End-to-End Guide: Build a Secure, Dockerized RAG Workflow with Postgres Vector DB

💡 Goal

Create a full Retrieval-Augmented Generation (RAG) system that:

- Loads documents into a Postgres + pgvector database.
- Builds a LangChain retriever.
- Uses a local Mistral-7B model.
- Deploys Postgres + RAG API together using Docker Compose.
- Secures database credentials using .env.

🛠️ Part 1: Load Documents into Postgres Vector DB

1.1 Install Required Python Packages

pip install psycopg2 langchain transformers sentence-transformers faiss-cpu pgvector sqlalchemy python-dotenv

1.2 Create .env File

.env

POSTGRES_HOST=localhost

POSTGRES_PORT=5432

POSTGRES_DB=ragdb

POSTGRES_USER=postgres

POSTGRES_PASSWORD=yourpassword

Never commit your .env file to GitHub.

1.3 Python Script: load_documents_to_pg.py

import psycopg2

import numpy as np

from langchain.embeddings import HuggingFaceEmbeddings

```
from langchain.document_loaders import DirectoryLoader, TextLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from dotenv import load_dotenv
import os

# Load environment variables
load_dotenv()

POSTGRES_HOST = os.getenv("POSTGRES_HOST")
POSTGRES_PORT = os.getenv("POSTGRES_PORT")
POSTGRES_DB = os.getenv("POSTGRES_DB")
POSTGRES_USER = os.getenv("POSTGRES_USER")
POSTGRES_PASSWORD = os.getenv("POSTGRES_PASSWORD")

# Connect to Postgres
conn = psycopg2.connect(
   host=POSTGRES_HOST,
   port=POSTGRES_PORT,
   database=POSTGRES_DB,
   user=POSTGRES_USER,
   password=POSTGRES_PASSWORD
)
cursor = conn.cursor()

# Load and split documents
loader = DirectoryLoader('./documents', glob="**/*.txt", loader_cls=TextLoader)
```

```
documents = loader.load()
splitter = RecursiveCharacterTextSplitter(chunk_size=500, chunk_overlap=50)
docs = splitter.split_documents(documents)

# Generate embeddings
embedding_model = HuggingFaceEmbeddings(model_name="sentence-transformers/all-MiniLM-L6-v2")

# Insert into Postgres
for doc in docs:
    text = doc.page_content
    vector = embedding_model.embed_query(text)
    vector = np.array(vector).tolist()

    cursor.execute(
        "INSERT INTO documents (content, embedding) VALUES (%s, %s)",
        (text, vector)
    )

conn.commit()
cursor.close()
conn.close()
print("✅ Successfully loaded documents into Postgres!")
```

💡 Part 2: Build LangChain Retriever

2.1 Python Script: rag_retriever.py

```
from langchain.vectorstores.pgvector import PGVector
from langchain.embeddings import HuggingFaceEmbeddings
from langchain.llms import HuggingFacePipeline
from langchain.chains import RetrievalQA
from transformers import pipeline
from dotenv import load_dotenv
import os

# Load environment variables
load_dotenv()

# Setup Embedding Function
embeddings = HuggingFaceEmbeddings(model_name="sentence-transformers/all-MiniLM-L6-v2")

# Connect to PGVector
connection_string = f"postgresql+psycopg2://{os.getenv('POSTGRES_USER')}:{os.getenv('POSTGRES_PASSWORD')}@{os.getenv('POSTGRES_HOST')}:{os.getenv('POSTGRES_PORT')}/{os.getenv('POSTGRES_DB')}"

vectorstore = PGVector(
    connection_string=connection_string,
    embedding_function=embeddings,
    collection_name="documents",
)
retriever = vectorstore.as_retriever()
```

```
# Load Mistral-7B
mistral_pipeline = pipeline(
    "text-generation",
    model="path_to_mistral_model",
    device_map="auto",
    torch_dtype="auto",
    max_new_tokens=512,
)
llm = HuggingFacePipeline(pipeline=mistral_pipeline)

# Build RetrievalQA Chain
qa_chain = RetrievalQA.from_chain_type(
    llm=llm,
    retriever=retriever
)

def ask(question: str):
    return qa_chain.run(question)
```

🌐 Part 3: Dockerize Postgres + RAG API with Docker Compose

3.1 Dockerfile for RAG API

```
FROM python:3.11-slim

WORKDIR /app
```

```
COPY . .

RUN apt-get update && apt-get install -y gcc git && rm -rf /var/lib/apt/lists/*
RUN pip install --upgrade pip
RUN pip install -r requirements.txt

EXPOSE 8000

CMD ["uvicorn", "api_server:app", "--host", "0.0.0.0", "--port", "8000", "--reload"]
```

3.2 requirements.txt

```
fastapi
uvicorn
langchain
transformers
sentence-transformers
psycopg2
pgvector
sqlalchemy
python-dotenv
```

3.3 FastAPI Server: api_server.py

```
from fastapi import FastAPI
from pydantic import BaseModel
from rag_retriever import ask

app = FastAPI()
```

```
class QueryRequest(BaseModel):
    question: str


@app.post("/query")
async def query_rag(request: QueryRequest):
    answer = ask(request.question)
    return {"answer": answer}
```

3.4 docker-compose.yml

```
version: "3.8"

services:
  db:
    image: ankane/pgvector
    container_name: pgvector-db
    environment:
      POSTGRES_PASSWORD: yourpassword
      POSTGRES_DB: ragdb
    ports:
      - "5432:5432"
    volumes:
      - pgdat
```