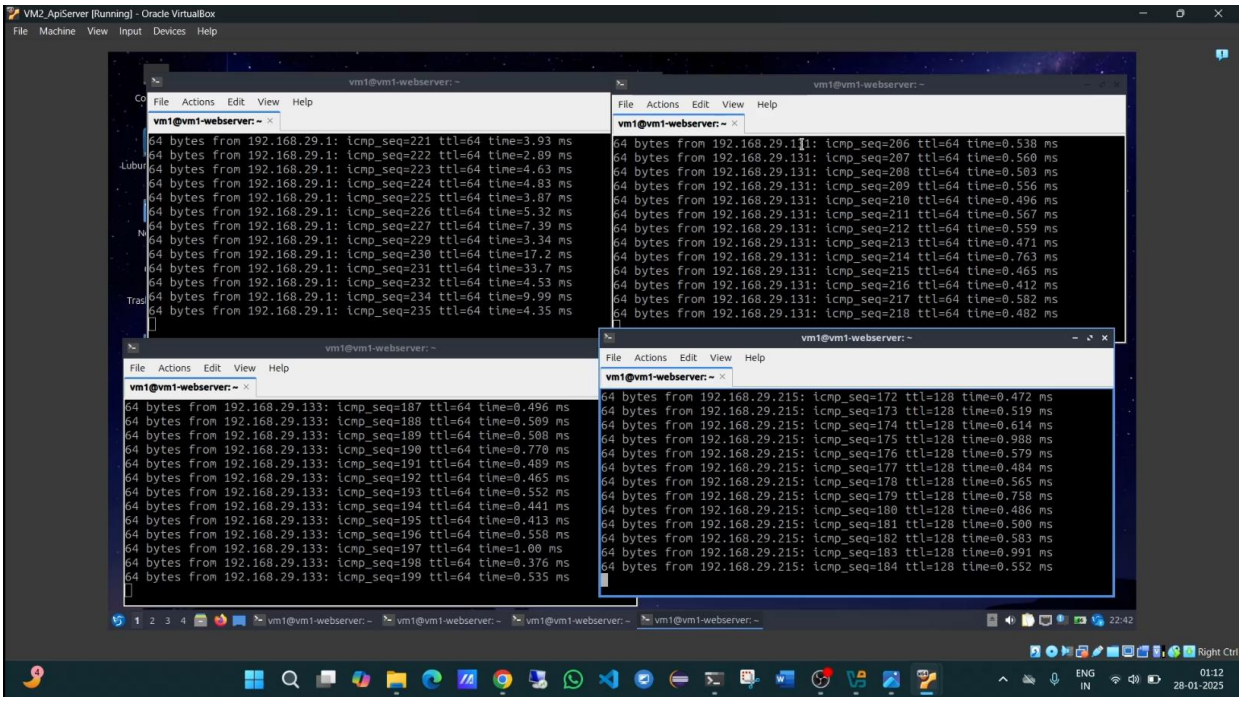

Fig. 3. VM2 is pinging Router(192.168.29.1), VM1(192.168.29.131), VM3(192.168.29.133) and Host(192.168.29.215)

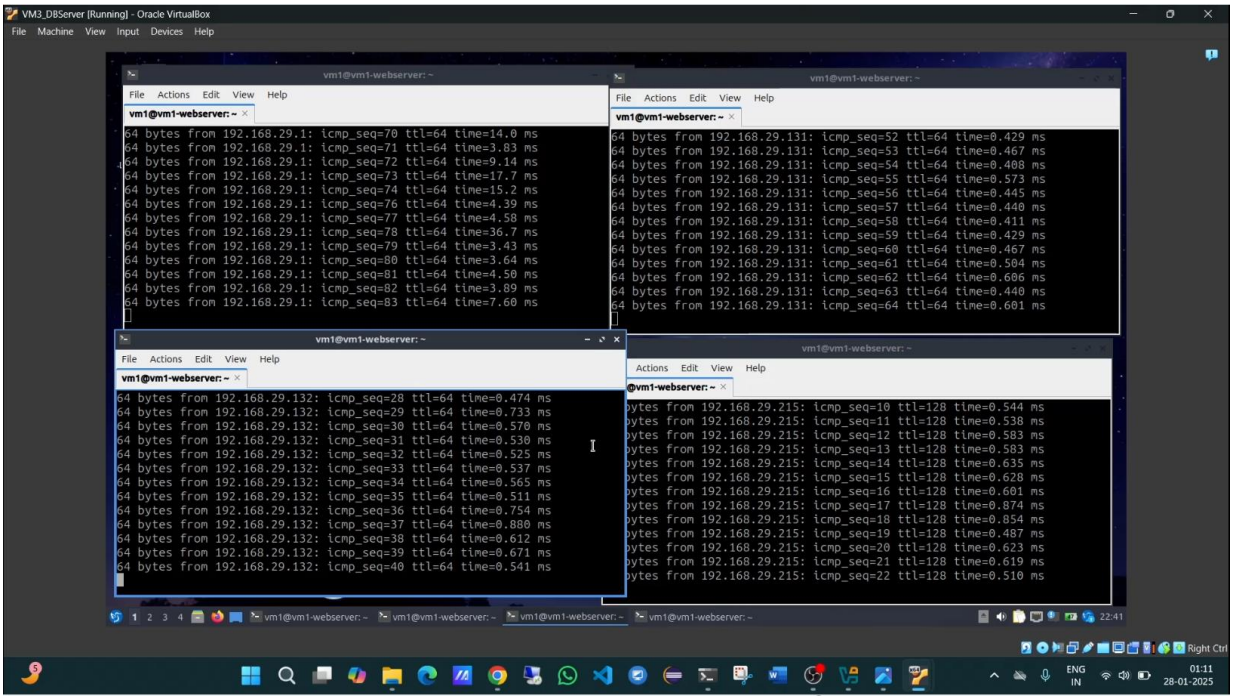

Fig. 4. VM3 is pinging Router(192.168.29.1),VM1(192.168.29.131),VM2(192.168.29.132) and Host(192.168.29.215)

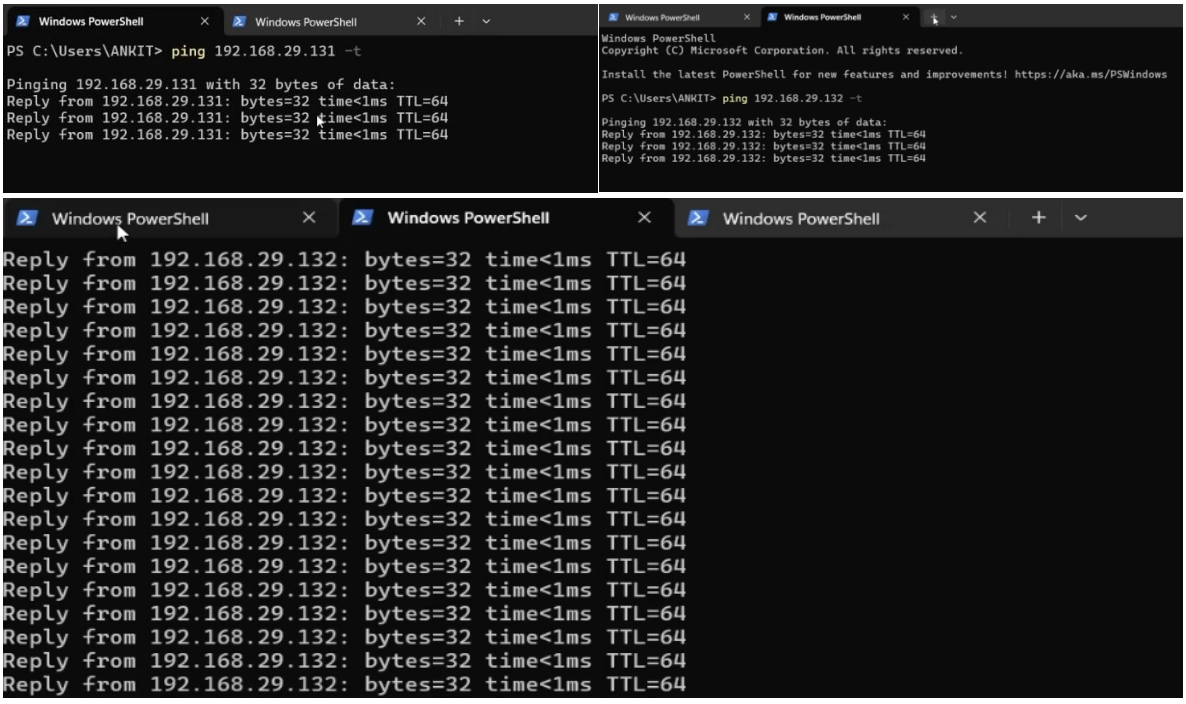

Fig. 5. Host is pinging VM1(192.168.29.131),VM2(192.168.29.132) and VM3(192.168.29.133)

3. Deployment of Microservice Application

Each VM has a specific role in hosting the microservice application. Follow the configurations that are explained in the scripts below.

VM3_DBServer Setup:

- Script: vm3setup.sh.
- Purpose: Host the database for the User Management system.
    - Install MySQL server and set up a ‘mydb’ database.
    - Create a ‘users’ table and insert initial record.
    - Create a user named ‘ankit’ to be used to connect to ‘mydb’.
    - Configure MySQL to accept connections from other VMs.
- The below Fig. 6. shows the successful installation and the setup of a MySQL server.

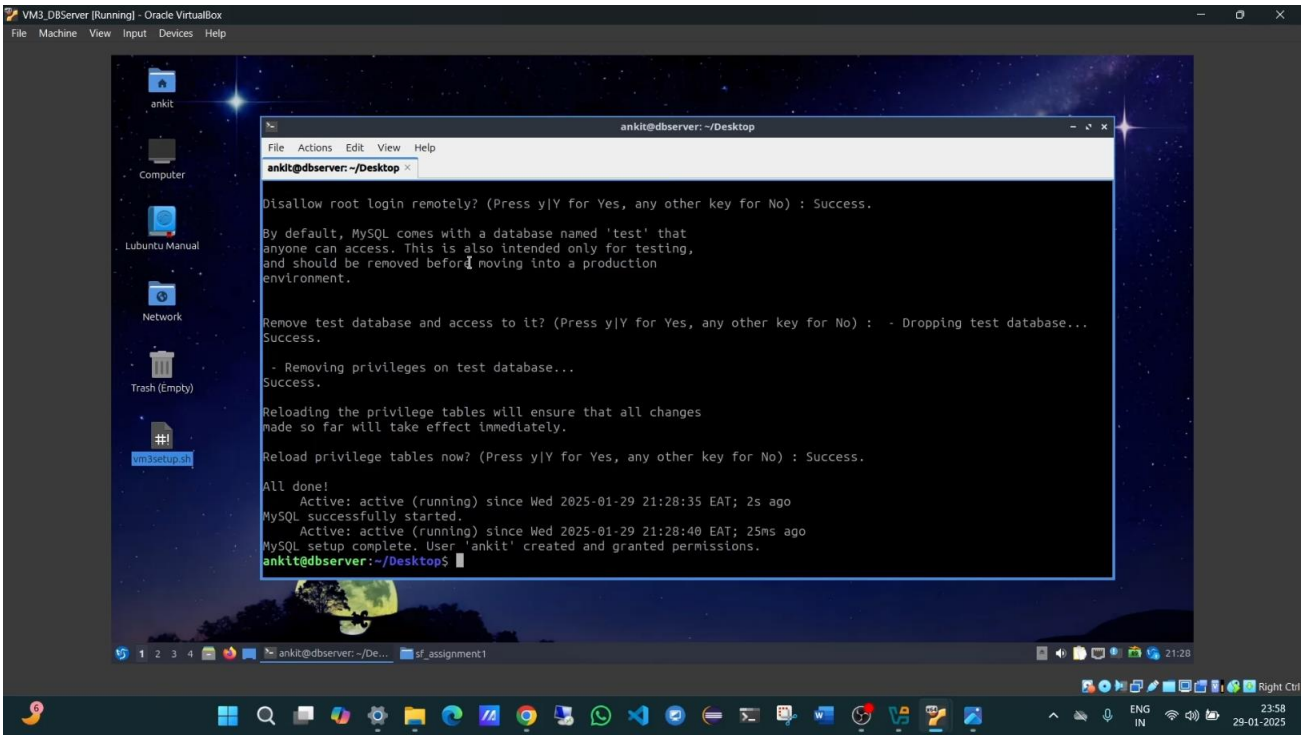

Fig. 6. Successful setup of VM3 (Database Server)

VM2_ApiServer Setup:

- Script: vm2setup.sh.
- Purpose: Host the backend JSON-based RESTful API using Node.js and Express.
    - Install Node.js and dependencies (Express, MySQL client).
    - Define and code RESTful API functions for CRUD operations.
    - Configure CORS to allow connections from VM1_WebServer.
- The Below Fig. 7 shows the successful installation and the setup of a microservice server.

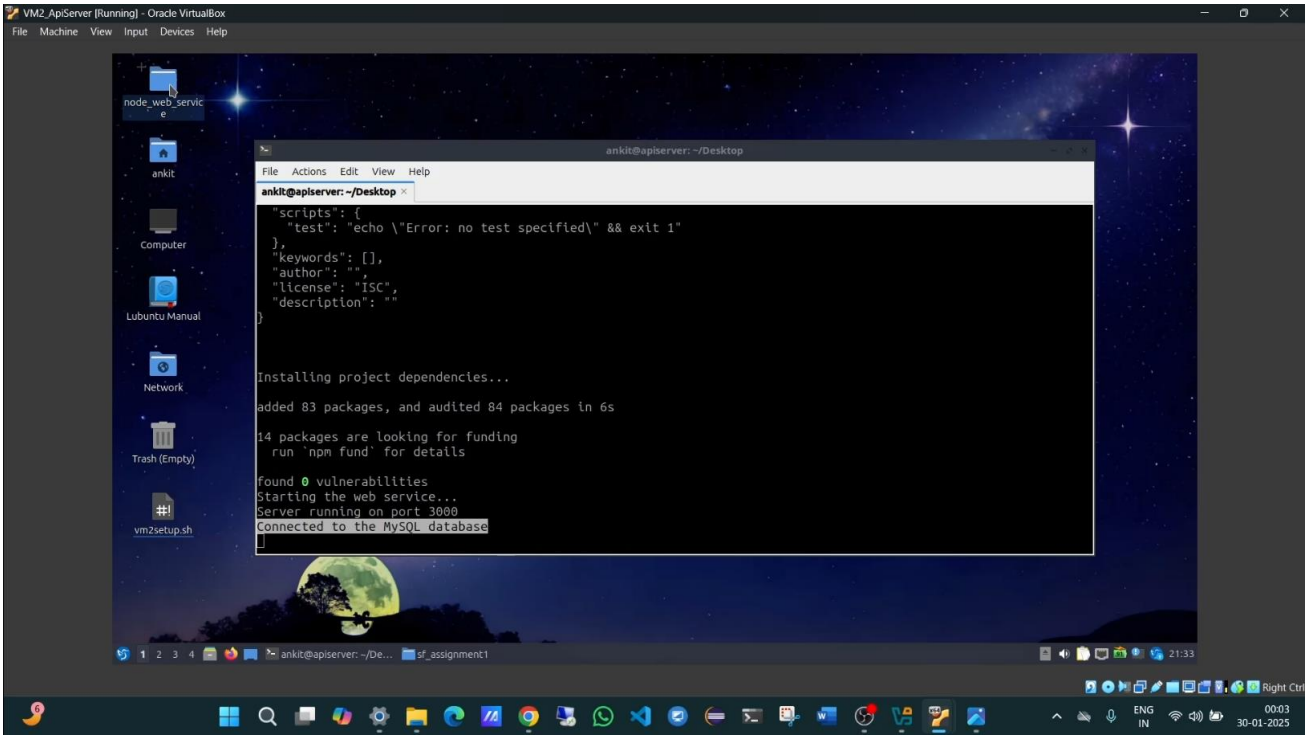

Fig. 7. Successful setup of VM2 (Api Server)

VM1_WebServer Setup:

- Script: vm1setup.sh.
- Purpose: Host the front-end for the User Management system using Lighttpd.
    - Install Lighttpd and create an HTML-based user interface.
    - Create the html page at designate location of /var/www/.
    - The HTML page interacts with the API hosted on VM2_ApiServer.
- The below Fig. 8 shows the successful installation and the setup of a web server.

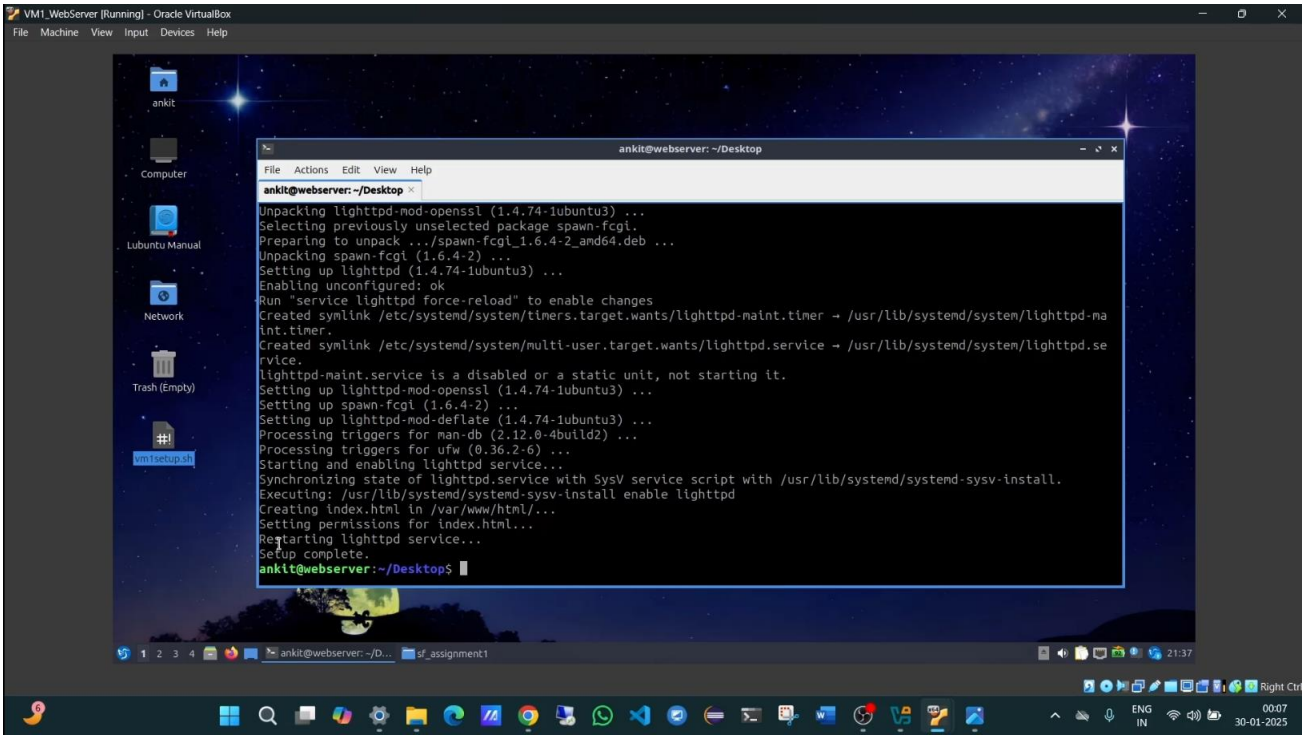

Fig. 8. Successful setup of VM1 (Web Server)

❖ Architecture Design:

a. Diagram:

- A diagram of Fig. 9 is provided below showing the architecture of network components:
    - **VM1_WebServer**: Hosts the Lighttpd server and static web page for user management.
    - **VM2_ApiServer**: Provides RESTful API endpoints for CRUD operations.
    - **VM3_DBServer**: Stores user data in a MySQL database.

b. Network Configuration:

- **IP Range**: 192.168.29.xxx
- **Host (Bridged)**: 192.168.29.215
- **Router**: 192.168.29.1

- **VM Static IPs**:
    - VM1_WebServer: 192.168.29.131
    - VM2_ApiServer: 192.168.29.132
    - VM3_DBServer: 192.168.29.133

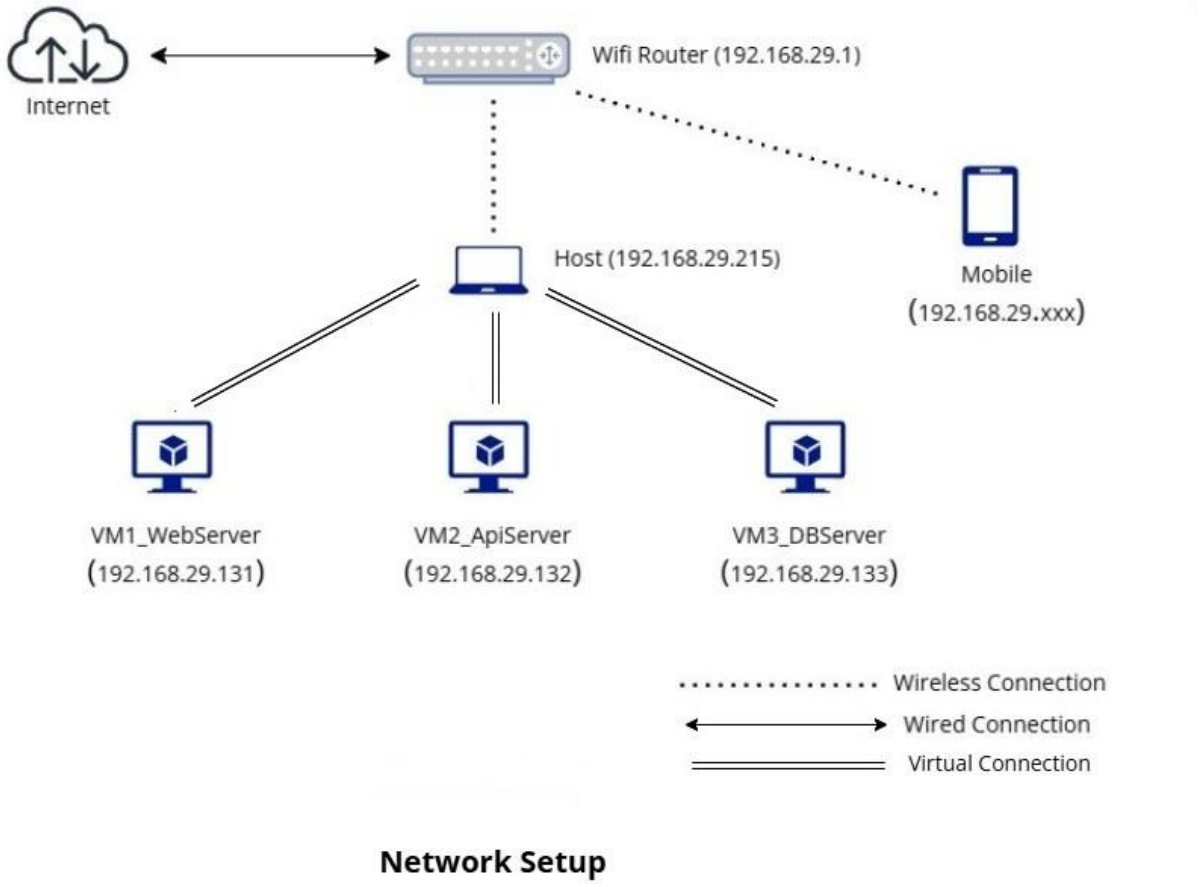

Fig. 9. Architecture Design

❖ Testing and Validation Summary:

a. Application Workflow

The User Management application was tested thoroughly to ensure seamless integration across the three VMs:

1. **Front-End (VM1_WebServer)**:
    - Hosted on Lighttpd and accessible via http://192.168.29.131 (Fig. 10 and Fig. 11)
    - CRUD operations were tested through the user-friendly interface:
        - **Create**: Adding new users from the front-end form.
        - **Read**: Displaying all users in the table.
        - **Update**: Editing user details from the table.
        - **Delete**: Removing users via the delete button.
    - Each operation triggered the API endpoints hosted on **VM2_ApiServer**.
2. **Back-End (VM2_ApiServer)**:
    - Hosted on Node.js and responsible for handling API requests (e.g., GET, POST, PUT, DELETE) at http://192.168.29.132:3000/users.
    - Verified API responses ensured accurate communication with the database on **VM3_DBServer**.
3. **Database (VM3_DBServer)**:
    - Hosted on MySQL and stored all user records.
    - The database operations (insert, update, delete, and fetch) were validated via API calls from VM2.

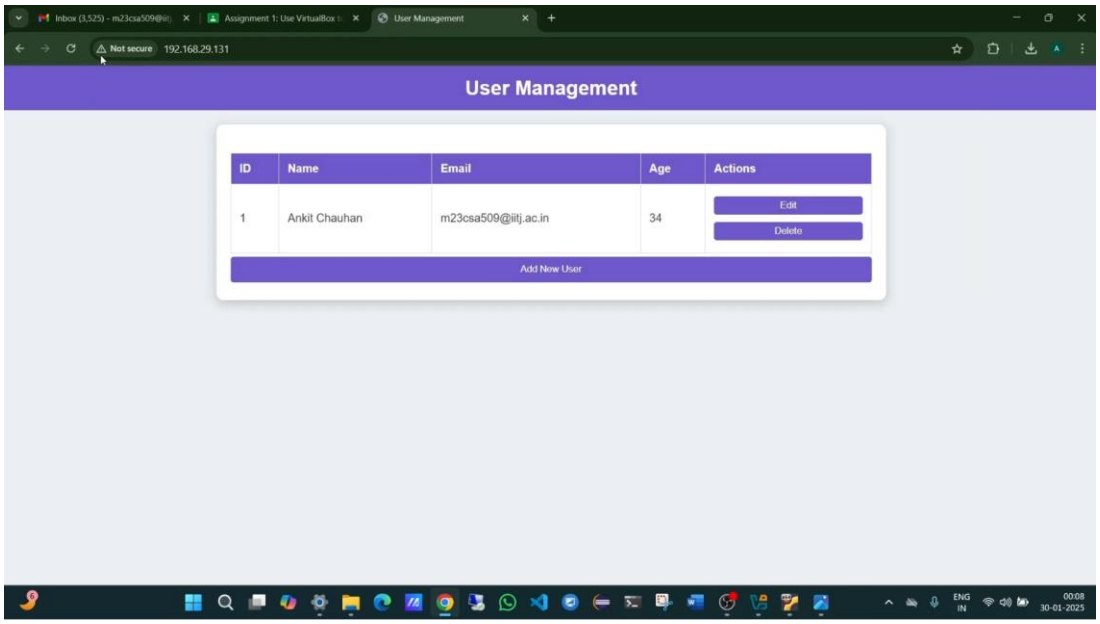

Fig. 10. Accessing application running on VM1_WebServer from Host machine

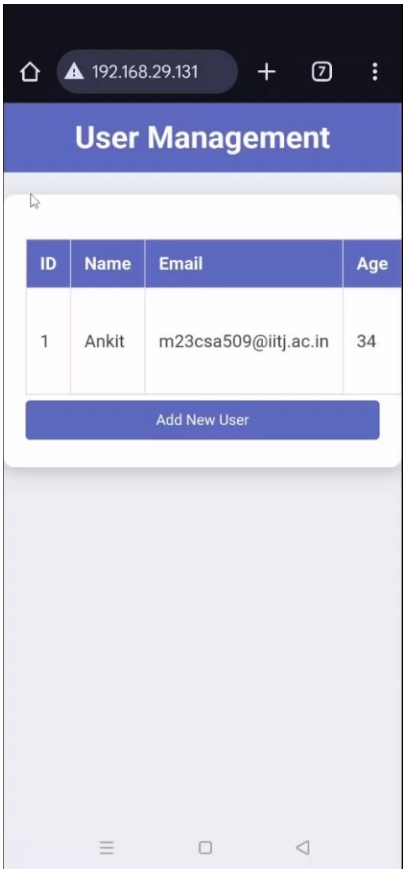

Fig. 11. Accessing application running on VM1_WebServer from Mobile Phone

b. System Testing Results

Component	Operation	Action Performed	Expected Outcome	Actual Outcome	Status
VM1_WebServer	Create User	Added a new user using the front-end form.	User record created in the database and displayed in the table.	User successfully added and displayed.	☑ **Successful**
VM1_WebServer	Read Users	Loaded the front-end page to display all users.	User records fetched from the database and displayed.	Records loaded and displayed correctly.	☑ **Successful**

VM1_WebServer	Update User	Edited an existing user's details from the front-end.	Updated user record reflected in the database and front-end.	User updated successfully.	☑ **Successful**
VM1_WebServer	Delete User	Deleted an existing user using the front-end delete button.	User record removed from the database and front-end table.	User deleted successfully.	☑ **Successful**
VM2_ApiServer	API Integration	Tested API endpoints for CRUD operations (e.g., GET, POST, PUT, DELETE)	Correct database operations for all requests.	API endpoints responded as expected.	☑ **Successful**
VM3_DBServer	Database Connection	Verified MySQL database connectivity from VM2_ApiServer.	Database connection established successfully.	Database connection successful.	☑ **Successful**
Network Connectivity	VM Communication	Tested ping between VM1 ↔ VM2, VM2 ↔ VM3, and all VMs ↔ Host.	All VMs should communicate without packet loss.	Ping successful with zero packet loss.	☑ **Successful**

❖ Observations and Insights:

- **Performance**: CRUD operations executed with minimal latency, with seamless interactions between VMs.
- **Reliability**: No failures were observed during tests, and all operations were consistent and accurate.

- **Scalability**: The architecture can be scaled horizontally by adding additional VMs for load balancing or redundancy.
- **Network Robustness**: All VMs communicated effectively via the configured bridged adapter, ensuring stable and reliable connectivity.

❖ References:

1. VirtualBox Documentation
2. Ubuntu Networking Configuration
3. Lighttpd Web Server Setup
4. Node.js Official Documentation
5. Express.js API Reference
6. MySQL Documentation
7. RESTful API Design Best Practices
8. Networking in Virtual Machines

➢ **GitHub Link:**

https://github.com/Ankit-IITJ/VCC_Assignment1.git

➢ **Recorded Video Link:**

https://drive.google.com/file/d/1QDCAUbed6xKeE-9bbDE1b2HHhfOjpGTB/view?usp=sharing