Virtualisation in cloud Computing

Assignment Document Work Report:--

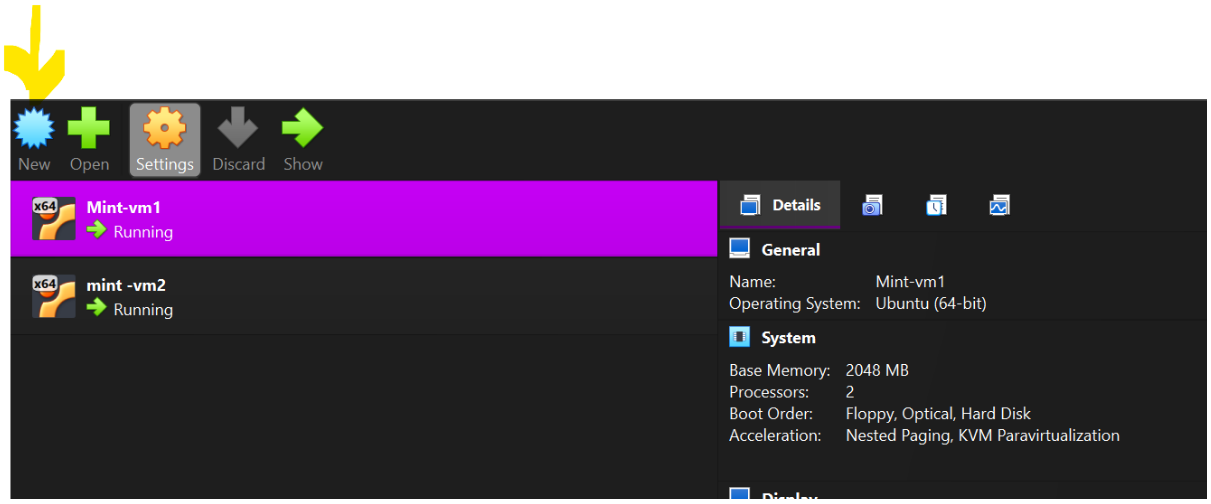
Objective:
Create and configure multiple Virtual Machines (VMs) using VirtualBox, establish a network between them, and deploy a microservice-based application across the connected VMs.

Main steps:

A. **Step-by-Step Instructions for Implementation:**
B. **Installation of VirtualBox and creation of multiple VMs.**
C. **Configuration of network settings to connect the VMs.**

Step:1 Downloaded the virtual box: --

Step:2 added 2 virtual machines of Linux Mint Operating system by using add the configuration



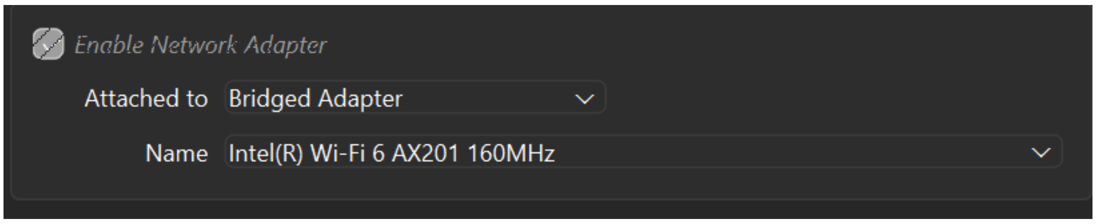
Mint-vm1
Running

mint -vm2
Running

General
Name: Mint-vm1
Operating System: Ubuntu (64-bit)

System
Base Memory: 2048 MB
Processors: 2
Boot Order: Floppy, Optical, Hard Disk
Acceleration: Nested Paging, KVM Paravirtualization

Step 3: once configured go to Settings→network -→bridged adapter for bring up the communications between the 2 virtual machines

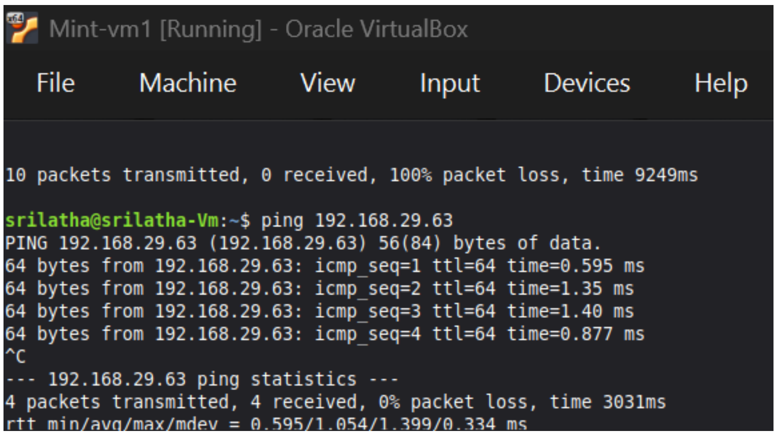


Enable Network Adapter
Attached to: Bridged Adapter
Name: Intel(R) Wi-Fi 6 AX201 160MHz

Step 4: once installed the os inside virtual box we have to know the of 2 Virtual machines by using ip addr here: VM 1 IP ADDRESS:192.162.29.90 and VM2 IP ADDRESS: 192.162.29.63

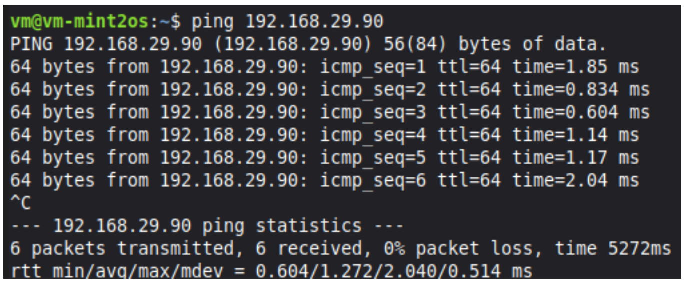
Step 5: We have to Use PING →Packet internet Groper to confirm the reachability and connection checks between 2 hosts

Virtual machine 1 to VM 2 ping status



```
10 packets transmitted, 0 received, 100% packet loss, time 9249ms

srilatha@srilatha-Vm:~$ ping 192.168.29.63
PING 192.168.29.63 (192.168.29.63) 56(84) bytes of data.
64 bytes from 192.168.29.63: icmp_seq=1 ttl=64 time=0.595 ms
64 bytes from 192.168.29.63: icmp_seq=2 ttl=64 time=1.35 ms
64 bytes from 192.168.29.63: icmp_seq=3 ttl=64 time=1.40 ms
64 bytes from 192.168.29.63: icmp_seq=4 ttl=64 time=0.877 ms
^C
--- 192.168.29.63 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3031ms
rtt min/avg/max/mdev = 0.595/1.054/1.399/0.334 ms
```

From Virtual machine 2 to VM 1 ping status



```
vm@vm-mint2os:~$ ping 192.168.29.90
PING 192.168.29.90 (192.168.29.90) 56(84) bytes of data.
64 bytes from 192.168.29.90: icmp_seq=1 ttl=64 time=1.85 ms
64 bytes from 192.168.29.90: icmp_seq=2 ttl=64 time=0.834 ms
64 bytes from 192.168.29.90: icmp_seq=3 ttl=64 time=0.604 ms
64 bytes from 192.168.29.90: icmp_seq=4 ttl=64 time=1.14 ms
64 bytes from 192.168.29.90: icmp_seq=5 ttl=64 time=1.17 ms
64 bytes from 192.168.29.90: icmp_seq=6 ttl=64 time=2.04 ms
^C
--- 192.168.29.90 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5272ms
rtt min/avg/max/mdev = 0.604/1.272/2.040/0.514 ms
```

In terminal **Nodes:**

- **VM1 (Sender):** IP 192.168.29.90
- **VM2 (Listener):** IP 192.168.29.63

Step 6

Setting up the Listener (VM2) In the VM2 terminal, we opened a port to wait for incoming data: nc -l 5000

Step7: Initiating the Sender (VM1) In the VM1 terminal, we connected to VM2's IP and port: nc 192.168.29.63 5000

Data Transfer

- **Action:** Typed "Hi" in VM1 and pressed Enter.

Step 8: Result: The message appeared instantly on the VM2 terminal.

D.Deployment of a simple microservice application (e.g., a RESTful API or a Node.js-based service) across the VMs.

Step:1 The command sudo apt update was executed to refresh the local package index from the Linux Mint and Ubuntu repositories. This ensures that the system is aware of the latest available software versions and security updates.

After updating the package list, the command sudo apt install nodejs npm -y was used to install Node.js and npm. The output indicates that Node.js and npm were already installed and were at their latest available versions. Therefore, no new packages were installed, and no upgrades were required at this stage.

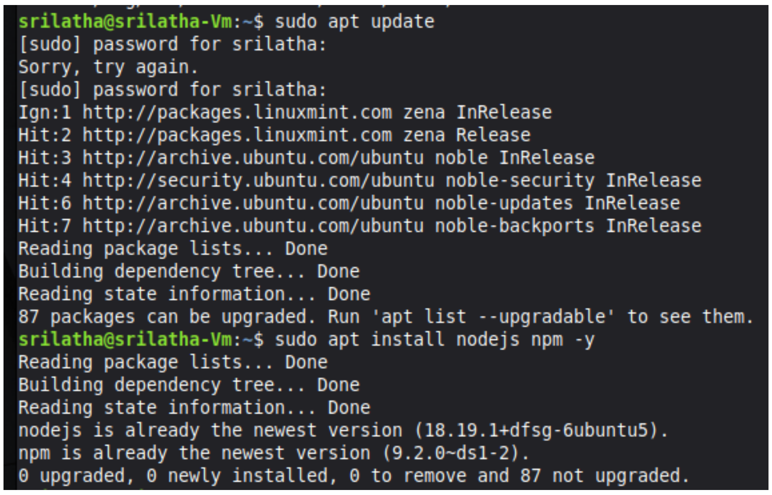
Sudo-→ Super User DO----- Allows a normal user to execute commands with administrator (root) privileges.

Apt→ Advanced Package Tool

Node.j→ runtime environment that allows execution of JavaScript code on the server side

Npm→ Node Package Manager-y “yes” to all installation prompts

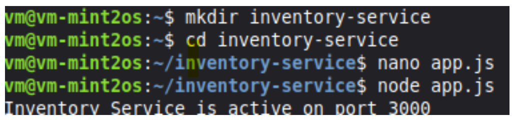
This confirms that the system environment was already correctly configured for running Node.js-based microservice applications.



```
srilatha@srilatha-Vm:~$ sudo apt update
[sudo] password for srilatha:
Sorry, try again.
[sudo] password for srilatha:
Ign:1 http://packages.linuxmint.com zena InRelease
Hit:2 http://packages.linuxmint.com zena Release
Hit:3 http://archive.ubuntu.com/ubuntu noble InRelease
Hit:4 http://security.ubuntu.com/ubuntu noble-security InRelease
Hit:6 http://archive.ubuntu.com/ubuntu noble-updates InRelease
Hit:7 http://archive.ubuntu.com/ubuntu noble-backports InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
87 packages can be upgraded. Run 'apt list --upgradable' to see them.
srilatha@srilatha-Vm:~$ sudo apt install nodejs npm -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
nodejs is already the newest version (18.19.1+dfsg-6ubuntu5).
npm is already the newest version (9.2.0~ds1-2).
0 upgraded, 0 newly installed, 0 to remove and 87 not upgraded.
```

Step:2

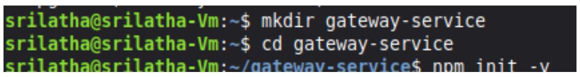
- A second virtual machine (VM2) was used to verify application setup and execution.
- Node.js and npm were installed using the APT package manager.
- A project directory named inventory-service was created.
- An application file app.js was created using a text editor.
- The Node.js application was executed using the node app.js command.
- The service started successfully and listened on port **3000**.
- This confirms that the Node.js runtime environment is properly configured.



```
vm@vm-mint2os:~$ mkdir inventory-service
vm@vm-mint2os:~$ cd inventory-service
vm@vm-mint2os:~/inventory-service$ nano app.js
vm@vm-mint2os:~/inventory-service$ node app.js
Inventory Service is active on port 3000
```

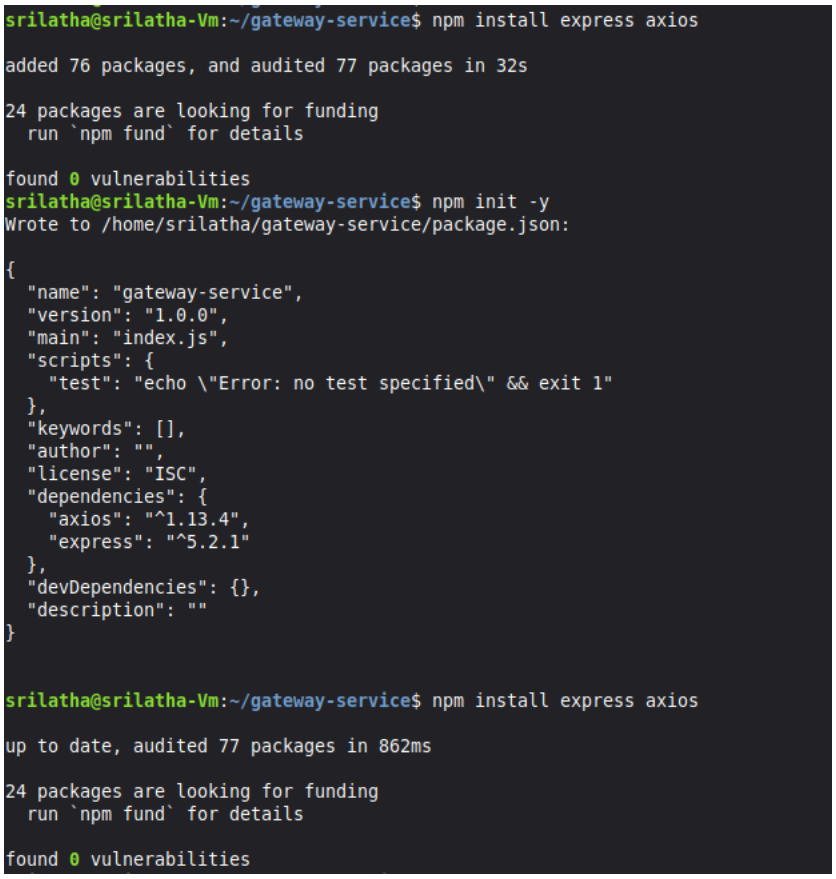
Step:3

- **mkdir gateway-service**: To create a **dedicated workspace** and ensure the Gateway's source code is isolated from other system files.
- **cd gateway-service**: To move the terminal's focus into the project directory so all subsequent commands and files stay **contained within the project**.
- **npm init -y**: To instantly generate a **package.json file**, which acts as the project's "ID card" and tracks all the external libraries (dependencies) the service will need to run.



```
srilatha@srilatha-Vm:~$ mkdir gateway-service
srilatha@srilatha-Vm:~$ cd gateway-service
srilatha@srilatha-Vm:~/gateway-service$ npm init -y
```

Step:4



```
srilatha@srilatha-Vm:~/gateway-service$ npm install express axios

added 76 packages, and audited 77 packages in 32s

24 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
srilatha@srilatha-Vm:~/gateway-service$ npm init -y
Wrote to /home/srilatha/gateway-service/package.json:

{
  "name": "gateway-service",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "axios": "^1.13.4",
    "express": "^5.2.1"
  },
  "devDependencies": {},
  "description": ""
}


srilatha@srilatha-Vm:~/gateway-service$ npm install express axios

up to date, audited 77 packages in 862ms

24 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

- **npm install express axios**: To download and integrate the core libraries required for the service. **Express** is used to build the web server and handle API routing, while **Axios** is used to make HTTP requests to other microservices.

- **Verification (npm init -y)**: To confirm that the dependencies were correctly registered in the package.json file. This ensures the project manifest reflects the current environment, showing **axios (^1.13.4)** and **express (^5.2.1)** under the "dependencies" section.
- **Security & Integrity Audit**: The command automatically triggers a security audit (showing **0 vulnerabilities**), ensuring the installed packages are safe and stable for development.

Step:5 In Virtual Machine 1 API Gateway Service Implementation

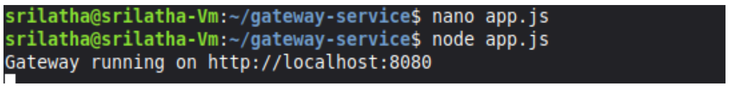
In this **Overview** phase, a **Node.js** application was developed to act as an **API Gateway**. This service serves as the single entry point for the system, routing requests to the appropriate backend services.

Environment & Execution

- **Editor:** The nano text editor was used to write and modify the app.js source code.
- **Runtime:** Node.js.
- **Command to Start:** node app.js.
- **Active Port:** The gateway is configured to listen on **Port 8080**.

Implementation Details

- **Directory:** The service is located in the ~/gateway-service directory.
- **Initialization:** Upon execution, the console confirms the status with the message: Gateway running on http://localhost:8080.
- **Functionality:** This gateway acts as a bridge, similar to the **Bridged Adapter** concept used in the VM setup, but at the application layer for handling HTTP traffic.



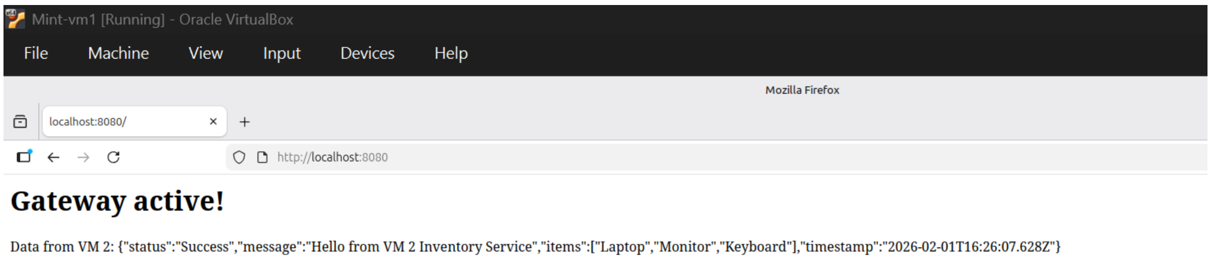
```
srilatha@srilatha-Vm:~/gateway-service$ nano app.js
srilatha@srilatha-Vm:~/gateway-service$ node app.js
Gateway running on http://localhost:8080
```

Gate way app.js

```
const express = require('express');

const axios = require('axios');

const app = express();


app.get('/', async (req, res) => {

   try {

      // This connects to the IP of VM 2

      const response = await axios.get('http://192.168.29.63:3000/api/inventory');

      res.send(`<h1>Gateway active!</h1><p>Data from VM 2:
${JSON.stringify(response.data)}</p>`);
```

```
    } catch (err) {
        res.status(500).send("Error: Cannot reach VM 2. Ensure VM 2 is running node app.js");
    }
});
```

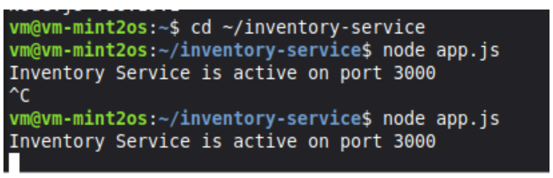
VM 1 :To check the service on localhost on Gateway


Mint-vm1 [Running] - Oracle VirtualBox
File Machine View Input Devices Help
Mozilla Firefox
localhost:8080/
http://localhost:8080

Gateway active!

Data from VM 2: {"status":"Success","message":"Hello from VM 2 Inventory Service","items":["Laptop","Monitor","Keyboard"],"timestamp":"2026-02-01T16:26:07.628Z"}

Step:6 **in virtual machine 2 we are making inventory report**



```
vm@vm-mint2os:~$ cd ~/inventory-service
vm@vm-mint2os:~/inventory-service$ node app.js
Inventory Service is active on port 3000
^C
vm@vm-mint2os:~/inventory-service$ node app.js
Inventory Service is active on port 3000
```

. Directory Navigation

- **cd ~/inventory-service**: Navigates into the specific microservice folder to ensure commands are executed within the correct **isolated environment**.
- **We have used this code**

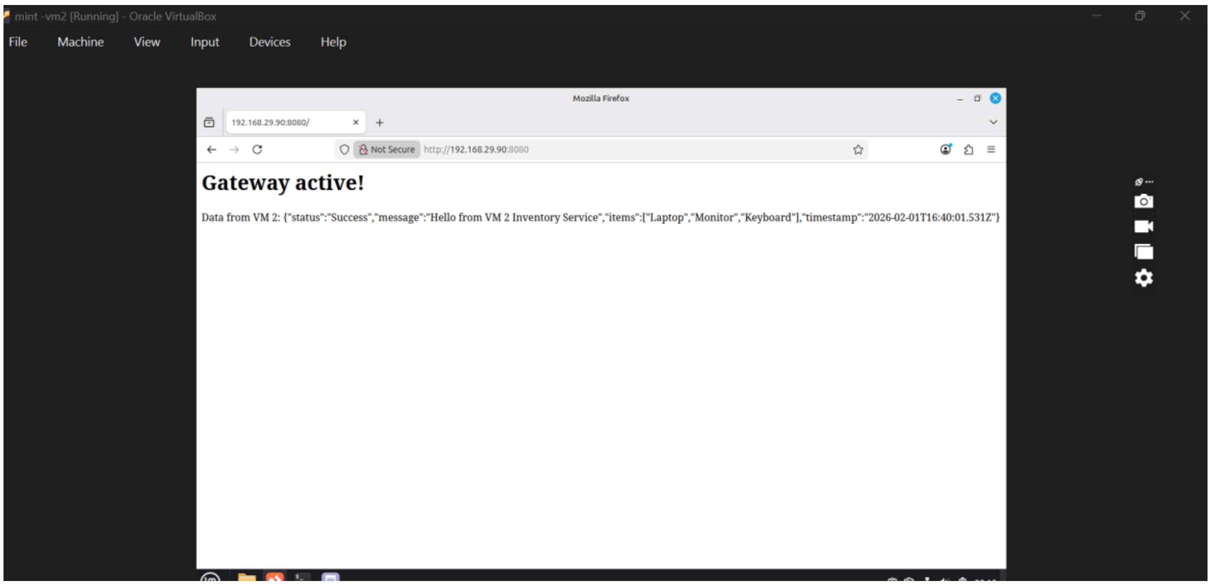
```
const express = require('express');

const app = express();

const PORT = 3000;
```

```
// This is the data your microservice will provide
app.get('/api/inventory', (req, res) => {
res.json({
status: "Success",
message: "Hello from VM 2 Inventory Service",
items: ["Laptop", "Monitor", "Keyboard"],
timestamp: new Date()
});
});
 app.listen(PORT, '0.0.0.0', () => {
console.log(`Inventory Service is active on port ${PORT}`);
});
```

. Service Execution

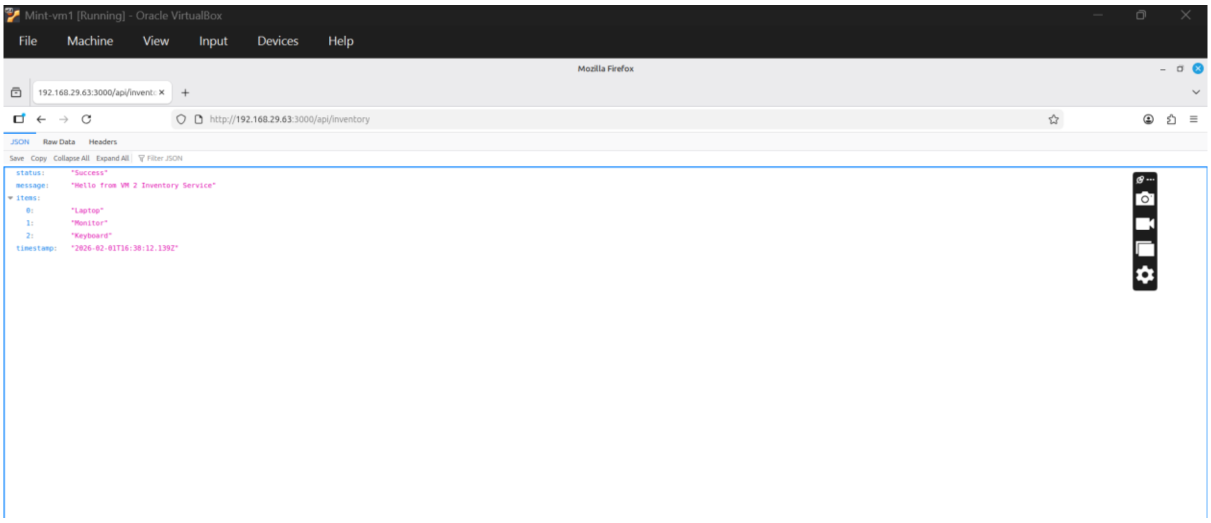
- **node app.js**: Launches the **Node.js runtime** for the Inventory Service, making it live and "active on port 3000" to listen for incoming data.

To check the Gateway service which is running on VM1 (192.168.29.90) from VM2 (192.168.29.63)



Gateway active!

Data from VM 2: {"status":"Success","message":"Hello from VM 2 Inventory Service","items":["Laptop","Monitor","Keyboard"],"timestamp":"2026-02-01T16:40:01.531Z"}


Mint-vm1 [Running] - Oracle VirtualBox
File Machine View Input Devices Help
Mozilla Firefox
192.168.29.63:3000/api/invent
JSON Raw Data Headers
Save Copy Collapse All Expand All Filter JSON

http://192.168.29.63:3000/api/inventory

```
status:     "Success"
message:    "Hello from VM 2 Inventory Service"
▾ items:
   0:       "Laptop"
   1:       "Monitor"
   2:       "Keyboard"
timestamp:  "2026-02-01T16:38:12.139Z"
```

Conclusion:-

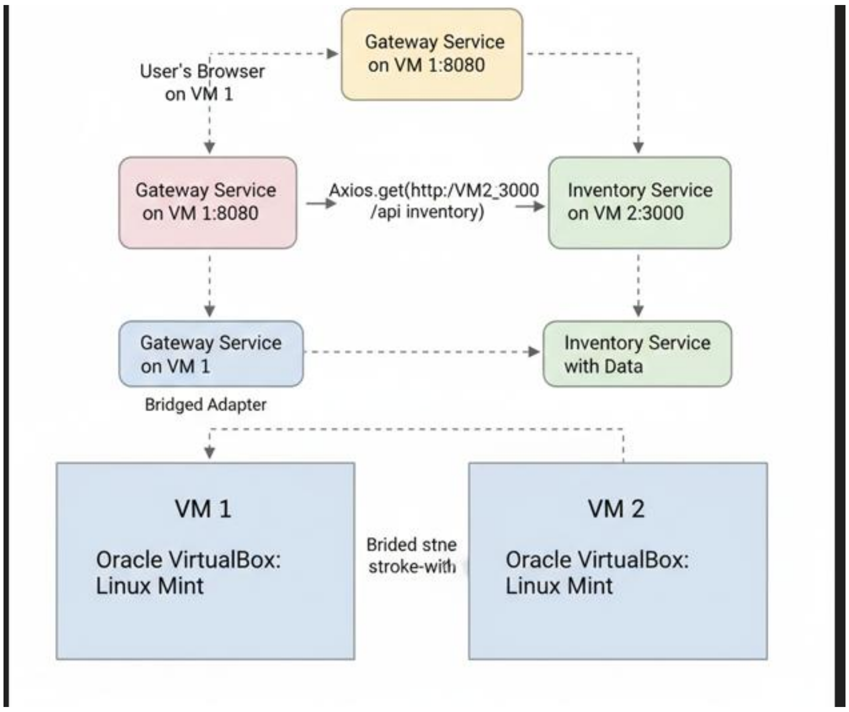
- **Service Creation**: Developed a Node.js API Gateway located in the ~/gateway-service directory.
- **Code Management**: Used the nano editor to configure app.js, ensuring the gateway handles routing between services.
- **Service Launch**: Executed the service using the node app.js command, which successfully initialized the server on **Port 8080**.

Cross-VM Service Testing

- **Gateway Access**: Accessed the API Gateway through the Firefox browser on VM1 via http://localhost:8080.
- **Success Confirmation**: The browser displayed "Gateway active!", confirming the primary service was reachable.
- **Data Integration**: Successfully retrieved data from the **Inventory Service** hosted on **VM2**.
- **Payload Verification**: The gateway successfully pulled a JSON object containing a success message and an item list (Laptop, Monitor, Keyboard) from the second virtual machine.
- **Network Integrity**: The Bridged Adapter configuration successfully allowed VM1 and VM2 to communicate as if they were physical hardware on the same local network.
- **Architectural Success**: Demonstrated a functional Microservices architecture where an API Gateway on one VM successfully fetched and displayed data from an Inventory Service on a separate VM.
- **Performance**: Communication was verified to be stable with no packet loss during the initial handshake and low latency during data retrieval.

Architecture Design:

Diagram showing the connection of VMs and their roles in hosting the microservice application.



Here is a short, point-by-point explanation of your project architecture for your report and video.

Project Architecture Overview

- **Infrastructure**: Two Linux Mint Virtual Machines (VM 1 and VM 2) running on Oracle VirtualBox.
- **Networking**: Both VMs are connected via a **Bridged Adapter**, allowing VM 1 to talk directly to VM 2 using its local IP address (192.168.29.63).
- **No Database (SQL-less)**: Instead of a database, the data is stored as a hardcoded **JSON array** directly inside the Inventory Service code for simplicity and speed.

Component Roles

- **Inventory Service (VM 2 / Backend)**:
  - **Port**: 3000.
  - **Role**: Acts as the "Data Provider".
  - **Function**: Listens for requests and sends back an array of items (Laptop, Monitor, Keyboard) as JSON.
- **Gateway Service (VM 1 / Entry Point)**:
  - **Port**: 8080.

   - **Role**: Acts as the "Aggregator" or "Bridge".
   - **Function**: Receives requests from the user’s browser, fetches data from VM 2 using the **Axios** library, and displays the final result.

Step-by-Step Data Flow

1. **User Access**: The user opens Firefox on VM 1 and goes to http://localhost:8080.
2. **Request Trigger**: The Gateway (VM 1) triggers an **Axios GET request** to VM 2's IP: http://192.168.29.63:3000/api/inventory.
3. **Data Transfer**: VM 2 processes the request and sends the inventory data back across the network bridge.
4. **Response**: The Gateway receives the data and renders the "Gateway active!" message along with the inventory list in the browser.

   Link : https://github.com/srilatha-0008/Vcc-microservices/commit/f198dc3ed100e20f5f458ec6c2b5df6a66060203