Reference Architecture Manual: Local Retrieval-Augmented Generation (RAG) System with Weaviate and Hugging Face

---

Project Overview:

Project Name: Local RAG System with Weaviate and Hugging Face
Skill Level: Beginner
Core Technologies: Python, Docker, Weaviate, Hugging Face
Optional Add-ons: CUDA (for GPU acceleration if you have an Nvidia GPU)

Introduction:

What you'll learn:
In this project, you will build a Local Retrieval-Augmented Generation (RAG) system using Weaviate as a vector database for document retrieval and Hugging Face models for generating text responses. This project will teach you how to upload, index, and search documents in Weaviate and use Hugging Face models to generate responses based on the retrieved documents.

Who this project is for:
This project is designed for beginners who want to learn the foundational concepts of RAG systems. It provides clear instructions on setting up the necessary technologies and builds practical experience in creating a useful, real-world tool for document search and generation.

Architecture Overview:

System Components:

1. **Weaviate**
   This is a vector database for storing and retrieving documents based on their semantic meaning.
2. **Hugging Face Models**
   Pre-trained language models that generate text based on documents retrieved from Weaviate.
3. **Python**
   The programming language that ties everything together: from indexing documents to generating text.

4. **Docker (Optional)**
   A containerization platform that allows Weaviate to run independently of your system's other applications.

Basic Flow:

1. Meeting transcripts or other documents are **stored in Weaviate** (your vector database).
2. When you ask a question, **Weaviate retrieves** the most relevant documents.
3. These documents are passed to **Hugging Face's language model**, which **generates a response** based on the documents.
4. The system returns the **response to the user**.

This architecture lays the foundation for more advanced systems by teaching you to integrate Weaviate for document retrieval and Hugging Face for text generation.

Step-by-Step Instructions

Step 1: Setting Up Your Environment

What we're doing here:
You'll install the necessary tools to build this project: Python and Docker. Python will handle the code and Docker will run Weaviate.

Install Python

1. Visit the official Python website: `https://www.python.org/downloads/`
2. Download and install Python 3.9 (or later).
   - **Windows**: Make sure to check the box that says "Add Python to PATH" before clicking install.
   - **Mac/Linux**: Follow the standard installation process for your OS.
3. Open a terminal or command prompt and verify that Python was installed by typing:

```
python --version
```

- You should see something like `Python 3.9.x`.

Install Docker (Optional but recommended for Weaviate)

Docker is optional for this project, but it's highly recommended to run Weaviate in a consistent environment.

1. Visit the Docker website: `https://www.docker.com/products/docker-desktop`
2. Download Docker Desktop and install it.
3. After installation, verify it's running by typing the following in your terminal:

```
docker --version
```

- You should see something like `Docker version 20.10.x`.

Set Up a Project Folder

You'll create a folder on your computer where you'll store all the files for this project.

1. Open your terminal/command prompt.
2. Navigate to the location where you want to store the project.
3. Create a new folder:

```
mkdir local_rag_project

cd local_rag_project
```

Step 2: Running Weaviate with Docker

What we're doing here:
You'll run Weaviate inside a Docker container, allowing you to use Weaviate without installing it directly on your computer.

1. Pull and Run the Weaviate Docker Container:

```
docker pull semitechnologies/weaviate:latest
```

2. Run the Weaviate Docker container with the following command:

```
docker run -d --name weaviate -p 8080:8080 -e QUERY_DEFAULTS_LIMIT=100
-e AUTHENTICATION_ANONYMOUS_ACCESS_ENABLED=true
semitechnologies/weaviate:latest
```

- This command starts Weaviate on your local machine, making it accessible at `http://localhost:8080`.
- Check if Weaviate is running by opening a browser and navigating to: `http://localhost:8080/v1/schema`.

Step 3: Setting Up Python and Installing Required Libraries

What we're doing here:
You'll create a virtual Python environment and install the necessary libraries for this project, including Weaviate's Python client and Hugging Face's Transformers

1. In your project folder (`local_rag_project`), create a virtual environment:

```
python -m venv venv
```

- This creates a virtual environment named `venv`.
2. Activate the virtual environment:
    - On Windows:

```
venv\Scripts\activate
```

- On Mac/Linux:

```
source venv/bin/activate
```

3. Install the necessary Python libraries by creating a file named `requirements.txt` in the project folder with the following contents:

```
weaviate-client

sentence-transformers

transformers
```

4. Install the libraries by running:

```
pip install -r requirements.txt
```

Step 4: Defining the Schema and Uploading Meeting Transcripts

What we're doing here:
In this step, you'll define a schema in Weaviate for storing your meeting transcripts and upload them.

Define Your Schema:

1. Create a Python file named `define_schema.py` in the `local_rag_project` folder.
2. Copy and paste this Python code into `define_schema.py`:

```
import weaviate


client = weaviate.Client("http://localhost:8080")
```

```
schema = {
    "classes": [
        {
            "class": "Transcript",
            "properties": [
                {"name": "date", "dataType": ["date"]},
                {"name": "content", "dataType": ["text"]}
            ]
        }
    ]
}
client.schema.create(schema)
```

3. Run the script to create the schema:

```
python define_schema.py
```

Upload Your Transcripts:

1. Create a Python file named `upload_transcripts.py`.
2. Add the following code to upload a transcript:

```
import weaviate
```

```
client = weaviate.Client("http://localhost:8080")


transcript = {
    "date": "2024-01-15",
    "content": "We discussed project milestones and next steps."
}


client.data_object.create(transcript, "Transcript")
```

3. Run the script to upload your transcript:

```
python upload_transcripts.py
```

Step 5: Vectorizing Your Transcripts Locally

What we're doing here:
You'll generate vector embeddings for your meeting transcripts to enable efficient retrieval in Weaviate.

Install Sentence Transformers:

Install the Sentence Transformers library if you haven't already:

```
pip install sentence-transformers
```

Generate Vector Embeddings:

1. Create a Python file named `vectorize_transcripts.py`.
2. Add the following code to generate vector embeddings:

```
from sentence_transformers import SentenceTransformer

import weaviate


client = weaviate.Client("http://localhost:8080")

model = SentenceTransformer('all-MiniLM-L6-v2')


content = "Meeting transcript content to be vectorized."

vector = model.encode(content)


client.data_object.create(

    {"date": "2024-01-15", "content": content},

    "Transcript", vector=vector

)
```

3. Run the script to vectorize and store the transcript:

```
python vectorize_transcripts.py
```

Step 6: Integrating Hugging Face Models for Document Retrieval

RAG_Local_HF_Weaviate_v2

What we're doing here:
You'll use a pre-trained Hugging Face model to generate responses based on the documents retrieved from Weaviate.

Install Hugging Face Transformers:

```
pip install transformers
```

Generate Responses Using Hugging Face:

1. Create a Python file named `rag_system.py`.
2. Add the following code to search documents and generate a response:

```
from weaviate import Client

from transformers import pipeline

from sentence_transformers import SentenceTransformer


client = Client("http://localhost:8080")

model = SentenceTransformer('all-MiniLM-L6-v2')

generator = pipeline('text-generation',
model='EleutherAI/gpt-neo-125M')


# Function to search documents in Weaviate

def search_documents(query):

    query_vector = model.encode(query)

    results = client.query.get("Transcript",
["content"]).with_near_vector({
```

```
        "vector": query_vector,
    }).do()
    return results['data']['Get']['Transcript']


# Function to generate an answer
def generate_answer(documents, query):
    context = " ".join([doc['content'] for doc in documents])
    prompt = f"Using the following transcript: {context}. Answer the
question: {query}"
    return generator(prompt, max_length=150,
num_return_sequences=1)[0]['generated_text']


if __name__ == "__main__":
    query = input("Enter your question: ")
    documents = search_documents(query)
    if documents:
        answer = generate_answer(documents, query)
        print(f"Generated Answer: {answer}")
    else:
        print("No relevant documents found.")
```

3. Run the system:

```
python rag_system.py
```

Ask a question based on the documents you uploaded, and the Hugging Face model will generate a response based on the retrieved transcript.

Conclusion:

By following this guide, you have built a fully functional local Retrieval-Augmented Generation (RAG) system using Weaviate and Hugging Face models. You can now retrieve and generate responses based on your meeting transcripts, laying the foundation for more advanced RAG systems in the future.