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 workload 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 an 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

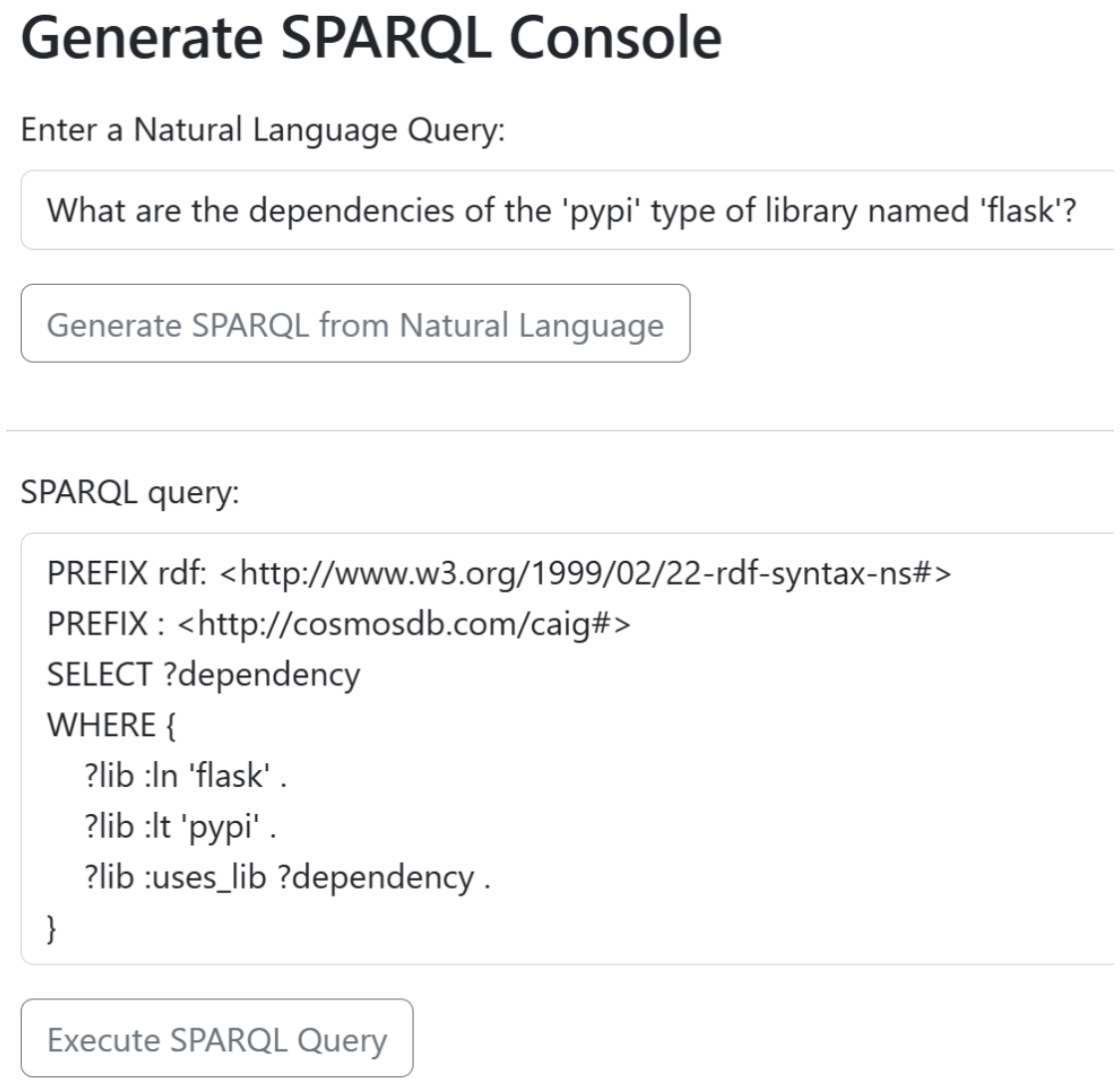
Vector Search – Development Steps

- **“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

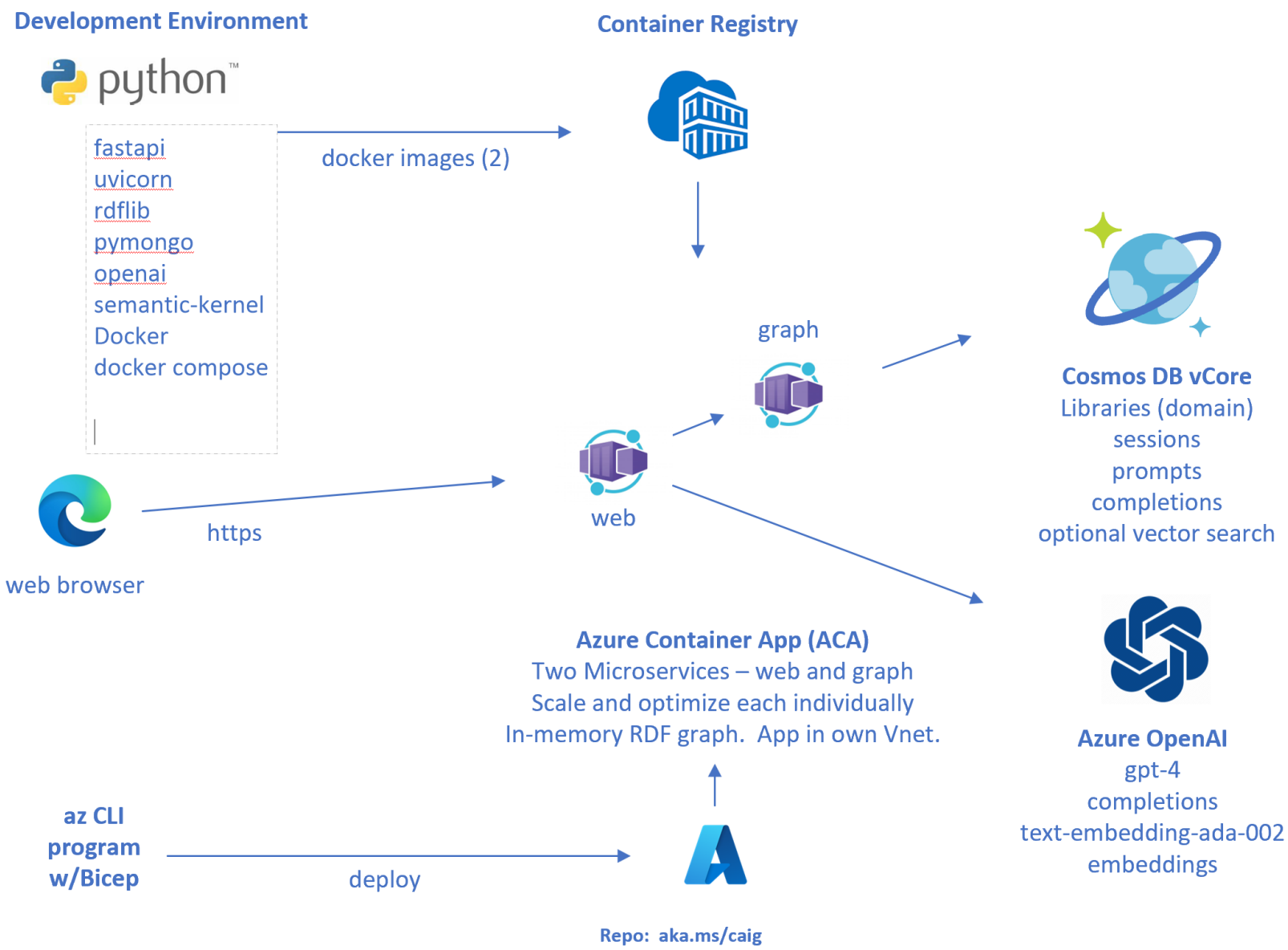
- **Vector Search**
  - Implement a vector index
    - https://learn.microsoft.com/en-us/azure/cosmos-db/mongodb/vcore/vector-search
  - Pass in a **vector** (i.e. – embedding) as the argument to a search in the database
  - Receive n-number of documents which match the given vector. Semantic similarity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

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



CosmosAIGraph Architecture

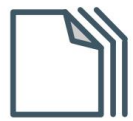


CosmosAIGraph Hybrid RAG Approach

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

User Input



RAG Data

In-Memory RDF Graph Database, loaded from vCore

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

Prompt

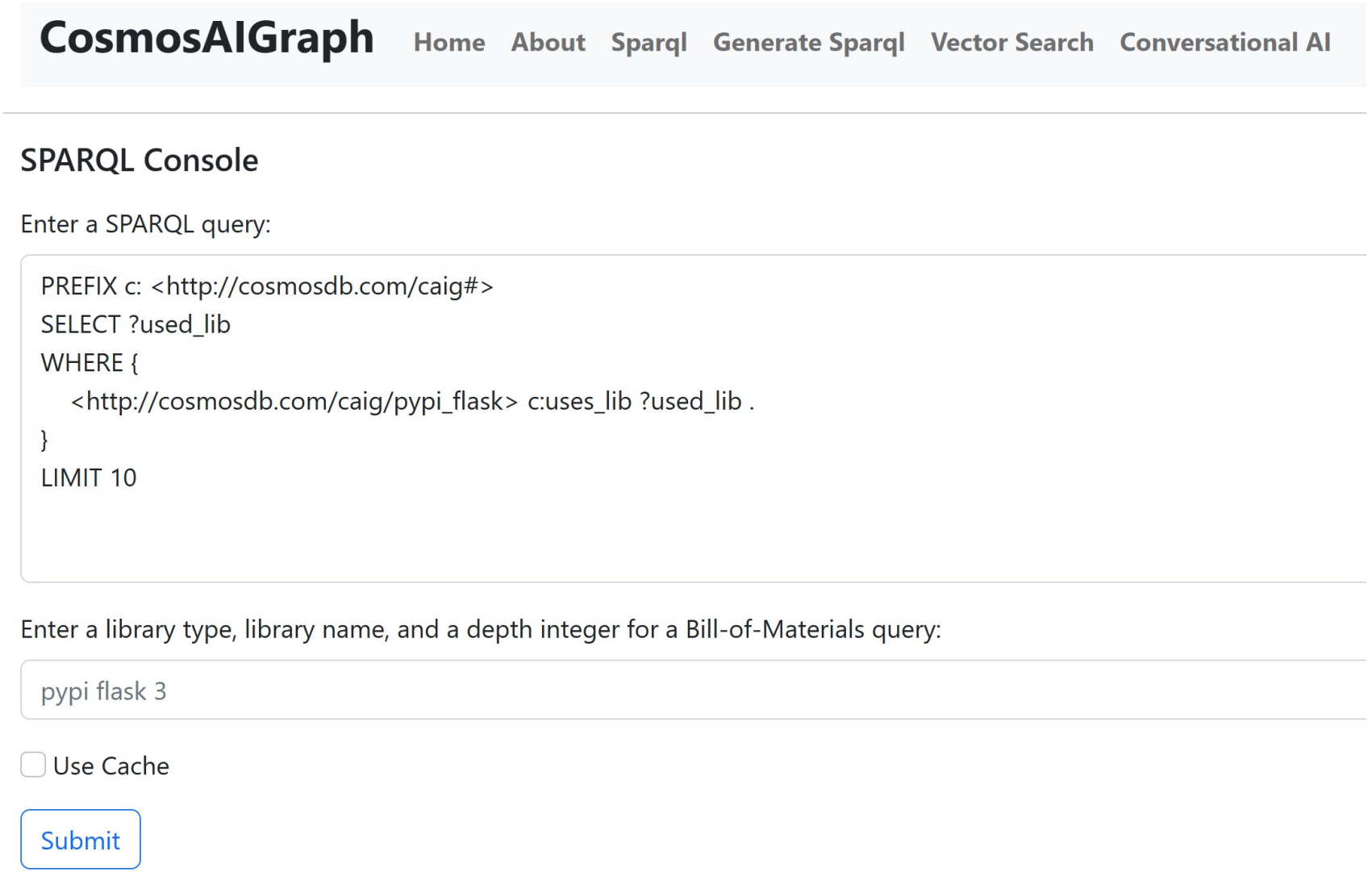
Cosmos DB vCore w/Vector data and search

LLM

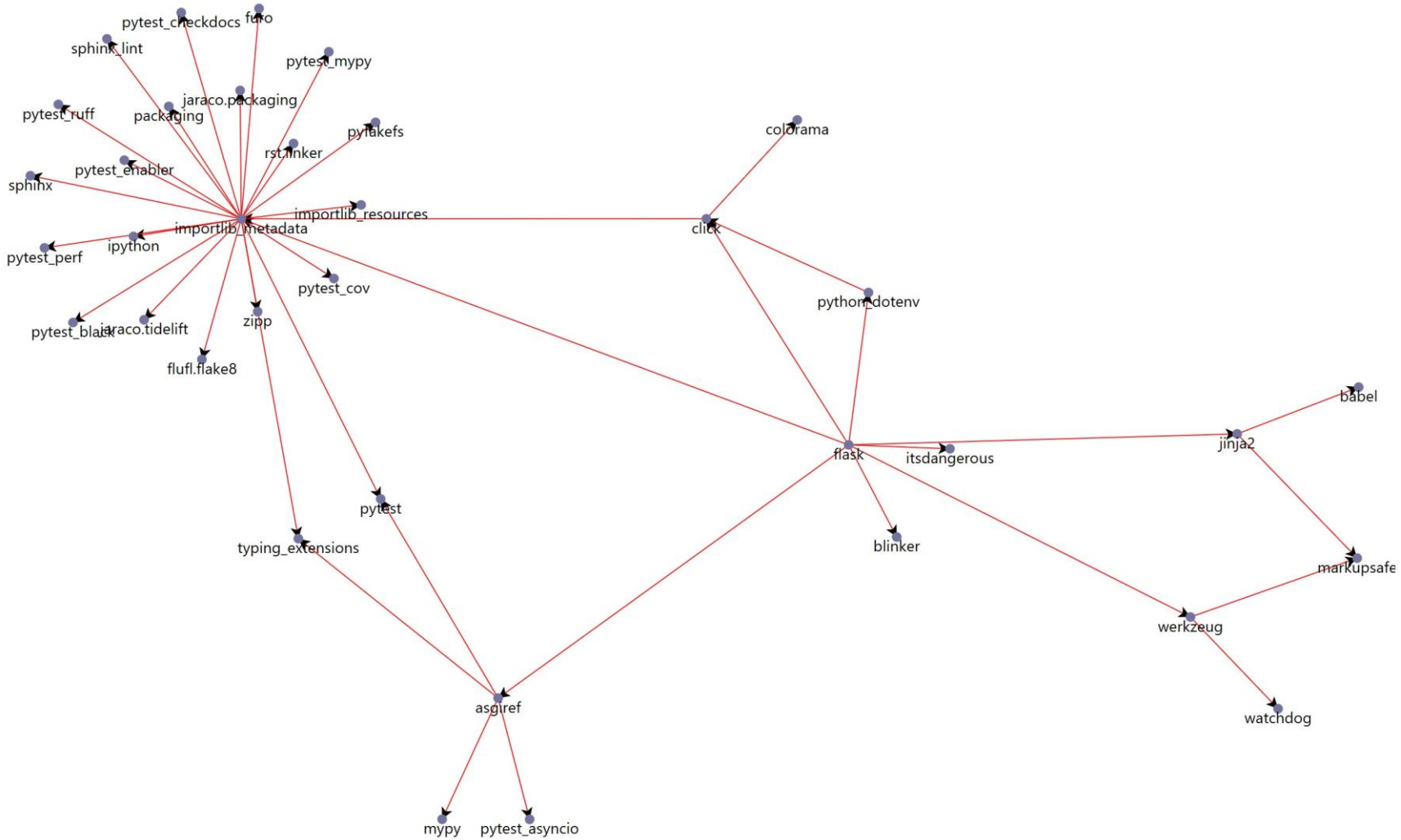
Output
JSON, TXT, etc

Repo: aka.ms/caig

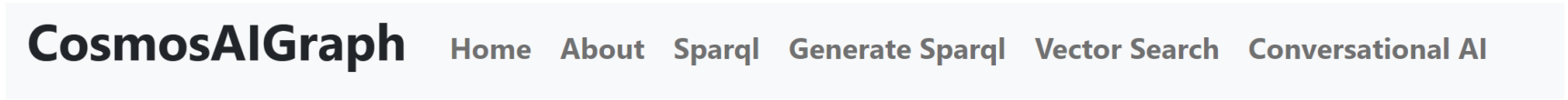
Web Application UI Screen Shots



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'?

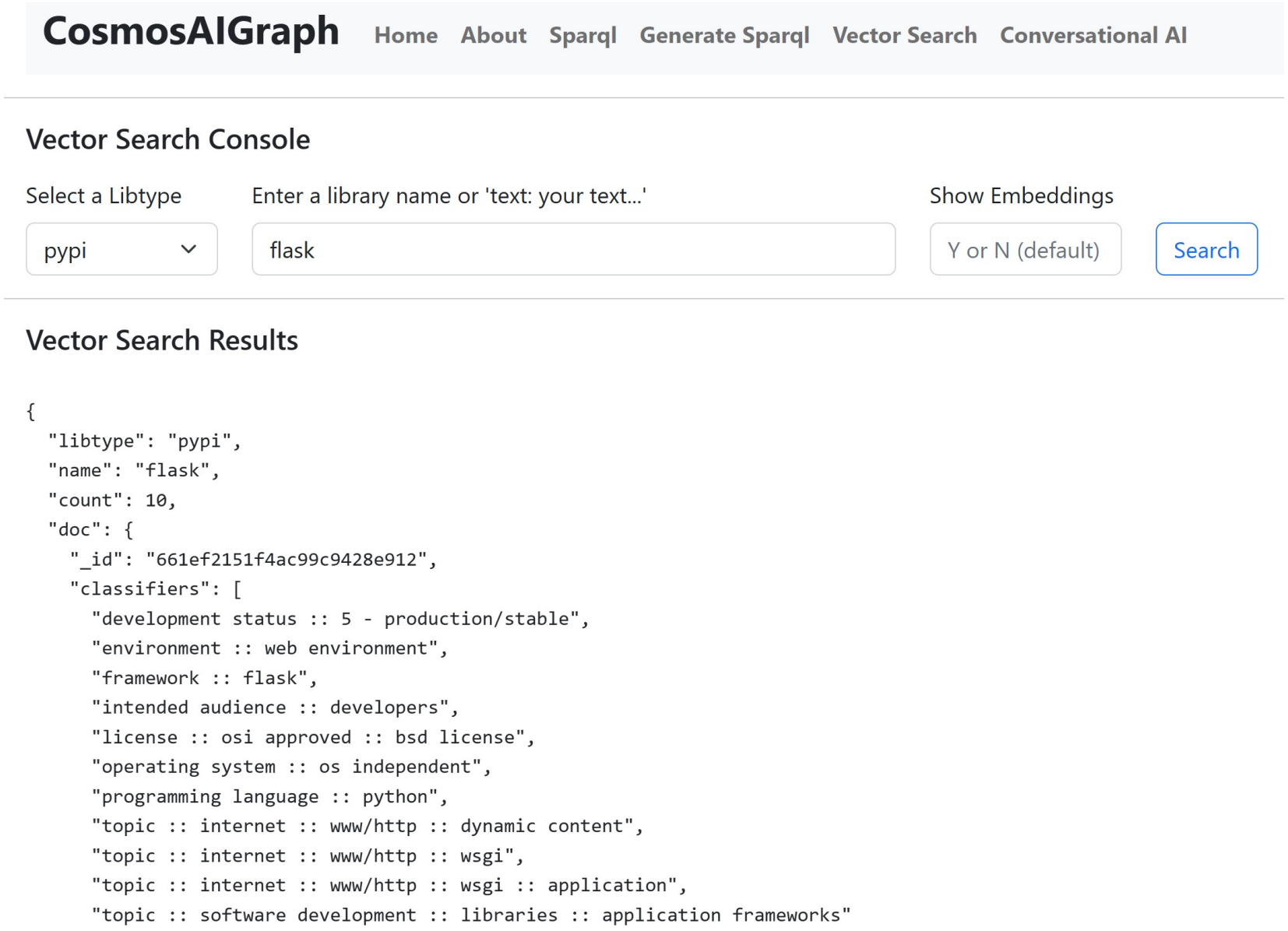
Generate SPARQL from Natural Language

SPARQL query:

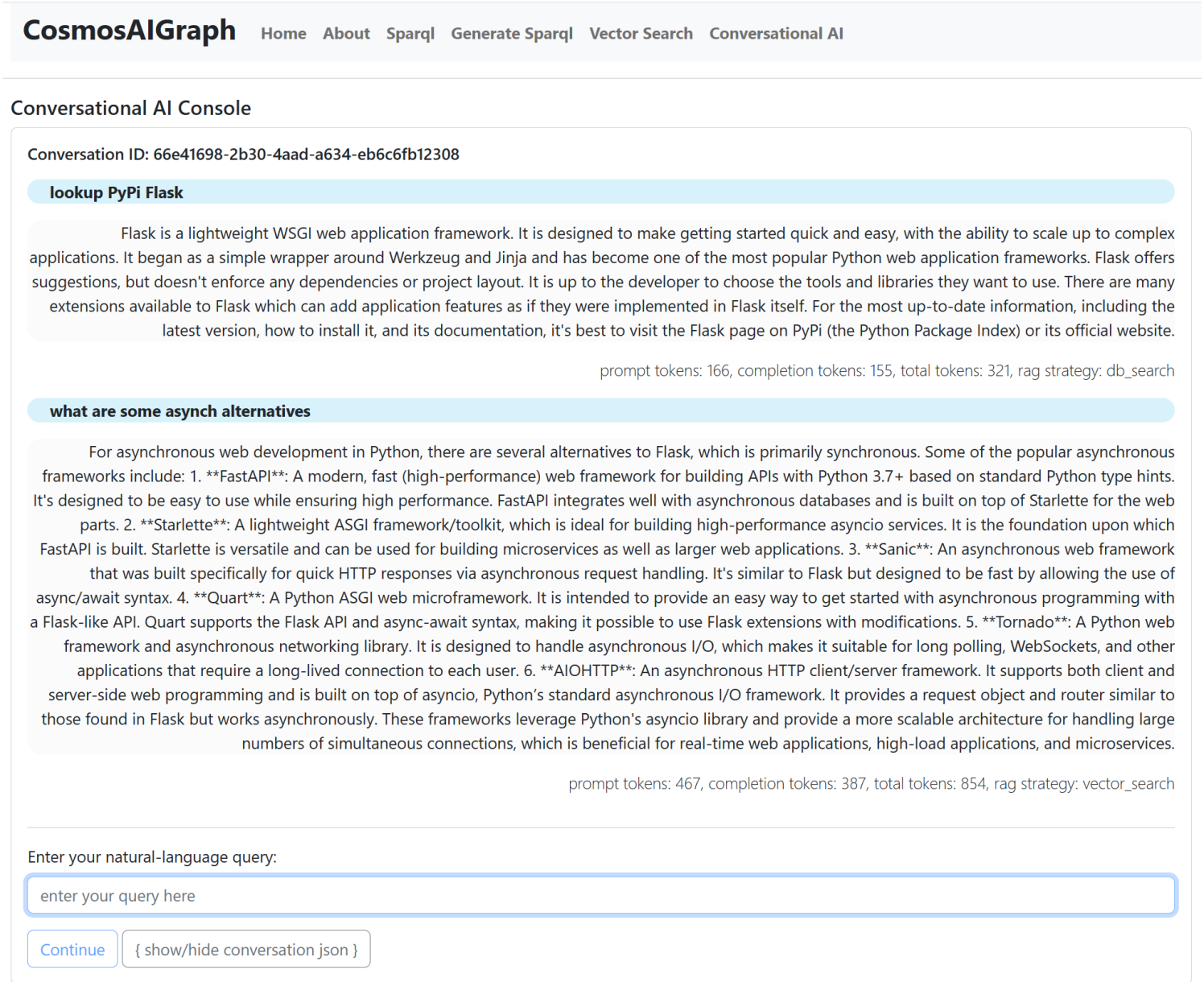
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX : <http://cosmosdb.com/caig#>
SELECT ?dependency
WHERE {
    ?lib :ln 'flask' .
    ?lib :lt 'pypi' .
    ?lib :uses_lib ?dependency .
} LIMIT 100
```

Execute SPARQL Query

Generate SPARQL Console demonstrates the power of generative AI



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



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?**