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

---

Project Overview:

Project Name: Local RAG System with Hugging Face and Elasticsearch
Skill Level: Beginner to Intermediate
Core Technologies: Python, Docker, Elasticsearch, 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 Hugging Face models for language generation and Elasticsearch for document retrieval. You'll retrieve relevant documents and generate responses based on those documents. This project will help you gain key foundational skills in AI-driven search systems.

Who this project is for:
This project is designed for beginners to intermediate-level developers. Even if you've never built a search system or worked with AI models, this project will help you understand the concepts behind Retrieval-Augmented Generation, how to integrate different technologies, and give you a practical application to add to your portfolio.

Architecture Overview:

System Components:

1. **Elasticsearch**
   This is a search engine and database that stores your documents and allows you to find relevant information quickly. It does this using both keyword searches and vector-based (semantic) searches.
2. **Hugging Face Models**
   These are pre-trained language models used to generate answers or text based on the documents retrieved from Elasticsearch.
3. **Python**
   You'll use Python as the main programming language to glue everything together: from fetching documents to generating text.

4. **Docker (Optional)**
   Docker makes it easier to run Elasticsearch by allowing you to "containerize" it so that it runs independently of your system's other applications.

Basic Flow:

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

This architecture provides the building blocks for more advanced systems. You'll also learn how to integrate Hugging Face models for text generation and manage Elasticsearch for document retrieval.

Step-by-Step Instructions

Step 1: Setting Up Your Environment

What we're doing here:
You'll set up the basic tools you'll need to build this project: Python, Docker, and Elasticsearch. Python will handle your code, and Docker will run Elasticsearch. If you're unfamiliar with these tools, don't worry! We'll guide you through every step.

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)

Docker is optional for this project, but it's highly recommended to run Elasticsearch 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 Elasticsearch with Docker

What we're doing here:
In this step, you'll run Elasticsearch inside a Docker container. This will allow you to use Elasticsearch without having to install it directly on your computer.

1. Pull the official Elasticsearch Docker image:

```
docker pull elasticsearch:8.0.0
```

2. Run the Docker container with Elasticsearch:

```
docker run -d --name elasticsearch -p 9200:9200 -e
"discovery.type=single-node" elasticsearch:8.0.0
```

- This command starts Elasticsearch on your local machine, making it accessible at `http://localhost:9200`.
- You can check if it's running by opening a browser and navigating to `http://localhost:9200`.

Step 3: Setting Up Python and Installing Required Libraries

What we're doing here:
You'll set up a virtual Python environment to keep your project's dependencies organized, and then you'll install the necessary Python libraries.

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:

```
elasticsearch==8.0.0

transformers==4.25.0

torch==1.11.0
```

4. Install the libraries by running:

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

Step 4: Indexing Documents into Elasticsearch

What we're doing here:
You'll index some sample documents into Elasticsearch. These will be the documents your RAG system will search through when generating responses.

1. Create a file called `index_documents.py` in the `local_rag_project` folder. This is where you'll write Python code to index documents into Elasticsearch.
   Important: Save the file with a `.py` extension.
2. Copy and paste this Python code into `index_documents.py`:

```
from elasticsearch import Elasticsearch

import json
```

```
# Connect to Elasticsearch

es = Elasticsearch("http://localhost:9200")


# Load your documents

with open('documents.json') as f:

    documents = json.load(f)


# Index each document

for i, doc in enumerate(documents):

    es.index(index='my_documents', id=i, body=doc)


print("Documents indexed successfully!")
```

3. Create a `documents.json` file with some sample data. Each entry should contain a simple document (e.g., a sentence or paragraph of text). Example:

```
[    {"content": "Elasticsearch is a distributed search engine."},
{"content": "Hugging Face provides pre-trained language models."}]
```

4. Run the indexing script:

```
python index_documents.py
```

Step 5: Integrating Hugging Face Models

What we're doing here:
In this step, you'll use a pre-trained Hugging Face model to generate text based on the documents retrieved from Elasticsearch.

1. Create a file called `rag_system.py` in the `local_rag_project` folder. Again, save it with a `.py` extension.
2. Copy and paste this code into `rag_system.py`:

```
from elasticsearch import Elasticsearch

from transformers import pipeline



# Connect to Elasticsearch

es = Elasticsearch("http://localhost:9200")



# Load Hugging Face model

generator = pipeline('text-generation', model='gpt2')



# Function to search documents in Elasticsearch

def search_documents(query):

    search_query = {

        "query": {

            "match": {"content": query}

        }
    }

    results = es.search(index='my_documents', body=search_query)

    return results['hits']['hits']


# Function to generate answers using Hugging Face

def generate_answer(documents, query):

    context = " ".join([doc['_source']['content'] for doc in 
documents])

    prompt = f"Based on the following context: {context}. Answer the 
question: {query}"

    result = generator(prompt, max_length=150, num_return_sequences=1)

    return result[0]['generated_text']


# Example usage

if __name__ == "__main__":

    query = input("Enter your question: ")

    documents = search_documents(query)

    answer = generate_answer(documents, query)

    print(answer)
```

3. Run the system:

```
python rag_system.py
```

4. Ask a question based on the documents you indexed, and the Hugging Face model will generate a response.

Optional: Step 6: Adding GPU Support with CUDA

What we're doing here:
If you have an Nvidia GPU, you can use CUDA to accelerate the Hugging Face model's inference time.

1. Install the correct version of CUDA following the instructions here: `https://pytorch.org/get-started/locally/`
2. Modify the Hugging Face pipeline to use the GPU:

```
generator = pipeline('text-generation', model='gpt2', device=0)
```

This tells Hugging Face to use the GPU (`device=0`).

Troubleshooting Tips:

- **Elasticsearch not connecting**: Ensure Docker is running and Elasticsearch is accessible at `http://localhost:9200`.
- **Indexing issues**: Double-check your `documents.json` file. Make sure the syntax is correct.
- **Hugging Face model not loading**: Ensure that the model name (`gpt2`) is correct, and you have enough system memory to load it.
- **Out-of-memory issues**: Try using a smaller model like `distilgpt2` if your system can't handle the default one.

Conclusion:

Congratulations! You've successfully built a Local Retrieval-Augmented Generation (RAG) system using Hugging Face and Elasticsearch. This project demonstrates your ability to combine search and language models into a functional system, making it a valuable addition to your portfolio. Keep building on this foundation by adding new features or deploying it in a cloud environment.