Implementation of Microservices using Virtual machines

Introduction

The objective of this experiment was to create multiple virtual machines, configure networking between them, and deploy a simple microservice-based application that communicates across the virtual machines. This setup demonstrates the fundamentals of virtualization, inter-VM networking, and microservice deployment.

Since the host system used was a **MacBook with Apple Silicon (M1)**, **UTM** was used as the virtualization platform.

Tools and Environment

- **Host Machine:** MacBook (Apple Silicon M1)
- **Virtualization Tool:** UTM (VirtualBox alternative for Apple Silicon)
- **Guest Operating System:** Ubuntu Server 22.04 LTS (ARM64)
- **Number of VMs:** 2
- **Backend Technology:** Python Flask (REST API)
- **Client Tool:** curl
- **Network Mode:** Shared Network (NAT)

Step-by-Step Implementation

Step 1: Installation of Virtualization Tool

1. UTM was installed on macOS from the official website and it provides virtualization support compatible with Apple Silicon processors.
2. Ubuntu Server 22.04 ARM64 ISO was downloaded for use as the guest OS.

Step 2: Creation of Virtual Machines

Two virtual machines were created using UTM with the following configuration:

Common Configuration for Both VMs:

- OS: Ubuntu Server 22.04 LTS (ARM64)
- RAM: 2 GB
- CPU: 2 cores
- Storage: 20 GB
- Network: Shared Network (NAT)

VM Roles:

- **VM-1:** Backend Microservice (Flask REST API)
- **VM-2:** Client VM (API consumer)

Ubuntu Server was installed on both VMs using default settings, and SSH access was enabled during installation.

Step 3: Network Configuration

- Both VMs were configured using **UTM Shared Network**.

- IP addresses were assigned automatically using **DHCP**.
- No static IP configuration was required.
- Connectivity was verified using the ping command from VM-2 to VM-1.

This confirmed that both VMs were connected to the same virtual network and could communicate with each other.

Step 4: Deployment of Microservice Application (VM-1)

A simple REST API was deployed on VM-1 using **Python Flask**.

Steps performed on VM-1:

1. Installed Python package manager:

   a. sudo apt update
   b sudo apt install -y python3-pip

2. Installed Flask:

   a. pip3 install flask

3. Created a Flask application with two endpoints:

   a. / – returns a greeting message
   b. /health – returns service health status

4. The application was configured to listen on 0.0.0.0 so that it could accept requests from other VMs.

5. The service was started using:

   a. python3 app.py

Step 5: Client Interaction (VM-2)

VM-2 acted as a client that consumed the backend service.

Steps performed on VM-2:

1. Verified network connectivity using ping.

2. Accessed the backend REST API using:

   a. curl http://<VM1_IP>:5000
   b. curl http://<VM1_IP>:5000/health

3. Successful responses confirmed inter-VM microservice communication.

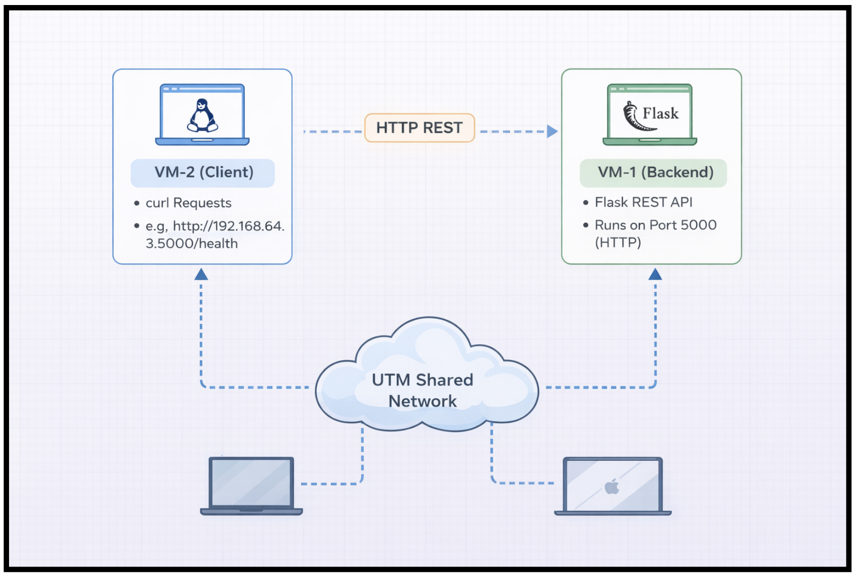
Observations

- When the backend service was stopped, client requests resulted in "connection refused," demonstrating service dependency.
- Restarting the backend service restored connectivity without any configuration changes.
- This validated correct networking and service behavior.

Conclusion

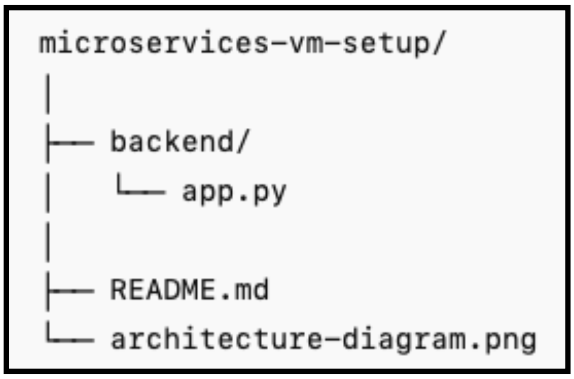
The experiment successfully demonstrated the creation of multiple virtual machines, configuration of virtual networking, and deployment of a microservice-based application across VMs. The backend service hosted on VM-1 was successfully accessed from VM-2, fulfilling the objective of inter-VM microservice communication.

Architecture Diagram



- **VM-1:** Hosts the Flask-based REST microservice
- **VM-2:** Acts as a client consuming the REST API

Repository Structure



```
microservices-vm-setup/
│
├── backend/
│   └── app.py
│
├── README.md
└── architecture-diagram.png
```

GitHub Repository Link

https://github.com/GopiHaribabu/VCC_Assignment1_Gopi_Haribabu_M25AI2092_microservices-vm-setup.git