Project 1 &2

Multi-Tier Java Application Using Vagrant & Virtual Machines**

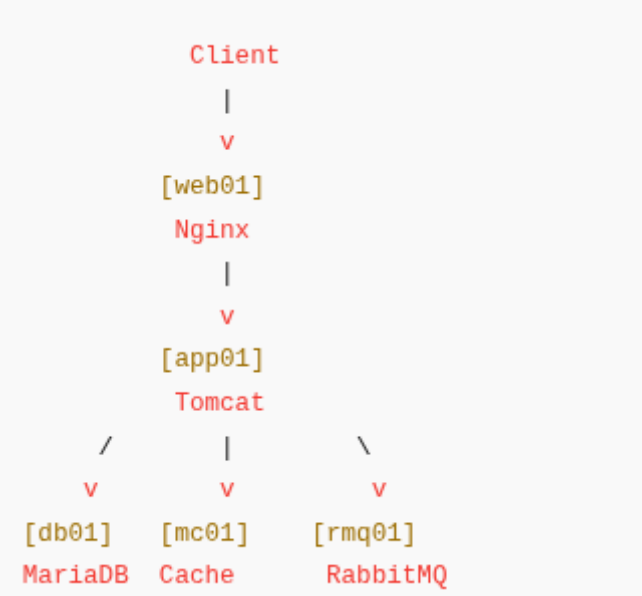
1. What This Project Is

This project how to build a **complete on-prem enterprise environment** locally using only:

- **Vagrant** (Infrastructure as Code)
- **VirtualBox** (Hypervisor)
- **Shell scripts** (Provisioning)
- **Java Web App (WAR)**
- **Tomcat, Nginx, MariaDB, Memcached, RabbitMQ**

2. What You Will Build

You will build **five VMs**, each VM running one service only.



Each VM is isolated, has a fixed IP, and has a dedicated role.

This is how production environments normally look.

3. Technologies You Will Learn

Infrastructure

- Vagrant
- VirtualBox private networking
- Provisioning with shell scripts

Web Layer

- Nginx reverse proxy

App Layer

- Apache Tomcat
- Java WAR deployment
- Maven build pipeline

Backend Services

- MariaDB database
- Memcached caching
- RabbitMQ messaging

4. The Five Virtual Machines

VM	IP	Purpose	OS	RAM	Service
web01	192.168.56.11	Web Tier	Ubuntu 22.04	800MB	Nginx
app01	192.168.56.12	Application Tier	CentOS Stream 9	4200MB	Tomcat
rmq01	192.168.56.13	Message Queue	CentOS Stream 9	600MB	RabbitMQ
mc01	192.168.56.14	Cache Tier	CentOS Stream 9	900MB	Memcached
db01	192.168.56.15	Database Tier	CentOS Stream 9	2048MB	MariaDB

6. How Everything Works Together

Step 1: User Opens Browser

User sends request → http://192.168.56.11

Step 2: Nginx Receives the Request

Nginx → checks config → forwards request to Tomcat server at:

`http://192.168.56.12:8080`

Step 3: Tomcat Runs the Java Application

The WAR file is already deployed.

Java app needs:

- DB (MariaDB)
- Cache (Memcached)
- Queue (RabbitMQ)

So the app connects using the fixed IPs.

Step 4: Backend Interaction

- **Reads/Writes** → MariaDB
- **Caching** → Memcached
- **Background Jobs** → RabbitMQ

Step 5: Response Goes Back to User

Tomcat → Nginx → Browser.

This is a real enterprise flow.

HINTS: How to Connect Your Java Backend to All Services

Use these hints to correctly configure your Java backend so it communicates with:

- Database (MariaDB)
- Cache (Memcached)
- Queue (RabbitMQ)
- Tomcat (Deployment)
- Nginx (Reverse Proxy)

All connections happen inside:

src/main/resources/application.properties

**HINT 1

Configure the Database (MariaDB)**

Use the database VM IP:

`192.168.56.15`

Add these properties:

spring.datasource.url=jdbc:mysql://192.168.56.15:3306/appdb
spring.datasource.username=appuser
spring.datasource.password=app123
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true

Make sure your DB name, user, and password match the ones created in `mariadb.sh`.

**HINT 2

Configure Memcached (Cache Layer)**

Use the cache VM IP:

192.168.56.14
memcached.servers=192.168.56.14:11211
memcached.protocol=BINARY
memcached.opTimeout=5000
Hint:
Memcached always listens on port **11211**.

**HINT 3

Configure RabbitMQ (Message Queue)**

Use the queue VM IP:

`192.168.56.13`

Add:

```
spring.rabbitmq.host=192.168.56.13
spring.rabbitmq.port=5672
spring.rabbitmq.username=guest
spring.rabbitmq.password=guest
```

```
spring.rabbitmq.virtual-host=/
```

Hint:
Keep username/password as `guest / guest` unless you configured RabbitMQ users manually.