Deployment Documentation:

1. Installation and Setuping up VMs:
   1. Introduction: This documentation provides a comprehensive guide to deploying a Retrieval-Augmented Generation (RAG) application using two Virtual Machines (VMs). The setup includes:
      1. VM without GPU: Hosts an Express server and a FastAPI server, and manages ChromaDB and MongoDB.
      2. VM with GPU: Runs large language models (LLMs) such as LLaMA 3.1 and LLAVA-LLaMA , managed by Ollama.
   2. Prerequisites
      1. Update VM1 and VM2: From here we are going to use the command “sudo apt update” to go ahead and have Ubuntu collect any pending updates.

Now we know that we have collected packages that can be upgraded, do that with the command “sudo apt upgrade” and then typing “Y” when prompted to continue.

* + 1. Ensure the following software is installed on the respective VMs:
* VM without GPU:
  + - * 1. Node.js
        2. Python (via Anaconda)
        3. Docker
        4. ChromaDB (Dockerized)
        5. MongoDB (installed natively, not in Docker)
        6. PM2 (for process management)
* VM with GPU:
  + - * 1. Ollama
  1. Installation Instructions

Step 1: Set Up Anaconda Environment:

1. Install Anaconda:
   * + - 1. Download and install Anaconda.

wget [https://repo.anaconda.com/archive/Anaconda3-<Latest](https://repo.anaconda.com/archive/Anaconda3-%3cLatest)\_Version>-Linux-x86\_64.sh

Latest Version can be found here - <https://repo.anaconda.com/archive/>

sha256sum Anaconda3-<Latest\_Version>-Linux-x86\_64.sh

bash Anaconda3-<Latest\_Version>-Linux-x86\_64.sh

After running the bash command, you’ll be welcomed to the Anaconda setup. However, you must review and agree to its license agreement before the installation. Hit Enter to continue.

Pressing the space bar a few times will bring you to the end of the license agreement, where you can accept the terms. Type in “yes” as highlighted and hit Enter.

After agreeing to the license terms, the following prompt will ask you to input the directory where to install the Anaconda on the Ubuntu system. The default location is the user’s HOME directory on Ubuntu.

It is recommended to have Anaconda installed in this location. Therefore, press Enter to confirm the default location.

* + - * 1. Create and activate the project environment:
* conda create --name <ENV\_NAME> python=3.11
* conda activate <ENV\_NAME>
  + - 1. Install Python Dependencies:
         1. Navigate to the FastAPI server directory and install dependencies:

pip install -r requirements.txt

Step 2: Install Node.js and PM2:

1. Install Node.js:
   1. Download and install Node.js
      * sudo apt install nodejs
2. Install npm Dependencies:
   1. Navigate to the Express server directory and install dependencies:
      * npm install
3. Install PM2:
   1. Install PM2 globally to manage Node.js applications:
      * npm install pm2@latest -g

Step 3: Install MongoDB:

1. Install MongoDB:
   * + - 1. Follow the instructions on this website to install MongoDB:

<https://www.mongodb.com/docs/manual/tutorial/install-mongodb-on-ubuntu/>

* + - * 1. Start and enable MongoDB:

sudo systemctl start mongod

sudo systemctl enable mongod

Step 4: Install Docker:

1. Install Docker:
   * + - 1. Install Docker on VM1;

sudo apt-get install -y docker.io

* + - * 1. Start and enable Docker:

sudo systemctl start docker

sudo systemctl enable docker

Step 5: Install ChromaDB in Docker (VM1):

1. Pull and Run ChromaDB:
   * + - 1. Pull the ChromaDB Docker image and start the container:

docker pull chromadb/chromadb

docker run -d --name chromadb -p 8000:8000 chromadb/chromadb

Step 6: Install Ollama on VM2:

1. curl -fsSL https://ollama.com/install.sh | sh

Using this command Ollama will be installed.

Step 7: Pull LLM models in Ollama (VM2):

1. Pull LLM models:

* ollama pull llama3.1:8b
* ollama pull llava-llama3:8b
  1. SSL Certificate:
     1. Certificate Format: PEM
     2. Required Files:
* “cert.pem” and “key.pem”
  + 1. When requesting SSL certificates following are the requirements:
* Certificate Authority (CA) information
* Certificate Type (DV, OV, EV)
* Supported Formats
* Domain Names and Wildcard Certificates details

After Obtaining the certificate place the certificate in cert folder of express and fastAPI directory.

* 1. There are 2 ways to run the express server and fastAPI. Using service file/pm2 and utilizing nginx or by using docker. Both the methods are listed below:

Method 1: Using Service File and Nginx

* + 1. By Using Service file/PM2
* For Express Server go to the express folder and run the command in terminal “pm2 start app.js -i <workers>”
* Create a Service File for fastAPI by going to the system directory using “cd /etc/system/system” and type “sudo nano fast.service” and paste in the given service file by replacing the directory and environment path and the port which needs to be used.

Example –

[Unit]

Description=Gunicorn instance serve Fast app

After=network.target

[Service]

User=Mayank.Sharma

Group=www-data

WorkingDirectory=/home/Mayank.Sharma/GV\_Test/backend/fast

Environment="PATH=/home/Mayank.Sharma/anaconda3/envs/GV\_Test/bin"

ExecStart=/home/Mayank.Sharma/anaconda3/envs/GV\_Test/bin/gunicorn main:app --workers 3 --worker-class uvicorn.workers.UvicornWorker --bind 0.0.0.0:6677

Restart=on-failure

[Install]

WantedBy=multi-user.target

* + Start and enable the created service.

“sudo systemctl restart fast.service”

“sudo systemctl enable fast.service”

* + 1. Setup Nginx for Express Server
* Edit the default Nginx configuration file for express application.

“sudo nano /etc/nginx/sites-available/default”

* Paste the given Nginx configuration for the express application by replacing the server’s name and set the port as in the app.js file in express directory.

Example –

server {

listen 80 ;

server\_name 20.191.112.232;

location /api {

proxy\_pass https://localhost:8080;

proxy\_http\_version 1.1;

proxy\_set\_header Upgrade $http\_upgrade;

proxy\_set\_header Connection 'upgrade';

proxy\_set\_header Host $host;

proxy\_cache\_bypass $http\_upgrade;

}

* + 1. Setup Nginx for FastAPI
* Edit the default Nginx configuration file for fastAPI.

“sudo nano /etc/nginx/sites-available/default”

* Paste the given Nginx configuration for fastAPI by replacing the server’s name, certificate location and set the port as in the service file of the fastAPI instance.

Example –

server{

listen 443 ssl;

server\_name 20.191.112.232;

ssl\_certificate /home/Mayank.Sharma/GV\_Test/backend/fast/certs/certificate.pem;

ssl\_certificate\_key /home/Mayank.Sharma/GV\_Test/backend/fast/certs/private-key.pem;

ssl\_protocols TLSv1.2 TLSv1.3;

location / {

proxy\_pass http://127.0.0.1:6677;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header X-Forwarded-Proto $scheme;

# Disable buffering to enable streaming

proxy\_buffering off;

proxy\_cache off;

proxy\_request\_buffering off;

}

}

* Test and Reload Nginx
  + “sudo nginx -t”
* If there are no errors, reload NGINX to apply the changes:
  + “sudo systemctl reload nginx”

Now the express server api can be accessed using <https://domain_name.com/api> and fast api can be accessed using <https://domain_name.com/>

Method 2: Using Docker

* Dockerize the Express Server

1. Create a Dockerfile in the Express server directory:

Example –

FROM node

WORKDIR /home/Mayank.Sharma/GV\_Test/backend/app

COPY package\*.json ./

RUN npm install

COPY .env .env

COPY certs /home/Mayank.Sharma/GV\_Test/backend/app

COPY . .

EXPOSE 8080

CMD ["node", "app.js"]

1. Build and run the Docker Image:

* docker build -t express-app .
* docker run -d –network= “host” express-app
* Dockerize the FastAPI Server

1. Create a Dockerfile in the FastAPI server directory:

Example –

FROM python:3.11.9

WORKDIR /home/Mayank.Sharma/GV\_Test/backend/fastapp

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY .env .env

COPY certs /home/Mayank.Sharma/GV\_Test/backend/fastapp/certs

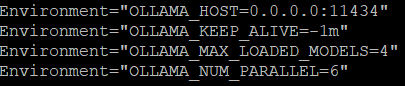
COPY . .

EXPOSE 443

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "443", "--ssl-keyfile", "/home/Mayank.Sharma/GV\_Test/backend/fastapp/certs/private-key.pem", "--ssl-certfile", "/home/Mayank.Sharma/GV\_Test/backend/fastapp/certs/certificate.pem"]

1. Build and run the Docker Image:

* docker build -t fastapi-app .
* docker run -d --network= “host” fastapi-app
  1. IP Allowlist: Make sure the service file of Ollama allow specific IPs to interact with the LLM services (sudo nano /etc/system/system/ollama.service) in VM with GPU.



1. Methodology:

The system is designed to efficiently manage and process user queries by leveraging two VMs with distinct roles:

* VM without GPU: Acts as the frontend-facing server, handling user requests, managing user data, and interacting with backend services like ChromaDB and MongoDB.
* VM with GPU: Specializes in processing computationally intensive tasks, particularly those involving large language models (LLMs) such as LLaMA 3.1 and LLAVA-LLaMA, which are deployed using Ollama.

1. User Query Flow
   1. Initial Request Handling:

* When a user initiates a chat or sends a query via the frontend, the request is first directed to fastAPI.
  1. Query Processing:
* FastAPI calls the required LLM model which are present in VM with GPU(Ollama) to process the query.
  1. LLM Processing on GPU VM:
* The LLMs (LLaMA 3.1, LLAVA-LLaMA) running on the GPU VM receive the query and begin processing it. These models are specifically designed to handle natural language understanding and generation tasks, making them well-suited for providing intelligent responses.
* The processed response, which may include detailed text, is generated by the LLM and is then sent back to the FastAPI server on the non-GPU VM.
  1. Streaming Response:
* The FastAPI server receives the processed response from the LLM and begins streaming this data back to the Frontend.
* The FastAPI server manages this streaming response, ensuring that it is sent back to the frontend in a format that allows for real-time interaction. This may involve chunking the response into smaller parts to provide a smoother user experience.

1. User Management and Experience
   1. User Management: The Express server is the central hub for user management. It handles tasks such as user authentication, session management, and permission checking. The server interacts with MongoDB to store and retrieve user data, including chat histories, preferences, and other metadata.
   2. User Experience Features:

* Bookmarking Chats: Users can bookmark specific chats for future reference. The Express server updates the MongoDB with the bookmark details.
* Deleting/Updating Titles: Users can delete or update chat titles. These actions trigger updates in the MongoDB to reflect the changes.
* Starter Questions and Filters: The Express server can provide starter questions to help users begin their queries. It also handles various filters that users can apply to refine their chat experience.