GenAI Stack: What it is and How to Build Your Own

A Docker Guide

Introduction

Ready to dive into building Generative AI apps but dreading the complex setup? This guide will walk you through how to get a full GenAI stack up and running quickly in a step-by-step format. We'll also explore two practical applications: extracting insights from StackOverflow and interpreting PDFs using RAG (Retriever-Augmented Generation) technology.

We'll start with a comprehensive introduction to what a Generative AI (GenAI) stack is and how to build one from scratch. Then, we'll show you how using Docker can streamline the development of your stack.

Prerequisites

Before you can start building your Generative AI tech stack, you'll need the following software. You can download and find the usage steps outlined in detail in the appropriate sections of this document.

- Docker Desktop
- Docker Compose
- Text Editor (dealer choice - pick VIM)
- GitHub Desktop
- Langchain
- Ollama

What is GenAI?

Generative AI, often abbreviated as GenAI, empowers users to submit various prompts to produce new content, including text, images, videos, sounds, code, 3D designs, and more. It acquires knowledge by training on existing digital content and documents found online.

This technology continuously advances as it processes and learns from additional data. It utilizes AI models and algorithms trained on extensive, unlabeled datasets.

GenAI architecture overview

The following diagram illustrates the typical components and topology of GenAI deployments.

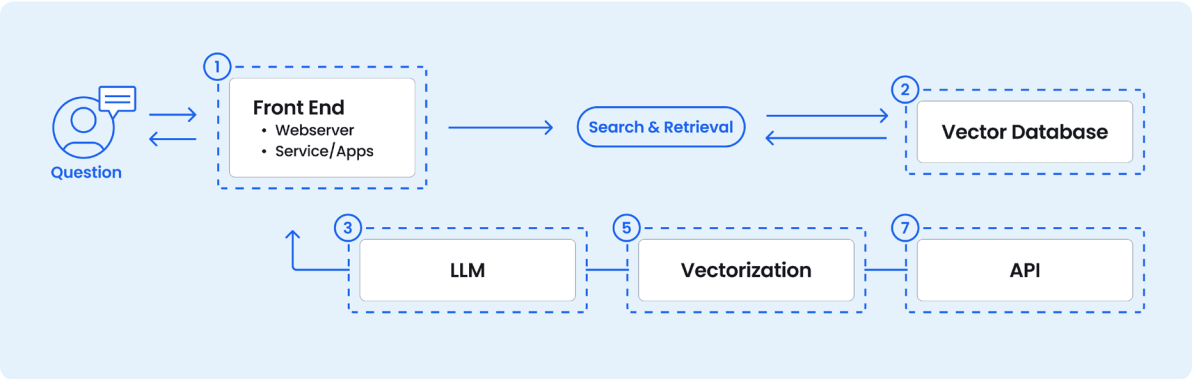

Figure 1: Diagram of a typical GenAI application architecture

When you build a GenAI stack from scratch, you'll first need to choose the technologies you'll use. You'll need to satisfy each of the following components and configure them to work together.

Images for these components can be found on Docker Hub and other online sources. But as with any software development project, make sure you're using trusted and tested software.

- **Front-End:** The front-end will consist of a web server, web framework (Vu.js, Angular, or React), CSS framework (bootstrap or material-ui), and WebSocket or AJAX. These will let the server and client communicate.
- **Vector Database:** A vector database efficiently stores, indexes, and retrieves vector embeddings. Vector embeddings are multi-dimensional representations of data typically used in machine learning and similarity search tasks.
- **Large Language Model:** Large Language Models (LLMs) are the core engines of GenAI applications. They drive the generation of human-like text and open a broad spectrum of functionalities. Since LLM models have designing and training for different use cases, you'll want to choose one that best suits your application.
- **Vectorization:** Vectorization converts text to numerical formats that machine learning models can process, often involving techniques like word embeddings. Several vector databases include this functionality, but there are also dedicated tools to manage this process.
- **Data Storage for RAG:** RAG enhances text output by combining retrieval systems and generative models. It retrieves pertinent data for a query from a database to produce relevant and accurate responses.

The Easier way: Docker GenAI stack

Introducing Docker's Generative AI and Machine Learning Stack (DockerCon 2023)

The Docker GenAI stack is a suite of nine containers orchestrated by Docker Compose that simplifies GenAI application development. This comprehensive library of containers lets developers deploy fully functioning GenAI environments with sample applications for experimentation and proof of concept.

Additionally, the Docker GenAI stack builds the components in the containers and configures the networking and connectivity between them. With only a few commands, you can have a fully functional GenAI stack running.

In other words, the Docker GenAI stack can help you put together a DIY stack much quicker. You'll also skip many complicated steps, setups, and configurations.

What is GenAI Stack composed of?

The stack is a set of containers that make it easy to experiment with building and running Generative AI apps. These containers provide a pre-built support agent app development environment with data import and response generation use cases. It includes:

1. **Ollama:** A management tool for local LLMs
2. **Neo4j:** A vector and knowledge graph database to provide RAG support
3. **LangChain:** A platform that integrates language models with external data sources and APIs, allowing GenAI applications to perform more complex tasks
4. **Pre-configured LLMs:** Preconfigured Large Language Models such as Llama2, GPT-3.5, and GPT-4

How do GenAI components work together?

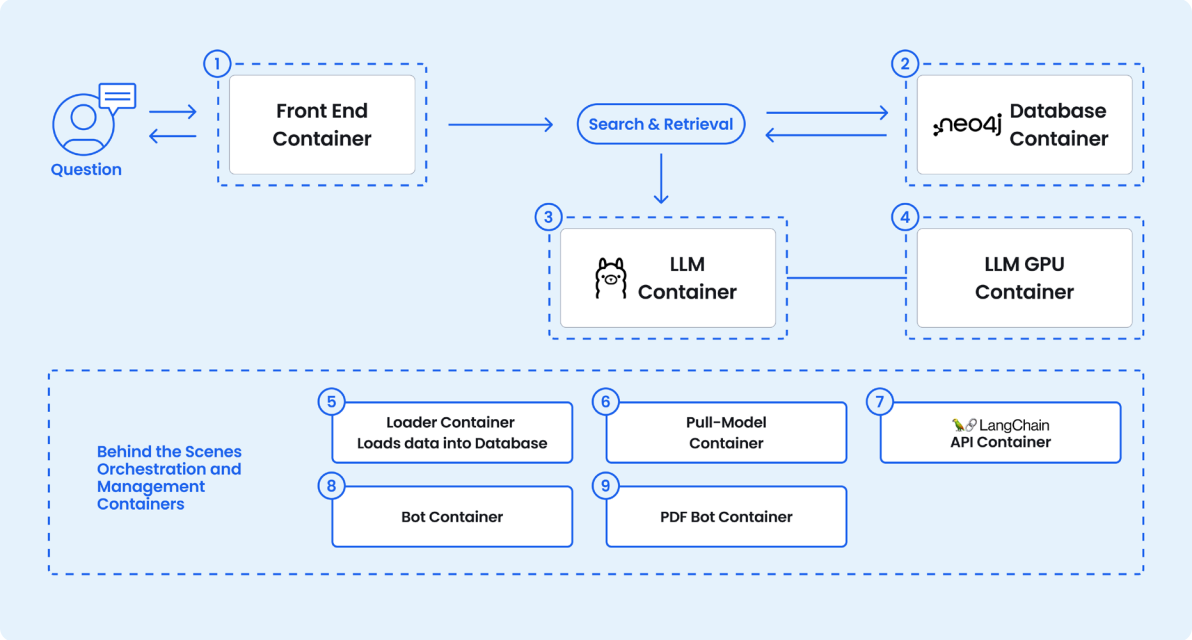

Figure 2: Diagram showing the 9 containers created when deploying the Docker GenAI Stack alongside a brief bulleted list of potentially complicated parts.

The architecture consists of the following 9 containers (shown in Figure 2):

1. **Front-end Container -** This contains the front-end user interface and is preconfigured to communicate with the LLM and Vector database.
2. **Database Container -** Neo4J is used as the vector and knowledge graph database to enable RAG support.
3. **LLM Container -** The chosen LLM is deployed inside this container.

4. **LLM GPU Container —** If the GenAI Stack is configured to utilize a GPU, this container houses the software to facilitate this.
5. **Loader Container —** This contains the logic for retrieving, chunking, and loading tags from StackOverflow into the vector database.
6. **Pull-Model Container —** Responsible for downloading and configuring the selected LLM. This container is intended to be short running, and will only run when changes are made to the LLM that is defined.
7. **API Container —** The contents of this container manage the communication between the components of the GenAI stack.
8. **Bot Container —** This is the application logic for the support chatbot.
9. **PDF Bot Container —** This application is responsible for processing uploaded PDFs and loading the chunked data into the Database Container.

Additional recommendations

Consider the following items, as they can affect performance, complexity, and operating costs.

LLM Choice

This reference architecture outlines the process for using a local LLM, Ollama, and ChatGPT. But with the Docker GenAI stack, you can change the model and model deployment as needed.

Enabling GPU Support

GenAI applications frequently benefit from GPU acceleration to enhance performance. Docker Desktop currently offers GPU acceleration support exclusively on Windows via the WSL2 backend. However, Linux users can leverage GPU acceleration through a native installation of the Docker Engine.

You can also configure Docker to utilize GPU for additional performance, and testing.

Deployment Options

If GPU support is required, it's critical to ensure that the provider allows for GPU access from self-deployed LLMs.

Get your GenAI Stack running in 10 steps

The following steps are your guide to the installation and configuration of the GenAI stack and sample applications.

Step 1. Install Docker Desktop

If you don't have Docker Desktop, download the current version. Our documentation provides additional information and instructions on how to install Docker Desktop.

Step 2. Install Ollama

Download and install Ollama.

*Please note that currently, Windows support is in **preview** by Ollama, so Windows users may choose to generate an OpenAI API key and configure the stack to use GPT-3.5 or GPT-4 in the .env file.*

Step 3. Create OpenAI Secret API Keys (optional)

If you choose to use an LLM from OpenAI, you'll need to create a new OpenAI Secret API Key. You'll use this API key in the .env file to configure the Docker GenAI stack to use OpenAI LLMs.

Step 4. Sign Up for LangSmith for API Keys

LangSmith allows you to monitor and evaluate your application closely so you can ship quickly and confidently. Visit their website to create Endpoint and API Keys.

A LangSmith account is required for this step. You will need the following information:

```
LANGCHAIN_ENDPOINT="https://api.smith.langchain.com"
LANGCHAIN_TRACING_V2=true # false
LANGCHAIN_PROJECT=default
LANGCHAIN_API_KEY=YOUR_API_KEY
```

Step 5. Clone the GenAI Stack repository

Use the following Git command to clone the GenAI Stack:

```
git clone https://github.com/docker/genai-stack
cd genai-stack
```


Step 6. Create .env file (or modify sample.env)

Using a text or code editor, create a new .env file or modify the existing sample.env file to reflect the following.

```
cat .env
OPENAI_API_KEY=YOUR_API_KEY
OLLAMA_BASE_URL=http://host.docker.internal:11434
NEO4J_URI=neo4j://database:7687

NEO4J_USERNAME=neo4j
NEO4J_PASSWORD=password
LLM=llama2 #or any Ollama model tag, or gpt-4 or gpt-3.5
EMBEDDING_MODEL=sentence_transformer #or openai or ollama

LANGCHAIN_ENDPOINT="https://api.smith.langchain.com"
LANGCHAIN_TRACING_V2=true # false
LANGCHAIN_PROJECT=default
LANGCHAIN_API_KEY=YOUR_API_KEY
```

Step 7. Bring up Compose services with Docker Compose

Execute the following Docker Compose command to initiate the build process.

```
docker compose up -d --build
```


You'll see output like the following:

```
 pulling 8daa9615cce3...  93% |               | (3.5/3.8 GB, 29 MB/s) [2m...
pulling model (250s) - will take several minutes
genai-stack-pull-model-1  | ... pulling model (260s) - will take several minutes
pulling 8daa9615cce3... 100% |              | (3.8/3.8 GB, 29 MB/s)
genai-stack-loader-1      | Collecting usage statistics. To deactivate, set
browser.gatherUsageStats to False.
genai-stack-loader-1      |
genai-stack-pdf_bot-1     |
genai-stack-pdf_bot-1     | Collecting usage statistics. To deactivate, set
browser.gatherUsageStats to False.


genai-stack-pdf_bot-1     |
genai-stack-bot-1         |
genai-stack-bot-1         | Collecting usage statistics. To deactivate, set
browser.gatherUsageStats to False.
genai-stack-bot-1         |
genai-stack-bot-1         |
genai-stack-bot-1         |   You can now view your Streamlit app in your browser.
genai-stack-bot-1         |
genai-stack-bot-1         |   URL: http://0.0.0.0:8501
genai-stack-bot-1         |
genai-stack-pdf_bot-1     |
genai-stack-pdf_bot-1     |   You can now view your Streamlit app in your browser.
genai-stack-pdf_bot-1     |
genai-stack-pdf_bot-1     |   URL: http://0.0.0.0:8503
genai-stack-pdf_bot-1     |
genai-stack-loader-1      |
genai-stack-loader-1      |   You can now view your Streamlit app in your browser.
genai-stack-loader-1      |
genai-stack-loader-1      |   URL: http://0.0.0.0:8502
genai-stack-loader-1      |
```


Step 8. View the Services in Docker Desktop

In Docker Desktop, you can check resource metrics and make sure the GenAI stack containers are running.

Note that the pull-model container will only run when a new model is being pulled. It will be stopped and have an “Exited” status after the model has been pulled.

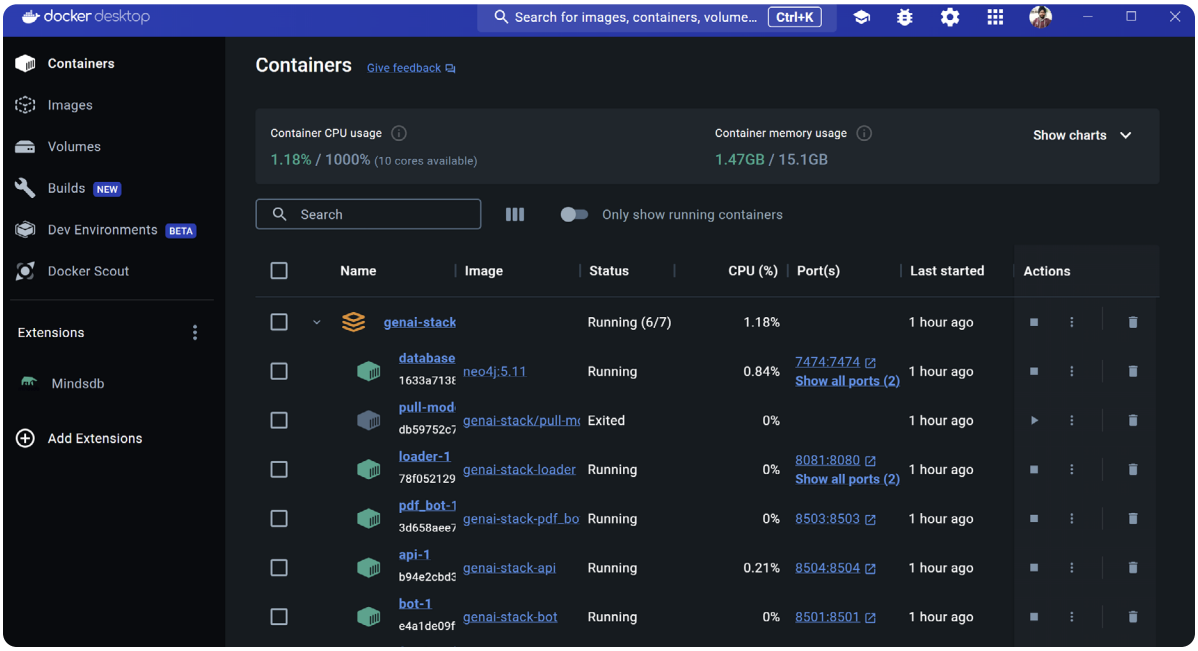

Figure 3: Screenshot showing the running GenAI Stack containers using Docker Desktop.

Step 9. Verify your app is functioning

Visit http://localhost:8502 to access the StackOverflow loader application. The image below is the page that will be displayed and where you set your ingestion parameters.

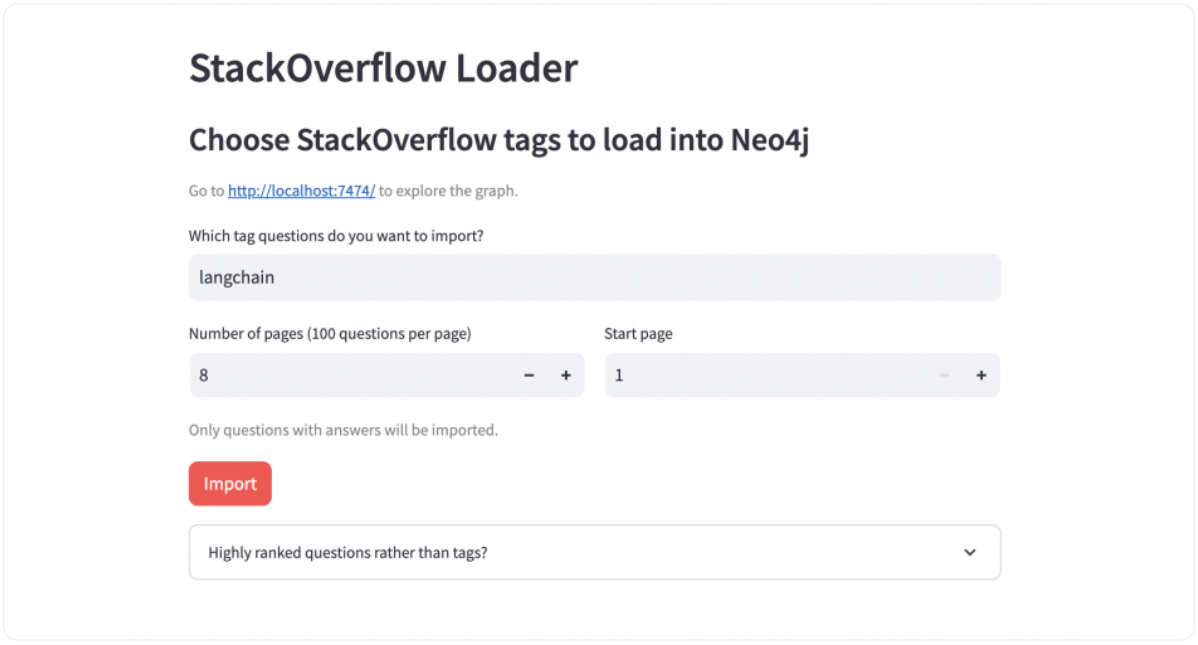

Figure 4: Screenshot showing the expected results when loading the StackOverflow sample application.

Clicking "`Import`" initiates the process that generates the embeddings for RAG functionality. After or during the import, you can click the link to http://localhost:7474. Log in with username "neo4j" and password "password" as configured in Docker Compose. There, you can see an overview in the left sidebar and show some connected data by clicking on the "pill" with the counts.

The data loader will import the graph using the following schema. Enter the tags for the content to ingest, adjust the number of pages, and start the page to match your needs.

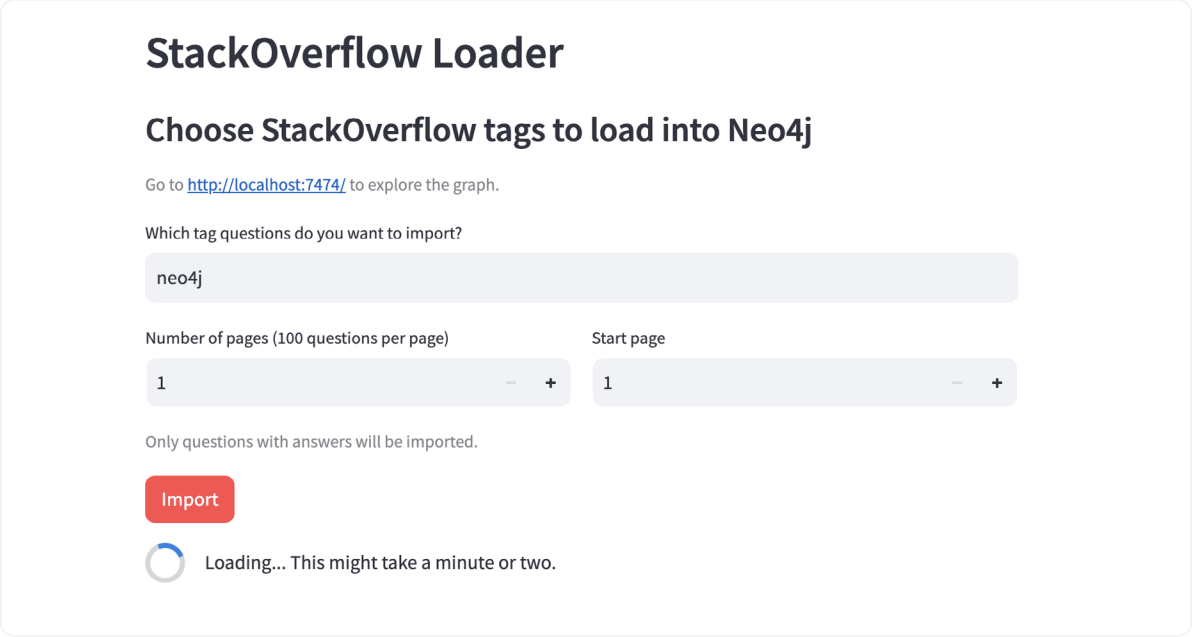

Figure 5: Screenshot showing the StackOverflow loader sample application importing data to the vector database.

Result:

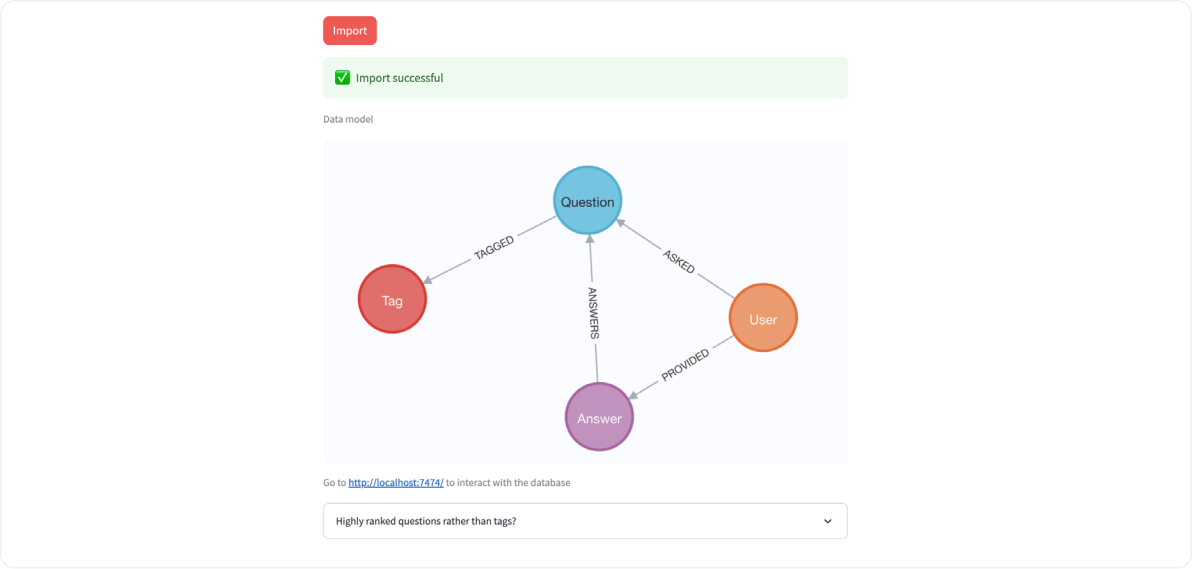

Figure 6: Screenshot showing the expected results of importing data using the StackOverflow sample application. This expected data includes tagged questions asked by the user that are then answered.

The graph schema for Stack Overflow consists of nodes representing Questions, Answers, Users, and Tags. Users are linked to Questions they've asked via the "ASKED" relationship and to Answers they've provided with the "ANSWERS" relationship. Each Answer is also inherently associated with a specific Question. Furthermore, Questions are categorized by their relevant topics or technologies using the "TAGGED" relationship connecting them to Tags.

Step 10. Access Neo4j

Open http://localhost:7474 and log in with username "neo4j" and password "password" as configured in Docker Compose.

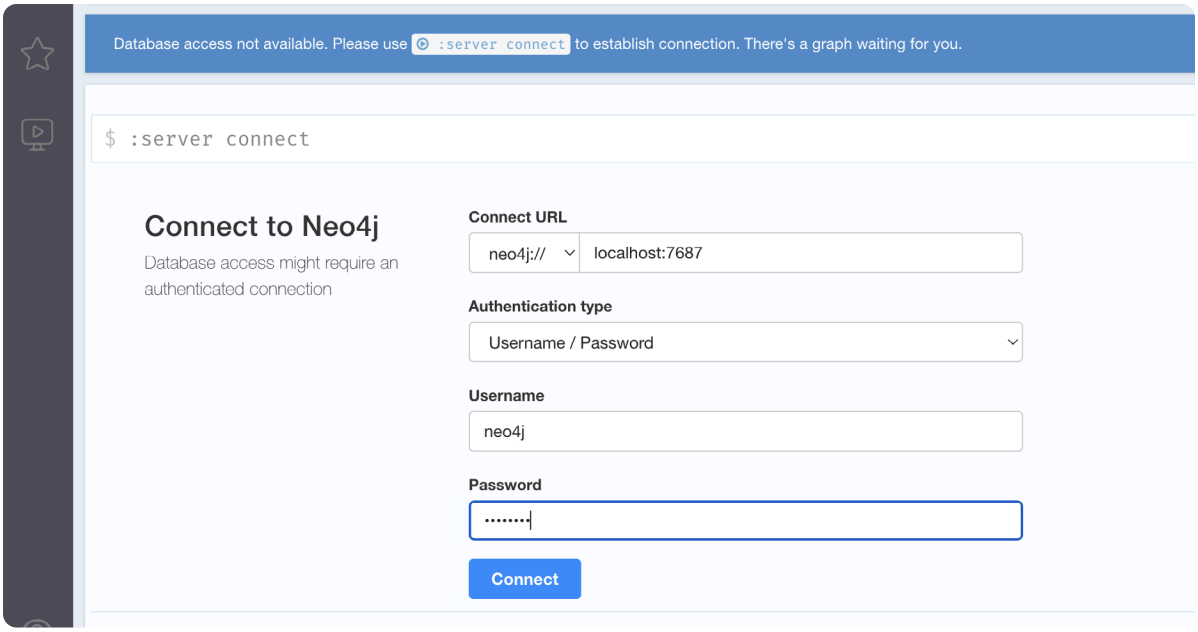

Figure 7: Screenshot showing the login screen for connecting to the included Neo4j database.

Query the Imported Data via a Chat Interface Using Vector + Graph Search

This application lets you interact with your imported data through a classic LLM chat-like user interface accessible at http://localhost:8501. You can ask questions and receive answers based on the information you've uploaded.

You can choose between two search methods: "Classic" or "RAG mode."

RAG mode leverages a smarter search technique that analyzes your data to find the most relevant questions and answers. This can potentially lead to more accurate results.

Disabling RAG mode uses the LLM's knowledge directly. When RAG mode is selected, the application uses similarity search using text embedding and graph queries to find the most relevant questions and answers in the database.

Accessing GenAI Stack PDF Bot

Open http://0.0.0.0:8503/ on the browser to access the PDF Bot that allows you to chat with your PDF file.

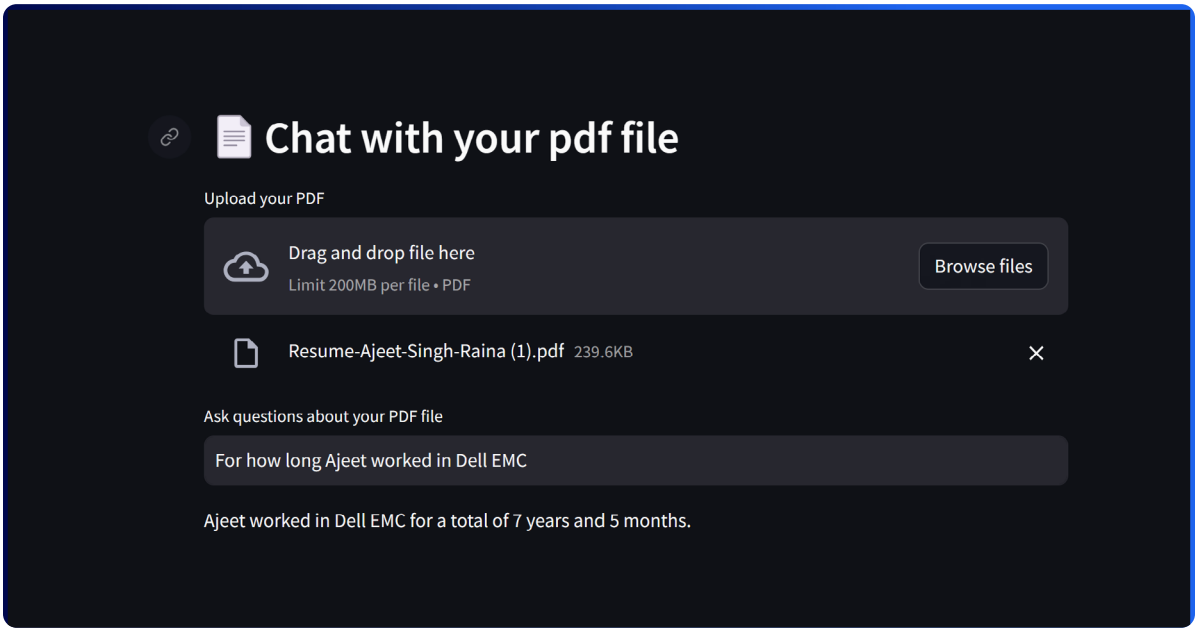

Figure 8: Screenshot showing user interaction with the PDF reader sample application.

Conclusion

The Docker GenAI stack provides AI developers with a simplified way to build GenAI applications. This stack empowers developers and lets them concentrate on innovation and the creation of impactful AI-driven solutions. And we encourage developers to leverage this guide to explore the vast possibilities of GenAI development with Docker. To continue your learning journey with Docker, check out what others built with the Docker GenAI stack during our recent Hackathon on YouTube.

Related Content:

- Introducing Docker's Generative AI and Machine Learning Stack (DockerCon 2023)
- Docker for Machine Learning, AI, and Data Science | Workshop (DockerCon 2023)
- How to Build LangChain-Based, Database-Backed GenAI Applications within Docker (DockerCon 2023)