Deploying a Microservice-Based Application on Multiple connected VM’s

Step 1: Install VirtualBox and extension pack(for clipboard sync and Drag’n’Drop option for ease)

1. Downloaded **VirtualBox(TYPE || Hypervisor)** and its extension pack from the official website
2. Installed on the **HOST OS : WIN 11**

Step 2: download Guest OS ISO file

1. Went on the ubuntu site and downloaded the following two ISO images.
    - Ubuntu Desktop : To view the rendered application in action on browser
    - Ubuntu server : To run the backend microservices like AUTH , CRUD based simple REST APIs

Step 3: Created VMS

Created Two VMs with Different Operating Systems

1. **Ubuntu Desktop System**
    - **RAM**: 4 GB / 8 (Minimum requirement to boot up the OS: 4 GB)
    - **CPU Logical Cores**: 2 / 8
    - **Storage (ROM)**: 25 GB (VDI - Virtual Disk Image)
2. **Ubuntu Server System**
    - **RAM**: 2 GB / 8 (Minimum requirement to boot up the OS: 1 GB)
    - **CPU Logical Cores**: 2 / 8
    - **Storage (ROM)**: 25 GB (VDI - Virtual Disk Image)

Step 4: Network Configuration

To enable communication between both Virtual Machines (VMs) while ensuring internet access, a **NAT Network** configuration was created in VirtualBox. This setup assigns unique IP addresses to each VM, allowing them to interact with each other and access external networks.

Configuration Details:

- **Network Type:** NAT Network
- **IP Assignment:** Automatic (DHCP)
- **VM Communication:** Enabled (Each VM can ping the other)
- **Internet Access:** Enabled (Able to ping Google's public server at `8.8.8.8`)

Reason for Choosing NAT Network:

The **NAT Network** mode in VirtualBox was selected as it provides the following advantages:

1. **Internal Communication:** Both VMs are assigned unique IP addresses within a private network, enabling seamless interaction.
2. **Internet Access:** The VMs can connect to external networks via the host machine's internet connection.
3. **Simplified Setup:** The built-in DHCP server automatically assigns IP addresses, reducing manual configuration efforts.
4. **Security:** The VMs are isolated from the host machine's network while still maintaining external connectivity.

Reference Resource:

To understand the different network configurations and select the best option, the following resource was used:
🔗 **VirtualBox Forums – "VirtualBox Networks: In Pictures"**
https://forums.virtualbox.org/viewtopic.php?t=96608

This guide provides a clear explanation of different networking modes in VirtualBox, including NAT Network, and how it facilitates both inter-VM communication and internet access.

Using this configuration, successful connectivity was confirmed by:

- **Pinging the other VM** within the network.
- **Pinging Google's public DNS server (`8.8.8.8`)**, verifying external internet access.

This successful setup ensures that the VMs are correctly networked, making them ready for deploying the microservice-based application

After setup **IP ADDR of Ubuntu Desktop : `198.168.100.3` , IP ADDR of Ubuntu server: `198.168.100.4`**

Step 5: Deploying the Microservice-Based Application

The microservice-based application consists of a **frontend** (hosted on the Ubuntu-Windows VM) and a **backend** (hosted on the Ubuntu-Server VM). The backend connects to **MongoDB Atlas**, a cloud database, due to resource constraints on the Ubuntu-Server VM.

Frontend Deployment (Ubuntu-Windows VM)

Installation and Setup:

- **Install Node.js and npm:**
  ```
  sudo apt update
  sudo apt install nodejs npm -y
  git clone https://github.com/vekariyasagar54/VCC-01.git
  ```
- **Navigate to the project folder and install dependencies:**
  ```
  cd VCC-01
  npm install
  ```
- **Start the frontend application:**
  ```
  npm run dev
  ```

Backend Deployment (Ubuntu-Server VM)

The backend of the application is deployed on the **Ubuntu-Server VM** using Node.js and Express.js.

Installation and Setup:

- **Install Node.js and npm:**
  ```
  sudo apt update
  Sudo apt install nodejs npm -y
  ```
- **Clone the backend repository:**
  ```
  git clone https://github.com/vekariyasagar54/VCC-01.git
  ```
- **Navigate to the backend directory and install dependencies:**
  ```
  cd VCC-01/backend
  npm install
  ```

Connect to MongoDB Atlas:

- Create an account on **MongoDB Atlas**.
- Set up a cluster and obtain the **MongoDB connection string**.
- Update `db.js` with the **MongoDB connection string**.

Start the backend application:

```
node index.js
```

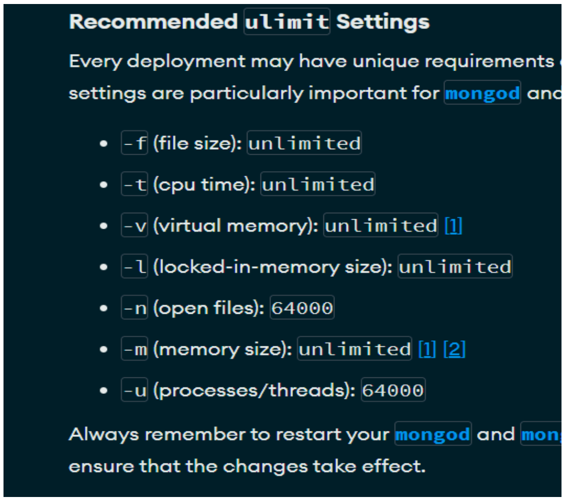
Troubleshooting on MongoDB Was Not Deployed on the Ubuntu-Server VM

Initially, an attempt was made to install MongoDB locally on the **Ubuntu-Server VM** following the **MongoDB [installation guide](#)**. However, MongoDB services (`mongod` and `mongos`) failed to run due to system resource limitations.

Key Issues Identified:

1. **RAM Limitation:**
   - MongoDB requires higher system resources for optimal performance.
   - The **Ubuntu-Server VM** has **only 2 GB RAM**, whereas MongoDB generally requires more.
2. **Thread Limit Constraint:**
   - MongoDB requires `ulimit` settings that allow up to **64,000 threads** for stable operation.
   - The **Ubuntu-Server VM** only supports **around 18,000 threads**, making it **impossible to run `mongod` as a service**.

Solution: Due to these constraints, **MongoDB Atlas** (a cloud-based database service) was used instead of deploying MongoDB locally. This ensures stable database performance without overloading the limited resources of the Ubuntu-Server VM.



Recommended `ulimit` Settings

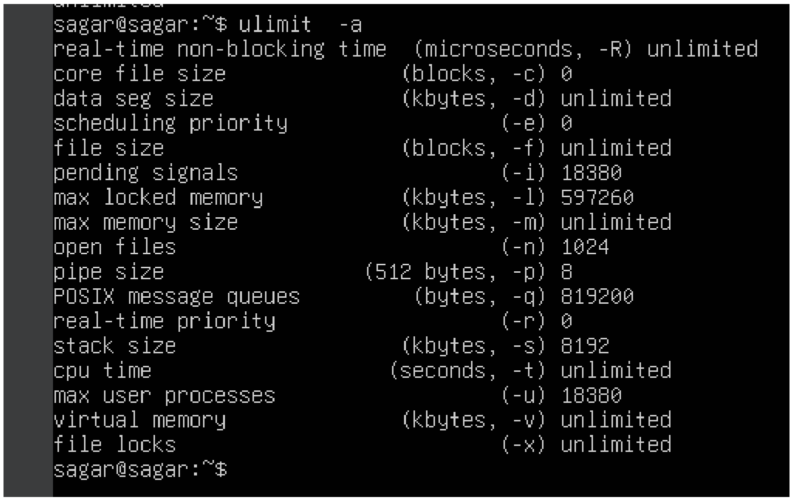
Every deployment may have unique requirements

settings are particularly important for `mongod` and

- `-f` (file size): `unlimited`
- `-t` (cpu time): `unlimited`
- `-v` (virtual memory): `unlimited` [1]
- `-l` (locked-in-memory size): `unlimited`
- `-n` (open files): `64000`
- `-m` (memory size): `unlimited` [1] [2]
- `-u` (processes/threads): `64000`

Always remember to restart your `mongod` and `mon`

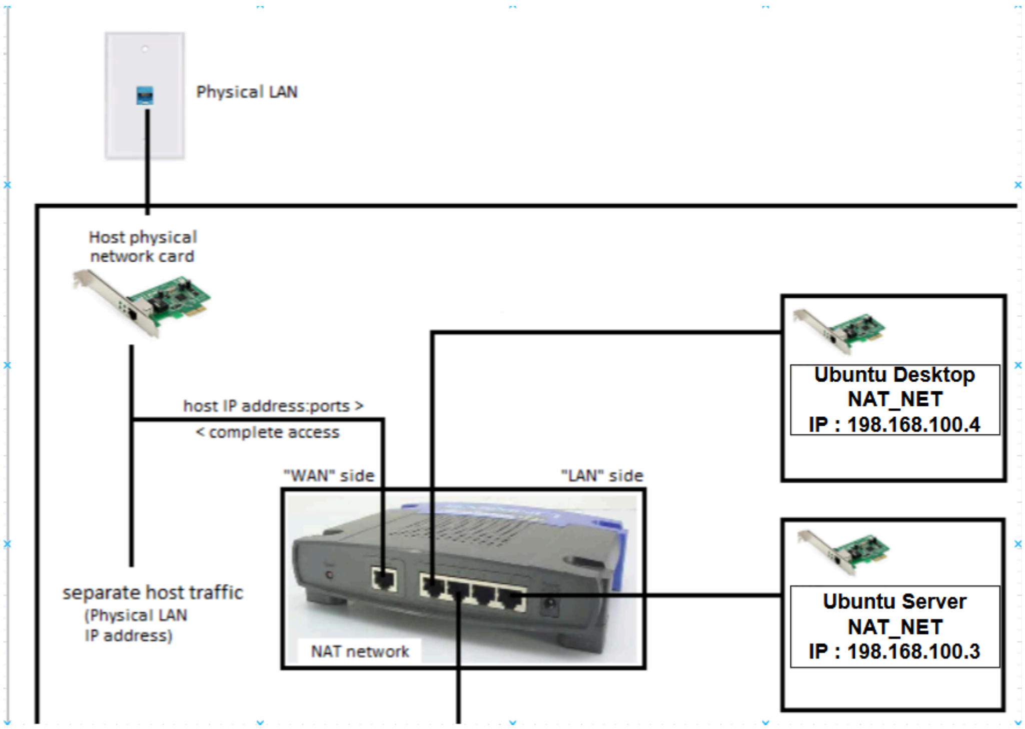
ensure that the changes take effect.



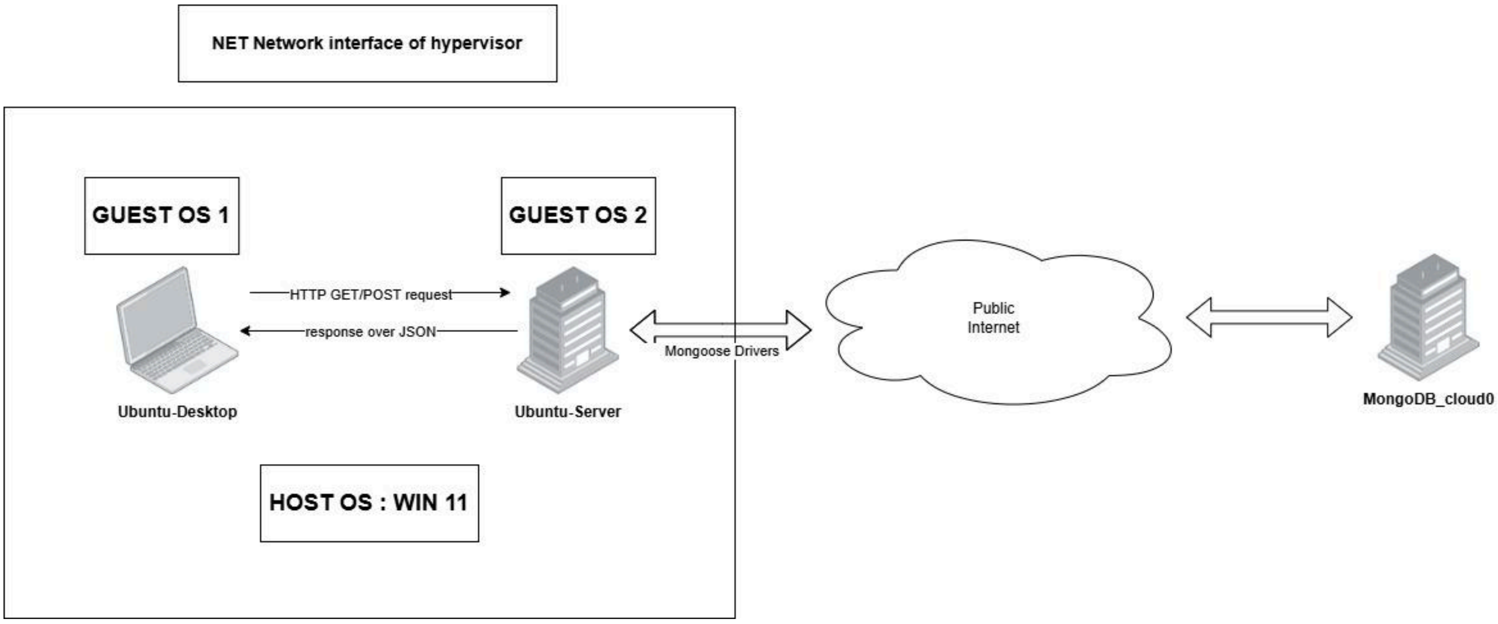
```
sagar@sagar:~$ ulimit  -a
real-time non-blocking time  (microseconds, -R) unlimited
core file size              (blocks, -c) 0
data seg size               (kbytes, -d) unlimited
scheduling priority                 (-e) 0
file size                   (blocks, -f) unlimited
pending signals                     (-i) 18380
max locked memory           (kbytes, -l) 597260
max memory size             (kbytes, -m) unlimited
open files                          (-n) 1024
pipe size                (512 bytes, -p) 8
POSIX message queues         (bytes, -q) 819200
real-time priority                  (-r) 0
stack size                  (kbytes, -s) 8192
cpu time                   (seconds, -t) unlimited
max user processes                  (-u) 18380
virtual memory              (kbytes, -v) unlimited
file locks                          (-x) unlimited
sagar@sagar:~$
```

Architecture Design

Network Architecture Diagram



Application Architecture Diagram



Resources

Github Repo Link : https://github.com/vekariyasagar54/VCC-01

Video demo Link;
https://drive.google.com/file/d/1B8WFOr5uyIRqnJcZQ3Ug3PN4tRI3hWTZ/view?usp=sharing