Labs Data & AI Innovation Day

Lab 2: Ingest data from csv

This document describes how to use a Python notebook to populate the Azure Cosmos DB for NoSQL ‘products’ container with the content of a csv 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

A screen shot of a computer

Description automatically generated

# Check ingested data

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

A screenshot of a computer

Description automatically generated