Implementing Retrieval Augmented Generation (RAG): A Hands-On Guide!

Pavan Belagatti · Follow
Published in Generative AI · 7 min read · Apr 18, 2024

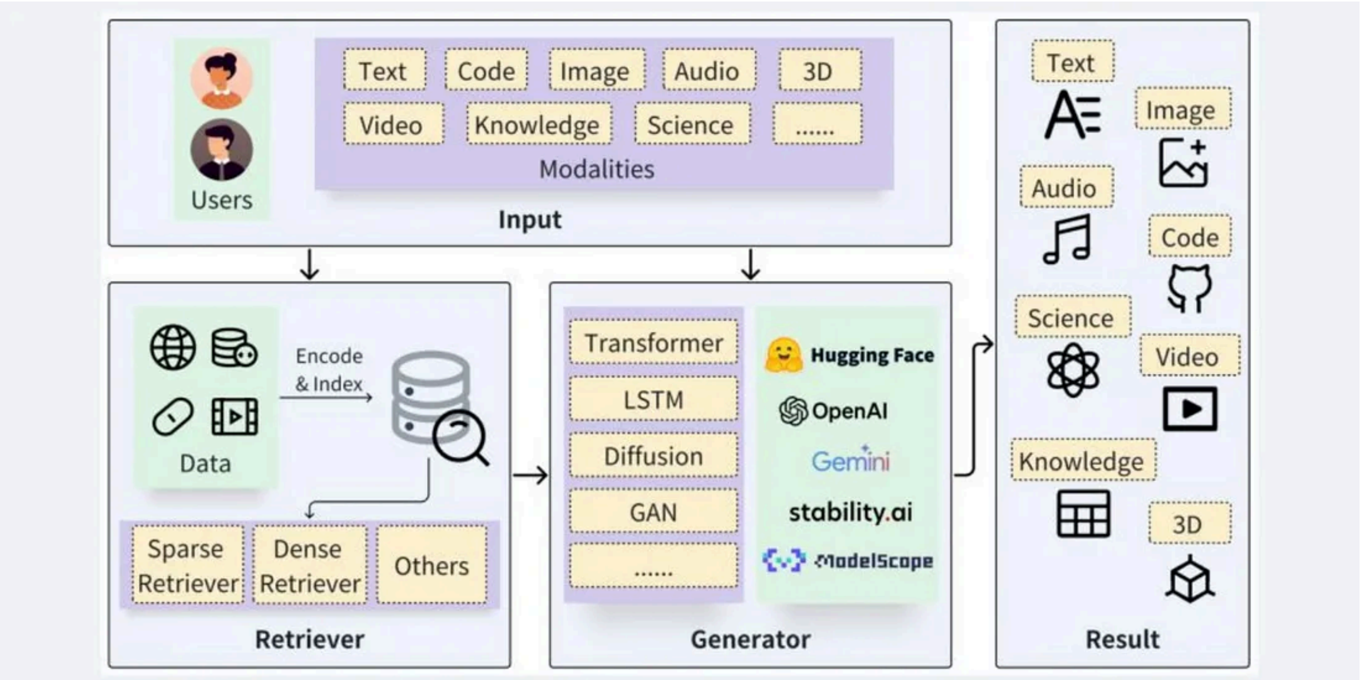


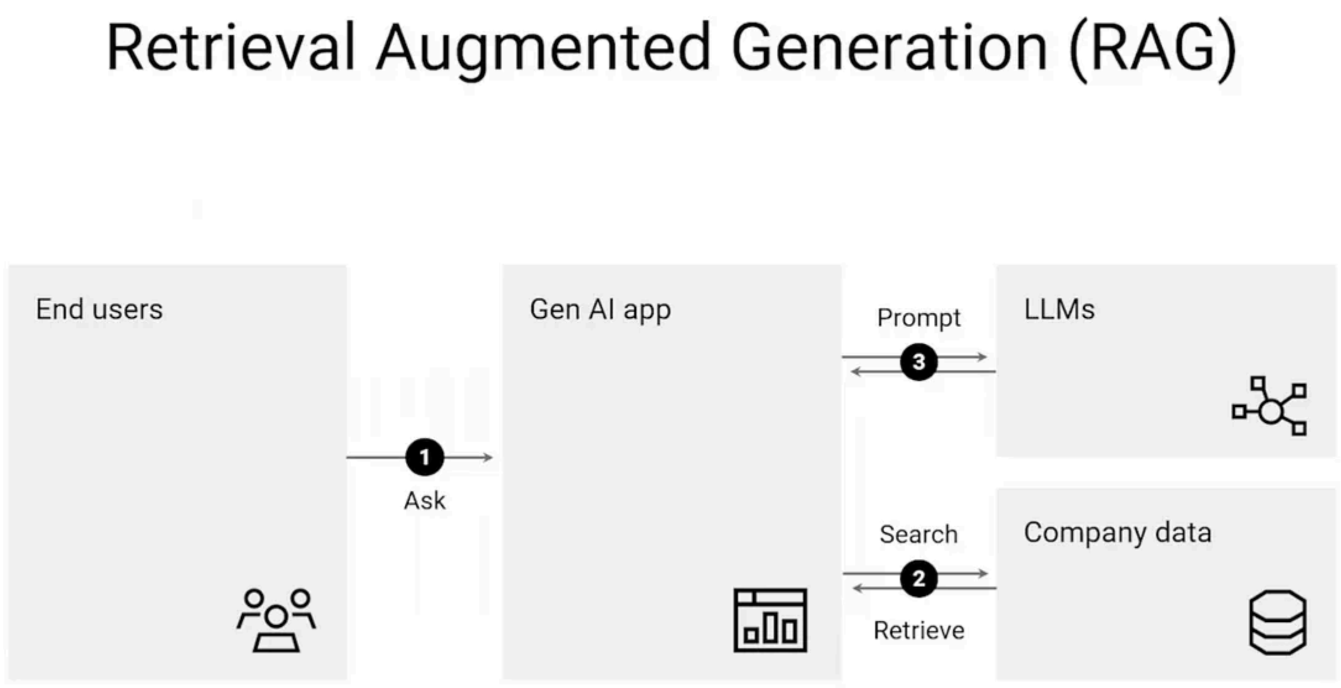
Image Credits: https://arxiv.org/abs/2402.19473

Large language models (LLMs) are becoming the backbone of most of the organizations these days as the whole world is making the transition towards AI. While LLMs are all good and trending for all the positive reasons, they also pose some disadvantages if not used properly. Yes, LLMs can sometimes produce the responses that aren't expected, they can be fake, made up information or even biased. Now, this can happen for various reasons. We call this process of generating misinformation by LLMs as hallucination.

There are some notable approaches to mitigate the LLM hallucinations such as fine-tuning, prompt engineering, retrieval augmented generation (RAG) etc. Retrieval augmented generation (RAG) has been the most talked about approach in mitigating the hallucinations faced by large language models.

Let's dwell deeper into understanding how RAG works through a hands-on implementation using SingleStore as a vector database to store the vector data.

What is Retrieval Augmented Generation (RAG)?



Large Language Models (LLMs) sometimes produce hallucinated answers and one of the techniques to mitigate these hallucinations is by RAG. For an user query, RAG tends to retrieve the information from the provided source/information/data that is stored in a vector database. A vector database is the one that is a specialized database other than the traditional databases where vector data is stored.

Vector data is in the form of embeddings that captures the context and meaning of the objects. For example, think of a scenario where you would like to get custom responses from your AI application.

First, the organization's documents are converted into embeddings through an embedding model and stored in a vector database. When a query is sent to the AI application, it gets converted into a vector query embedding and goes through the vector database to find the most similar object by vector similarity search. This way, your LLM-powered application doesn't hallucinate since you have already instructed it to provide custom responses and is fed with the custom data.

One simple use case would be the customer support application, where the custom data is fed to the application stored in a vector database and when a user query comes in, it generates the most appropriate response related to your products or services and not some generic answer. This way, RAG is revolutionizing many other fields in the world.

RAG pipeline

The RAG pipeline basically involves three critical components: Retrieval component, Augmentation component, Generation component.

- **Retrieval**: This component helps you fetch the relevant information from the external knowledge base like a vector database for any given user query. This component is very crucial as this is the first step in curating the meaningful and contextually correct responses.
- **Augmentation**: This part involves enhancing and adding more relevant context to the retrieved response for the user query.
- **Generation**: Finally, a final output is presented to the user with the help of a large language model (LLM). The LLM uses its own knowledge and the provided context and comes up with an apt response to the user's query.

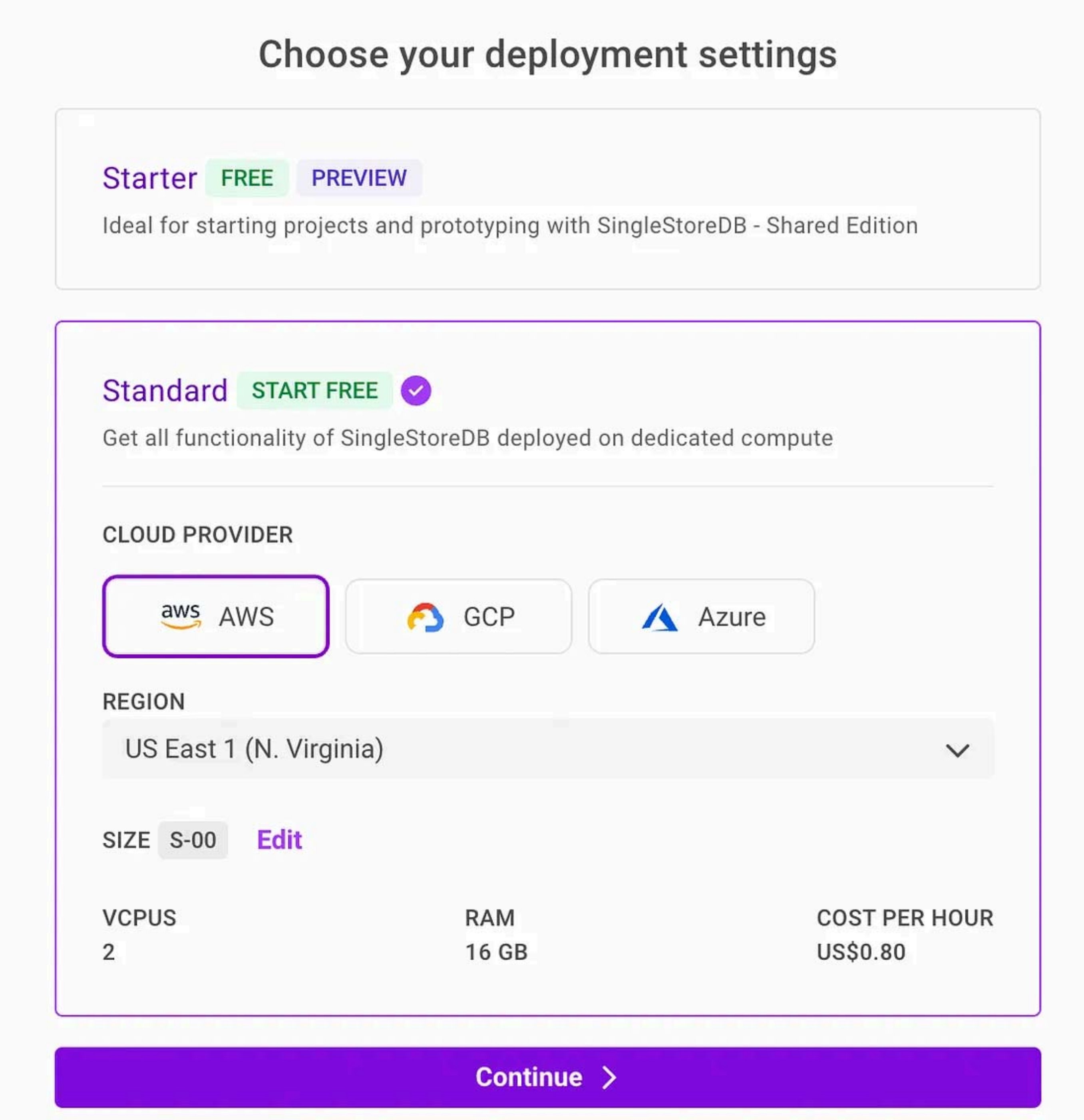
These three components are the basis of a RAG pipeline to help users to get the contextually-rich and accurate responses they are looking for. That is the reason why RAG is so special when it comes to building chatbots, question-answering systems, etc.

RAG Tutorial

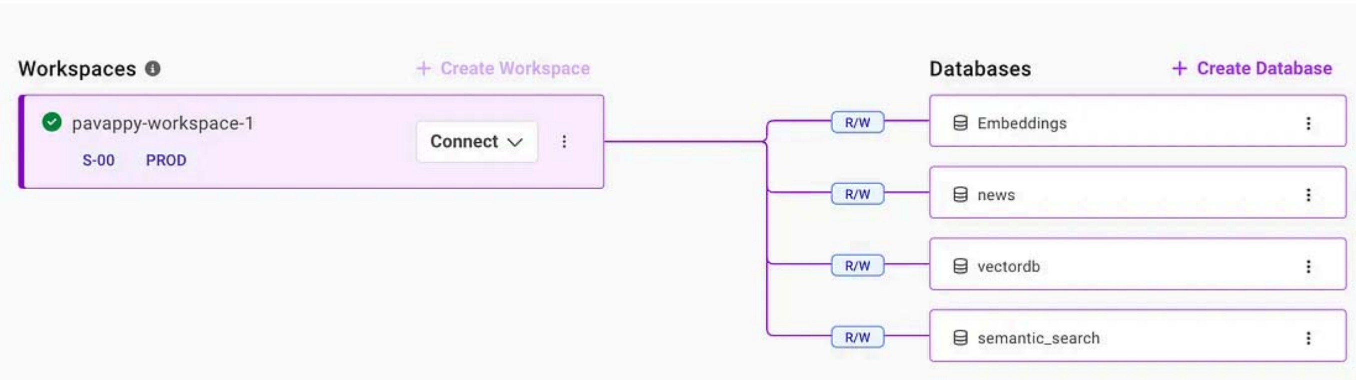
Let’s build a simple AI application that can fetch the contextually relevant information from our own data for any given user query.

Sign up to SingleStore database to use it as our vector database.

Once you sign up, you need to create a workspace. It is easy and free, so do it.

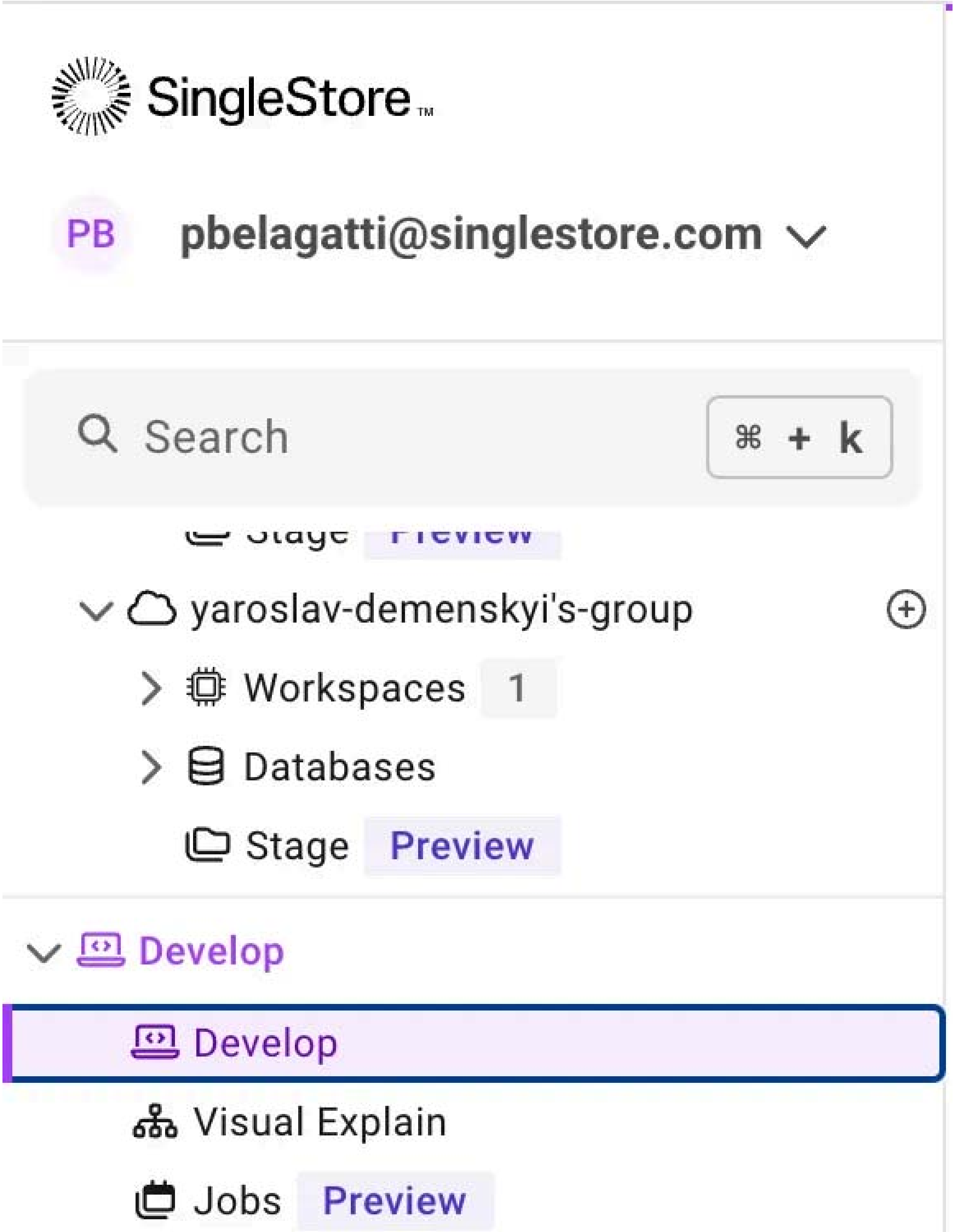


Once you create your workspace, create a database with any name of your wish.

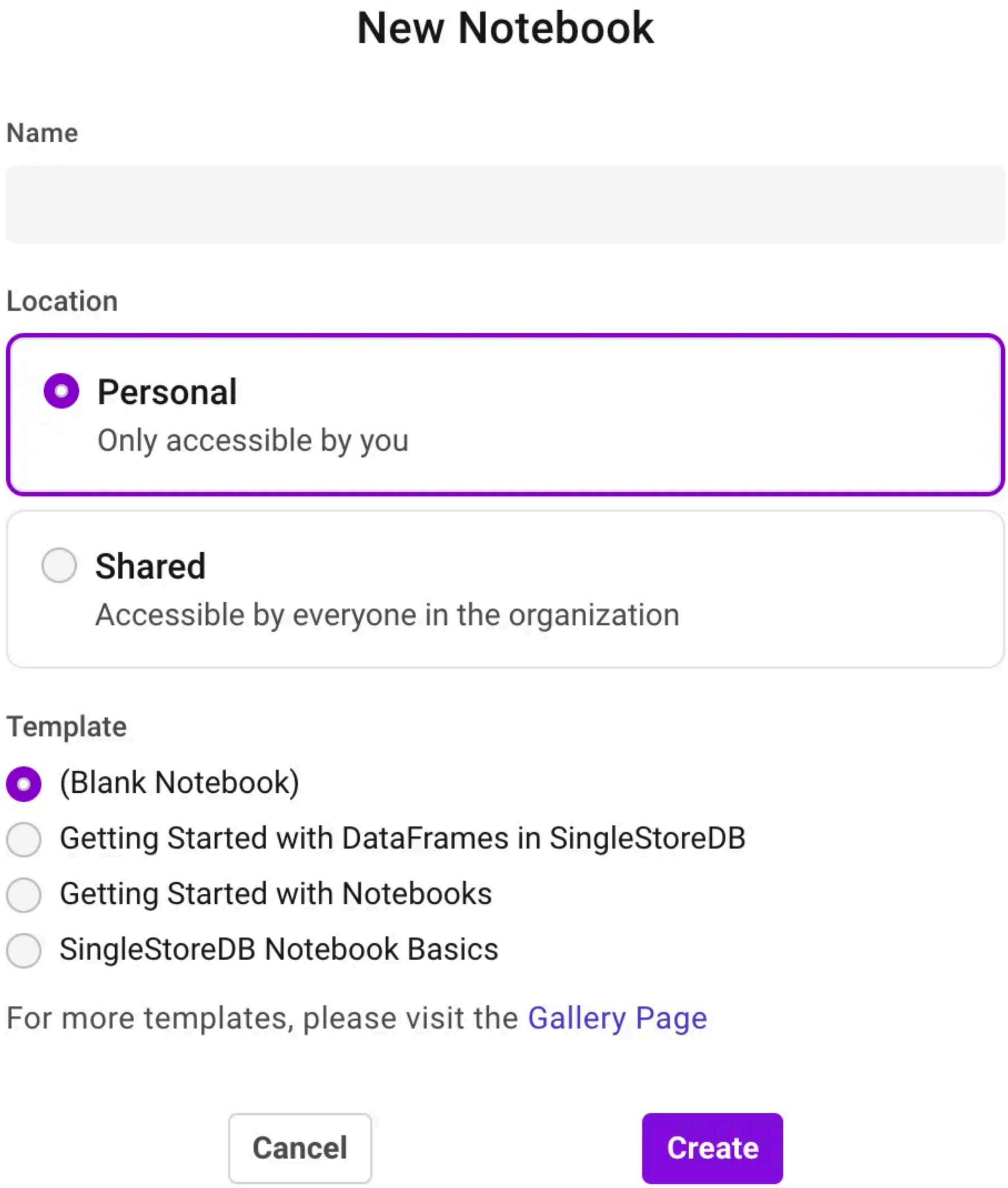


As you can see from the above screenshot, you can create the database from 'Create Database' tab on the right side.

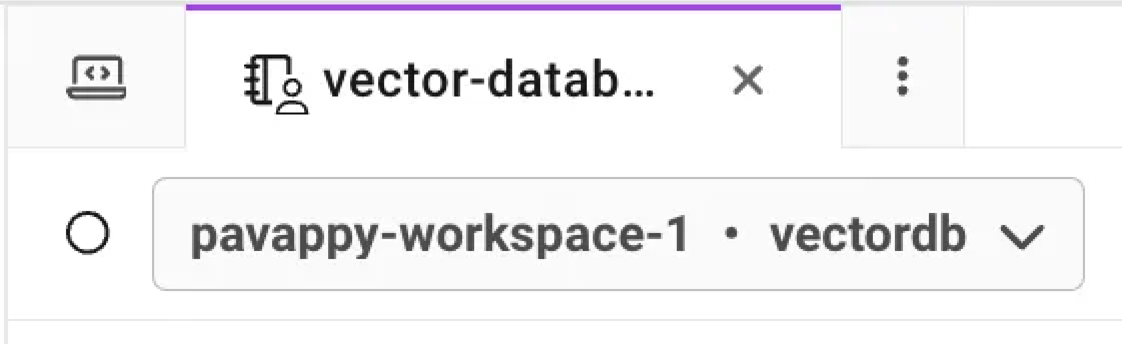
Now, let's go to 'Develop' to use our Notebooks feature [just like Jupyter Notebooks]



Create a new Notebook and name it as you wish.



Before doing anything, select your workspace and database from the dropdown on the Notebook.



Now, start adding all the below shown code snippets into your Notebook you just created as shown below.

Install the required libraries

```
!pip install openai numpy pandas singlestoredb langchain==0.1.8 langchain-community
```

Vector embeddings example

```
def word_to_vector(word):
    # Define some basic rules for our vector components
    vector = [0] * 5  # Initialize a vector of 5 dimensions

    # Rule 1: Length of the word (normalized to a max of 10 characters for simplici
    vector[0] = len(word) / 10
```

```
    # Rule 2: Number of vowels in the word (normalized to the length of the word)
    vowels = 'aeiou'
    vector[1] = sum(1 for char in word if char in vowels) / len(word)

    # Rule 3: Whether the word starts with a vowel (1) or not (0)
    vector[2] = 1 if word[0] in vowels else 0

    # Rule 4: Whether the word ends with a vowel (1) or not (0)
    vector[3] = 1 if word[-1] in vowels else 0

    # Rule 5: Percentage of consonants in the word
    vector[4] = sum(1 for char in word if char not in vowels and char.isalpha()) /

    return vector

# Example usage
word = "example"
vector = word_to_vector(word)
print(f"Word: {word}\nVector: {vector}")
```

Vector similarity example

```
import numpy as np

def cosine_similarity(vector_a, vector_b):
    # Calculate the dot product of vectors
    dot_product = np.dot(vector_a, vector_b)
    # Calculate the norm (magnitude) of each vector
    norm_a = np.linalg.norm(vector_a)
    norm_b = np.linalg.norm(vector_b)
    # Calculate cosine similarity
    similarity = dot_product / (norm_a * norm_b)
    return similarity

# Example usage
word1 = "example"
word2 = "sample"
vector1 = word_to_vector(word1)
vector2 = word_to_vector(word2)

# Calculate and print cosine similarity
similarity_score = cosine_similarity(vector1, vector2)
print(f"Cosine similarity between '{word1}' and '{word2}': {similarity_score}")
```

Embedding models

```
OPENAI_KEY = "INSERT OPENAI KEY"
from openai import OpenAI
client = OpenAI(api_key=OPENAI_KEY)

def openAIEmbeddings(input):
 response = client.embeddings.create(
     input="input",
     model="text-embedding-3-small"
 )
 return response.data[0].embedding
print(openAIEmbeddings("Golden Retreiver"))
```

Creating a vector database with SingleStoreDB

We will be using LangChain framework, SingleStore as a vector database to store our embeddings and a public .txt file link that is about the sherlock homes stories.

Add OpenAI API key as an environment variable.

```
import os
os.environ['OPENAI_API_KEY'] = 'mention your openai api key'
```

Next, import the libraries, mention the file you want to use in the example, load the file, split it and add the file content into SingleStore database. Finally, ask the query related to the document you used.

```
import openai
from langchain.text_splitter import CharacterTextSplitter
from langchain_community.document_loaders import TextLoader
```

```
from langchain_community.embeddings import OpenAIEmbeddings
from langchain_community.vectorstores.singlestoredb import SingleStoreDB
import os
import pandas as pd
import requests

# URL of the public .txt file you want to use
file_url = "https://sherlock-holm.es/stories/plain-text/stud.txt"

# Send a GET request to the file URL
response = requests.get(file_url)

# Proceed if the file was successfully downloaded
if response.status_code == 200:
    file_content = response.text

    # Save the content to a file
    file_path = 'downloaded_example.txt'
    with open(file_path, 'w', encoding='utf-8') as f:
        f.write(file_content)

    # Now, you can proceed with your original code using 'downloaded_example.txt'
    # Load and process documents
    loader = TextLoader(file_path)  # Use the downloaded document

    documents = loader.load()
    text_splitter = CharacterTextSplitter(chunk_size=2000, chunk_overlap=0)
    docs = text_splitter.split_documents(documents)

    # Generate embeddings and create a document search database
    OPENAI_KEY = "add your openai key"  # Replace with your OpenAI API key
    embeddings = OpenAIEmbeddings(api_key=OPENAI_KEY)

    # Create Vector Database
    vector_database = SingleStoreDB.from_documents(docs, embeddings, table_name="sc

    query = "which university did he study?"
    docs = vector_database.similarity_search(query)
    print(docs[0].page_content)

else:
    print("Failed to download the file. Please check the URL and try again.")
```

Once run the above code, you will see a tab to enter your query/question you would like to ask about the sherlock holmes story we have referenced.

```
You:  In which university did they get degree of Doctor of Medicine
AI: The University of London.

You:  what did the campaign brought?
AI: The campaign brought honours and promotion to many individuals, but for the user it brought nothing but misfortune and disaster.

You:  what the story is all about? can you share in 50 words or so?
AI: The story is about an old farmer who is trapped in a settlement where the inhabitants are controlled by a mysterious Council. The farmer's daughter
is in danger, and he is willing to sacrifice his own life to protect her from dishonor. Despite his efforts, he is unable to find a way out of the situ
ation and is left feeling helpless.

You:  Who is Watson in the story?
AI: Dr. Watson is one of the main characters in the story. He is a close friend and confidant of Sherlock Holmes, and often accompanies him on his inve
stigations. He is a former army doctor and serves as Holmes's assistant and chronicler, recording their adventures and cases in his journals.

You:  What did Sherlock Holmes go through in his life?
AI: Sherlock Holmes went through many adventures and investigations throughout his life. He was known for his keen observation and deductive reasoning,
which allowed him to solve complex cases and mysteries. His methods were often seen as unconventional, but his results were always accurate. He also st
ruggled with drug addiction and had a complicated relationship with his friend and colleague, Dr. John Watson.
```

We retrieved the relevant information from the provided data and then used this information to guide the response generation process. By converting our file into embeddings and storing them in the SingleStore database, we created a retrievable corpus of information. This way, we ensured that the responses are not only relevant but also rich in content derived from the provided dataset.