Labs Data & AI Innovation Day

Lab 2: Ingest data from json

This document describes how to use a Python notebook to populate the Azure Cosmos DB for NoSQL 'products' container with the content of a json file

Pre-requisites

Ensure that you have the following software installed on your system before proceeding with the lab:

- Visual Studio Code: A cross-platform code editor that supports Python development. You can download it from https://code.visualstudio.com/

- Python 3.10.11: The latest version of the Python programming language. You can download it from https://www.python.org/downloads/release/python-31011/

Note: If you are using a different version of Python, make sure that it is compatible with the libraries and packages used in this lab.

- Azure OpenAI account registered in the Azure subscription used for this lab
- Existing Python virtual environment as described in Lab1

Ingest data into Cosmos DB for NoSQL

In this section, you will use the Python notebook to upload sample data to your Azure Cosmos DB for NoSQL container

1. Create a "**Lab2**" folder in the "**Labs**" foler
2. Open Visual Studio Code
3. Download the original data files from **https://cosmosdbcosmicworks.blob.core.windows.net/cosmic-works-small/product.json** and **https://cosmosdbcosmicworks.blob.core.windows.net/cosmic-works-small/customer.json** to the lab2 folder on your local machine.
4. In the "**Lab2**" folder, create a new Jupyter notebook called "**import_cosmosdb_nosql.ipynb**"
5. Add the following cells

```
### Load required Python libraries
import json, os, uuid
from openai import AzureOpenAI
from azure.cosmos import CosmosClient
from dotenv import load_dotenv
from tenacity import retry, wait_random_exponential, stop_after_attempt

load_dotenv("..\.env")
```

```
# Function to create embeddings
@retry(wait=wait_random_exponential(min=1, max=20),
stop=stop_after_attempt(10))
def generate_embeddings(openai_client, text):
    """
    Generates embeddings for a given text using the OpenAI API v1.x
    """

    return openai_client.embeddings.create(
        input = text,
        model= os.getenv("AZURE_OPENAI_EMBEDDING_MODEL")
    ).data[0].embedding
```

```
# Init Azure Cosmos DB
COSMOS_DB_ENDPOINT = os.getenv('AZURE_COSMOSDB_NOSQL_ENDPOINT')
COSMOS_DB_KEY = os.getenv('AZURE_COSMOSDB_NOSQL_KEY')
DATABASE_NAME = os.getenv('AZURE_COSMOSDB_NOSQL_DATABASE_NAME')
```

```
CONTAINER_NAME = os.getenv('AZURE_COSMOSDB_NOSQL_CONTAINER_NAME')

client = CosmosClient(COSMOS_DB_ENDPOINT, COSMOS_DB_KEY)
database = client.get_database_client(DATABASE_NAME)
container = database.get_container_client(CONTAINER_NAME)
```

```
# Initialize Azure OpenAI client
openai_client = AzureOpenAI(
    api_key = os.getenv("AZURE_OPENAI_API_KEY"),
    api_version = os.getenv("AZURE_OPENAI_API_VERSION"),
    azure_endpoint =os.getenv("AZURE_OPENAI_ENDPOINT")
)
```

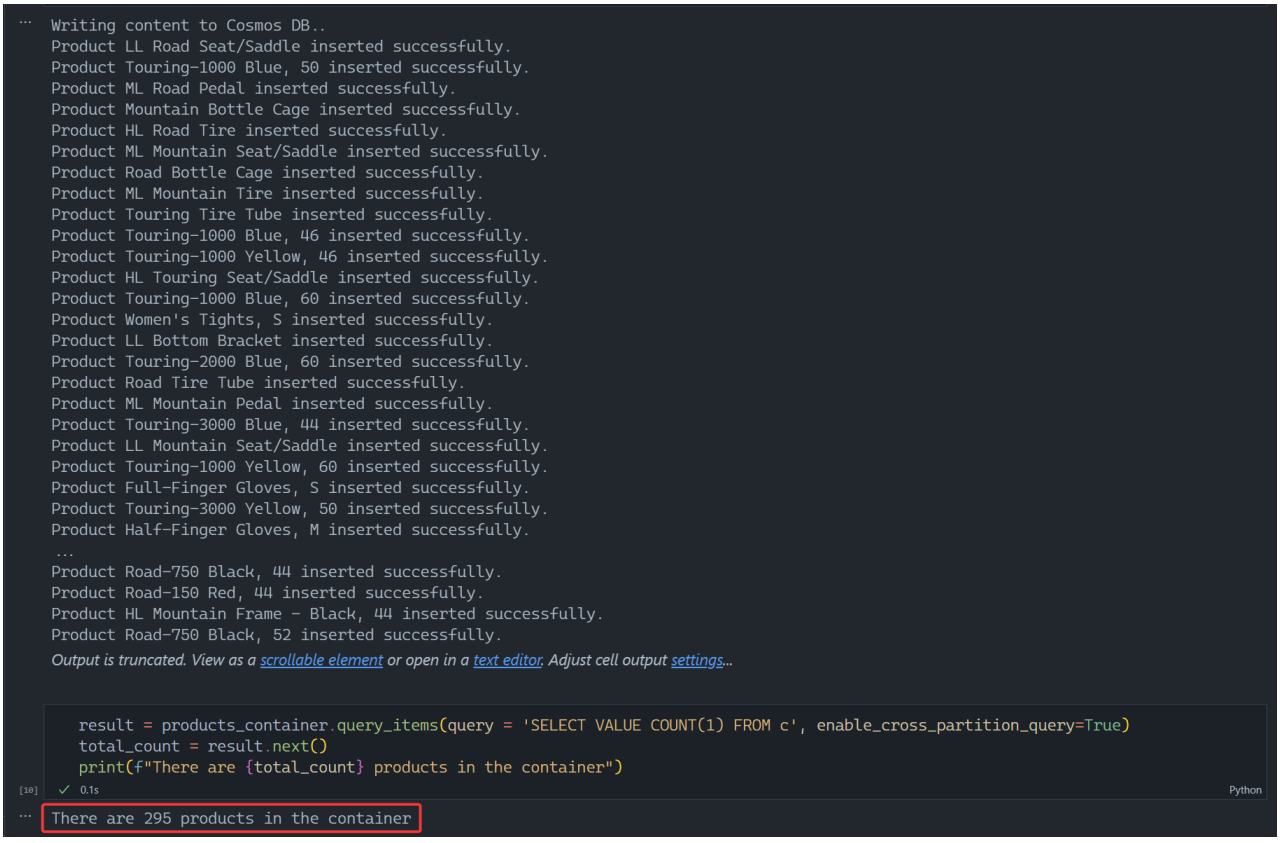
```
# Load products from json file
with open('product.json') as file:
  products = json.load(file)

# Write product and vector in separate documents (to make it easier for change
feed modifications)
print("Writing content to Cosmos DB..")
for p in products:
    productKey = str(uuid.uuid4())
    productJson = json.dumps(p)
    product = {
        "id": productKey,
        "categoryId": p["categoryId"],
        "categoryName": p["categoryName"],
        "sku": p["sku"],
        "name": p["name"],
        "description": p["description"],
        "price": p["price"],
        "tags": p["tags"],
        "type": "product"
    }
    vector = {
        "id": str(uuid.uuid4()),
        "productKey": productKey,
        "type": "vector",
        "embedding": generate_embeddings(openai_client, str(productJson))
    }
    container.create_item(product)
    container.create_item(vector)
    print(f"Product {p['name']} inserted successfully.")
```

```
result = container.query_items(query = "SELECT VALUE COUNT(1) FROM c WHERE
c.type = 'product'", enable_cross_partition_query=True)
```

```
total_count = result.next()
print(f"There are {total_count} products in the container")
```

In the menu bar, click on the “Run all” button to execute all cells at one.

At the end of the process, there should be 295 products in the products collection



Check ingested data

SELECT COUNT(1) FROM c WHERE c.type = 'product'