CosmosAIGraph

Americas Cosmos DB Global Black Belt (GBB) Team, Microsoft

Background

- Growth and demand for AI and Generative-AI accelerated in **2023**

- Specifically **Azure OpenAI**, **Chat-GPT**, and **Vector Search**

- Rise in "**Graph, AI-Driven Graph, and AI-Driven Knowledge Graph**" workloads
  - These are the target workloads for CosmosAIGraph. Operational in nature, not analytic

- The **AltGraph** solution, created in 2022:
  - Is a proven design that uses the Cosmos DB NoSQL API to solve "graph workloads"
  - https://devblogs.microsoft.com/cosmosdb/altgraph-graph-workloads-with-azure-cosmos-db-for-nosql/
  - However, it is lacking in some ways:
    - No Query Language, No Schema/Ontology, uses a Single Graph Replica
    - These are solved by the CosmosAIGraph solution

- These factors led to the creation of the **CosmosAIGraph** solution, 12/23-5/24

- Concurrent **MSR GraphRAG Whitepaper**
  - https://www.microsoft.com/en-us/research/blog/graphrag-unlocking-llm-discovery-on-narrative-private-data/

What is CosmosAIGraph?

- It's an **open-source** reusable design and set of **reference implementations**
- It is **_not_** a Microsoft, Azure, or Cosmos DB product
- It is built on the following:
  - **Cosmos DB Mongo vCore** PaaS service. Supports **Vector Search**
  - **Azure OpenAI** PaaS service
  - **RDF Technology** – triples, OWL ontologies (schemas), SPARQL queries
  - **In-memory graph** – inspired by LinkedIn, for faster performance and lower costs
  - **Python 3** – **rdflib**, FastAPI, Pydantic, pymongo, **Semantic Kernel**
  - Deployed to Azure Container Apps (ACA) with Bicep. Or AKS. Microservice design
  - Also supports **Generative AI**
  - Public repo: **aka.ms/caig** or https://github.com/cjoakim/CosmosAIGraph

- **RDF** is a standard industry solution for **Knowledge Graphs**
- Offers a simplified deployment architecture with just one DB: **Cosmos DB vCore**
- CosmosAIGraph introduces and implements the concept of **Hybrid RAG**

RDF Technology

- **Resource Description Framework (RDF)**
  - A set of W3C standards
  - Typically used for **Knowledge Graphs**

- **Web Ontology Language (OWL)**
  - An XML syntax to define the Classes and ObjectProperites of your graph
  - Think of these as the Entities and Relationships, or your graph **schema**

- **Triples**
  - A tuple of (subject, predicate, and object)
  - For example: (Cosmos DB → has_api → vCore).
  - A RDF graph consists of many of these simple triples, plus an ontology. Conceptually simple

- **SPARQL 1.1** – query language. Similar to SQL. Simpler than Gremlin & Cypher

- **rdflib** – A python library that implements a mutable in-memory RDF graph

CosmosAIGraph – Graph Design and Development Steps

- **Design and load your Cosmos DB Mongo vCore account**
  - Use typical NoSQL design patterns, JSON documents
  - No special "triples" documents are required, unlike AltGraph

- **Define your Graph Schema**
  - It's an XML syntax. Define Classes, attributes w/datatypes, and relationships

- **Load the in-memory RDF database from Cosmos DB**
  - Read only the necessary attributes of the Cosmos DB documents
  - The in-memory graph is mutable, but is static in the reference applications
  - Alternatively, in a dev environment, load the graph fro a "triples file" (i.e. – *.nt)
  - ***The graph is strictly an in-memory concept; it doesn't exist on disk***

- **Query the in-memory RDF database with SPARQL**
  - It's very fast because it's in-memory
  - It's low cost, because no vectorization is involved
  - The SPARQL can optionally be generated with GenAI & Azure OpenAI. This is a great learning tool

Vector Search – Development Steps

- **"Vectorize" your data**
  - Use the **Azure OpenAI SDK** with your Azure OpenAI PaaS service
  - Pass in a **text** value, receive back an "**embedding**" – an array of 1536 floats
  - The embedding captures the **semantic meaning** of the text
  - An embedding looks verbose, but it is a very efficient data structure
  - Use the **text-embedding-ada-002** model within Azure OpenAI
  - Store that vector, along with document context, in your **Cosmos DB vCore** database
  - https://learn.microsoft.com/en-us/azure/ai-services/openai/tutorials/embeddings

- **Vector Search**
  - Implement a vector index
    - https://learn.microsoft.com/en-us/azure/cosmos-db/mongodb/vcore/vector-search
  - Pass in a **vector** (i.e. – embedding) as the argument to a search in the database
  - Receive n-number of documents which match the given vector. Semantic similarity
  - Can return more **relevant** results vs traditional search engines
  - **Filtered** vector search is currently in preview

CosmosAIGraph – Code Generation

```powershell
 1   # This PowerShell script demonstrates command-line
 2   # Code-generation in CosmosAIGraph.
 3
 4   # Generate an Ontology *.owl file from input data metadata.
 5   # creates file tmp/generated.owl
 6
 7   python main.py generate_owl `
 8       meta/vertex_signatures_imdb.json `
 9       meta/edge_signatures_imdb.json `
10       http://cosmosdb.com/imdb
11
12   # Generate class RdflibTriplesBuilder that loads the rdflib graph
13   # from your Cosmos DB Documents.
14   # creates file: tmp/rdflib_triples_builder.py
15
16   python main.py generate_rdflib_triples_builder `
17       meta/vertex_signatures_imdb.json
18
```

CosmosAIGraph – Generative AI Use-Cases

- **Infer the "RAG Strategy" from User natural-language**
  - Implements "LUIS-like" utterances, entities, and intents with OpenAI and a prompt
  - See class StrategyBuilder#determine

- **Generate SPARQL Queries from User natural-language**
  - Uses the OWL ontology as a system prompt
  - See AIService#generate_sparql_from_user_prompt

- Generate Python code to wrangle your input data into Cosmos DB Documents
  - Infer input schemas from CSV headers, JSON structure
  - Targets the defined OWL ontology
  - This is currently a work-in-progress

- You may have additional use-cases for your CosmosAIGraph application

- **A note on Generative AI Skills:**
- See the O'Reilly Media "**AI-Assisted Programming**" book, 2024
- CEO: "Now when I consider hiring a developer, I want to know how they leverage AI.".
- "Satya Nadella" …. "they are becoming standard issue for any developer…".

Generative AI in CosmosAIGraph – Web App Example

- Uses Azure OpenAI, gpt-4, and the "RAG" pattern

- The OWL ontology is the "System Prompt"

- The Natural Language is the "User Prompt"

- The result is a working SPARQL query

Generate SPARQL Console

Enter a Natural Language Query:

What are the dependencies of the 'pypi' type of library named 'flask'?

Generate SPARQL from Natural Language

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX : <http://cosmosdb.com/caig#>
SELECT ?dependency
WHERE {
    ?lib :ln 'flask' .
    ?lib :lt 'pypi' .
    ?lib :uses_lib ?dependency .
}
```

Execute SPARQL Query

CosmosAIGraph Hybrid RAG Approach

User Inputs in an example AI Conversation:

1. What is the Python Flask Library	DB RAG
2. What are its dependences	Graph RAG
3. What are the alternatives that use async processing	Vector RAG
4. Who is the author	DB RAG
5. What other libraries did she write	Graph Rag
6. Display a graph of all her libraries and their dependencies	Graph RAG

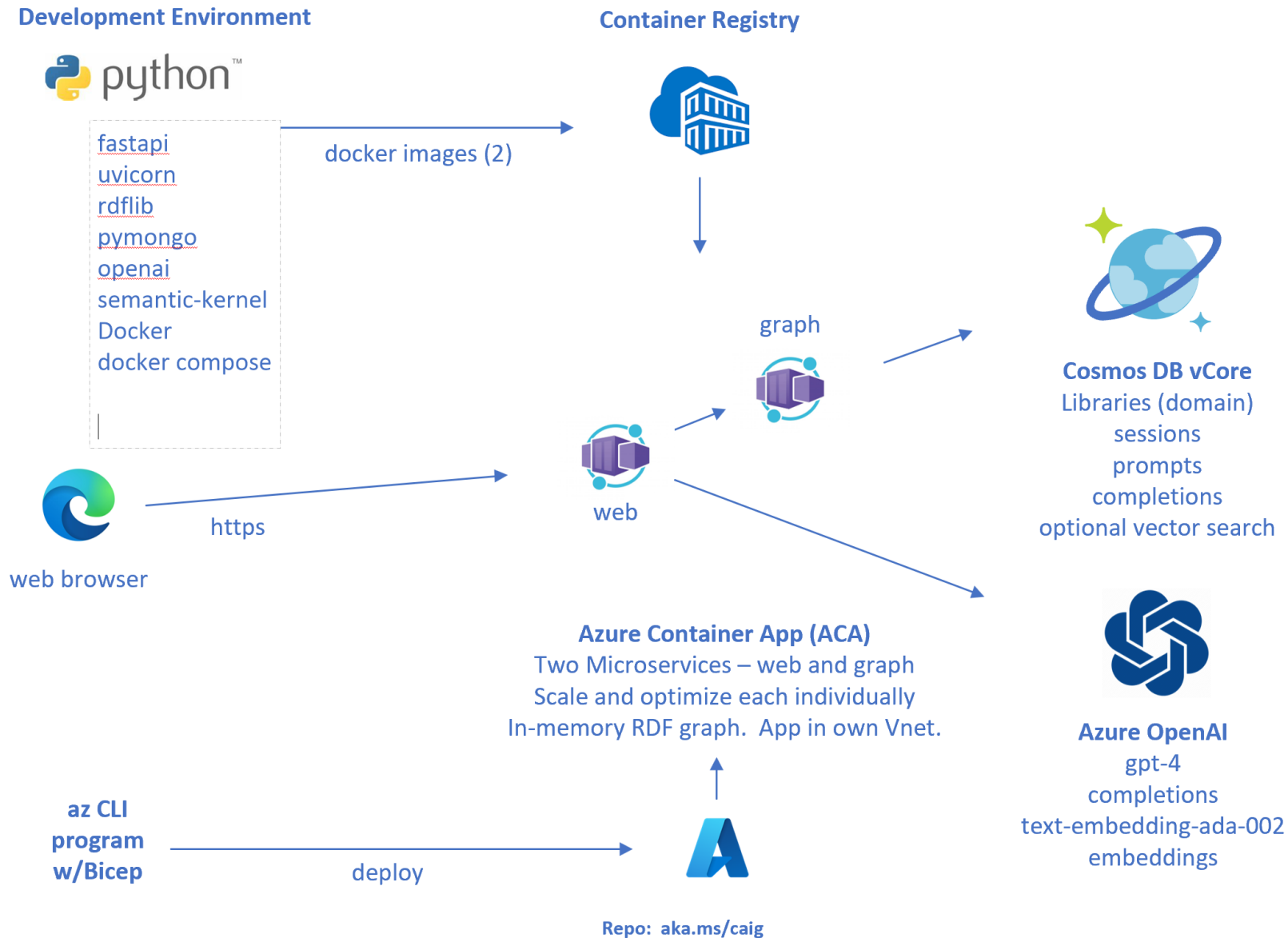**In-Memory RDF Graph Database, loaded from vCore**

Cosmos DB vCore w/Vector data and search

Application Logic:
- Determine Intent & RAG Strategy from User Intent
- Identifiy Entities
- Generate SPARQL query if Graph RAG
- Generate vCore query if DB RAG
- Vectorize user input if Vector RAG
- Execute the DB query to get Docs List
- Fetch Documents per List from Cosmos DB
- Craft the Prompt with the Document RAG Data
- Invoke the LLM with input & RAG data in prompt
- Parse the LLM response, and present in the web UI

User Input

RAG Data

Prompt

LLM

Output JSON, TXT, etc

Repo: aka.ms/caig

CosmosAIGraph Architecture

Development Environment

Container Registry

fastapi
uvicorn
rdflib
pymongo
openai
semantic-kernel
Docker
docker compose

docker images (2)

graph

Cosmos DB vCore
Libraries (domain)
sessions
prompts
completions
optional vector search

web

https

web browser

Azure Container App (ACA)
Two Microservices – web and graph
Scale and optimize each individually
In-memory RDF graph. App in own Vnet.

Azure OpenAI
gpt-4
completions
text-embedding-ada-002
embeddings

**az CLI
program
w/Bicep**

deploy

Repo: aka.ms/caig

Web Application UI Screen Shots

SPARQL Console

Enter a SPARQL query:

```
PREFIX c: <http://cosmosdb.com/caig#>
SELECT ?used_lib
WHERE {
    <http://cosmosdb.com/caig/pypi_flask> c:uses_lib ?used_lib .
}
LIMIT 10
```

Enter a library type, library name, and a depth integer for a Bill-of-Materials query:

pypi flask 3

☐ Use Cache

Submit

The SPARQL Console Page introduces and demonstrates queries vs the in-memory RDF graph

Graph Visualizations with the D3.js JavaScript library in the SPARQL Console page

Generate SPARQL Console

Enter a Natural Language Query:

What are the dependencies of the 'pypi' type of library named 'flask'?

Generate SPARQL from Natural Language

SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX : <http://cosmosdb.com/caig#>
SELECT ?dependency
WHERE {
    ?lib :ln 'flask' .
    ?lib :lt 'pypi' .
    ?lib :uses_lib ?dependency .
} LIMIT 100
```

Execute SPARQL Query

Generate SPARQL Console demonstrates the power of generative AI

CosmosAIGraph

Vector Search Console

Select a Libtype

pypi

Enter a library name or 'text: your text...'

flask

Show Embeddings

Y or N (default)

Search

Vector Search Results

```
{
  "libtype": "pypi",
  "name": "flask",
  "count": 10,
  "doc": {
    "_id": "661ef2151f4ac99c9428e912",
    "classifiers": [
      "development status :: 5 - production/stable",
      "environment :: web environment",
      "framework :: flask",
      "intended audience :: developers",
      "license :: osi approved :: bsd license",
      "operating system :: os independent",
      "programming language :: python",
      "topic :: internet :: www/http :: dynamic content",
      "topic :: internet :: www/http :: wsgi",
      "topic :: internet :: www/http :: wsgi :: application",
      "topic :: software development :: libraries :: application frameworks"
```

Vector Search Page demonstration page, executes vector search vs Cosmos DB vCore

Conversational AI Console

Conversation ID: 66e41698-2b30-4aad-a634-eb6c6fb12308

lookup PyPi Flask

Flask is a lightweight WSGI web application framework. It is designed to make getting started quick and easy, with the ability to scale up to complex applications. It began as a simple wrapper around Werkzeug and Jinja and has become one of the most popular Python web application frameworks. Flask offers suggestions, but doesn't enforce any dependencies or project layout. It is up to the developer to choose the tools and libraries they want to use. There are many extensions available to Flask which can add application features as if they were implemented in Flask itself. For the most up-to-date information, including the latest version, how to install it, and its documentation, it's best to visit the Flask page on PyPi (the Python Package Index) or its official website.

prompt tokens: 166, completion tokens: 155, total tokens: 321, rag strategy: db_search

what are some asynch alternatives

For asynchronous web development in Python, there are several alternatives to Flask, which is primarily synchronous. Some of the popular asynchronous frameworks include: 1. **FastAPI**: A modern, fast (high-performance) web framework for building APIs with Python 3.7+ based on standard Python type hints. It's designed to be easy to use while ensuring high performance. FastAPI integrates well with asynchronous databases and is built on top of Starlette for the web parts. 2. **Starlette**: A lightweight ASGI framework/toolkit, which is ideal for building high-performance asyncio services. It is the foundation upon which FastAPI is built. Starlette is versatile and can be used for building microservices as well as larger web applications. 3. **Sanic**: An asynchronous web framework that was built specifically for quick HTTP responses via asynchronous request handling. It's similar to Flask but designed to be fast by allowing the use of async/await syntax. 4. **Quart**: A Python ASGI web microframework. It is intended to provide an easy way to get started with asynchronous programming with a Flask-like API. Quart supports the Flask API and async-await syntax, making it possible to use Flask extensions with modifications. 5. **Tornado**: A Python web framework and asynchronous networking library. It is designed to handle asynchronous I/O, which makes it suitable for long polling, WebSockets, and other applications that require a long-lived connection to each user. 6. **AIOHTTP**: An asynchronous HTTP client/server framework. It supports both client and server-side web programming and is built on top of asyncio, Python's standard asynchronous I/O framework. It provides a request object and router similar to those found in Flask but works asynchronously. These frameworks leverage Python's asyncio library and provide a more scalable architecture for handling large numbers of simultaneous connections, which is beneficial for real-time web applications, high-load applications, and microservices.

prompt tokens: 467, completion tokens: 387, total tokens: 854, rag strategy: vector_search

Enter your natural-language query:

enter your query here

[Continue]   { show/hide conversation json }

Conversational AI Page demonstrates Hybrid RAG approach and conversation history

CosmosAIGraph - Summary

- CosmosAIGraph provides a cost-effective performant solution for:
  - **Graph Workloads (with little or no AI)**
  - **AI-Driven Graph Workloads**
  - **AI-Driven Knowledge Graph Workloads**
  - **AI Chat Applications**
- Supports **Generative AI**
- Uses the accurate and cost-effective **Hybrid RAG** Pattern
- Open-Standards Based – RDF, OWL Ontologies, SPARQL queries, Python
- Robust and Replicated Deployments with Docker Containers and Azure Container Apps
- Reference applications and documentation assist and guide users in adoption
- Public GitHub Reposiory: aka.ms/caig

Thank you!   Questions?