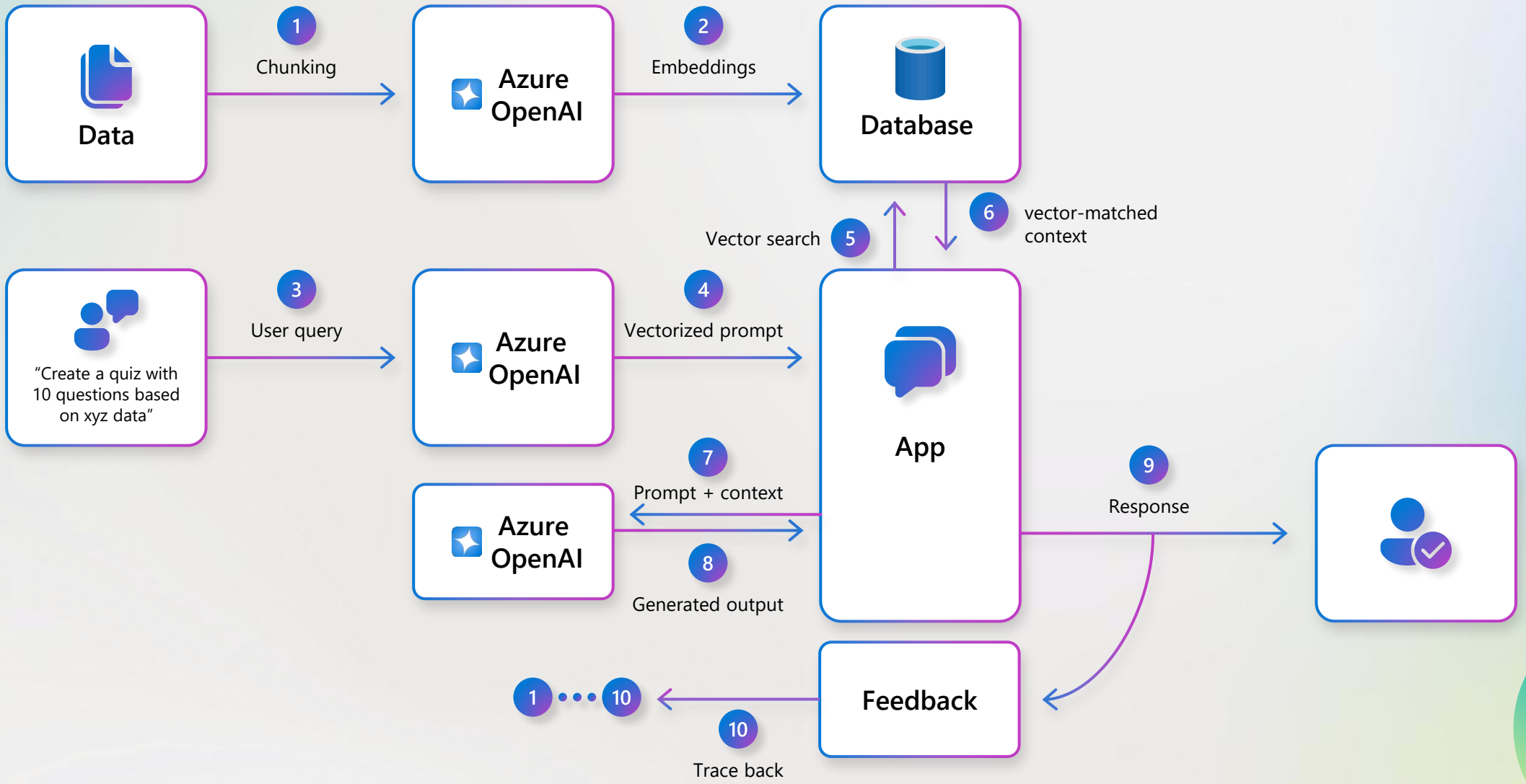
Microsoft Azure

OmniRAG Pattern and CosmosAIGraph Accelerator

Basic RAG (Retrieval Augmented Generation)



Examples of problems in AI Apps*

You: how many bikes of each type do you have in your shop?

Bot: there are two bikes – one mountain bike and one road bike

Too few records and no aggregates fetched from the vector store to answer the question

You: what is total order amount for the road bikes with white frame?

Bot: The records provided don't contain this information

No aggregates fetched from vector store

You: how much was my last transaction?

Bot: Looks like your last purchase order of $100 was made a year ago

Stale and unsorted data fetched from vector store

* - Using canonical CosmicWorks dataset

Basic RAG is NEVER enough!

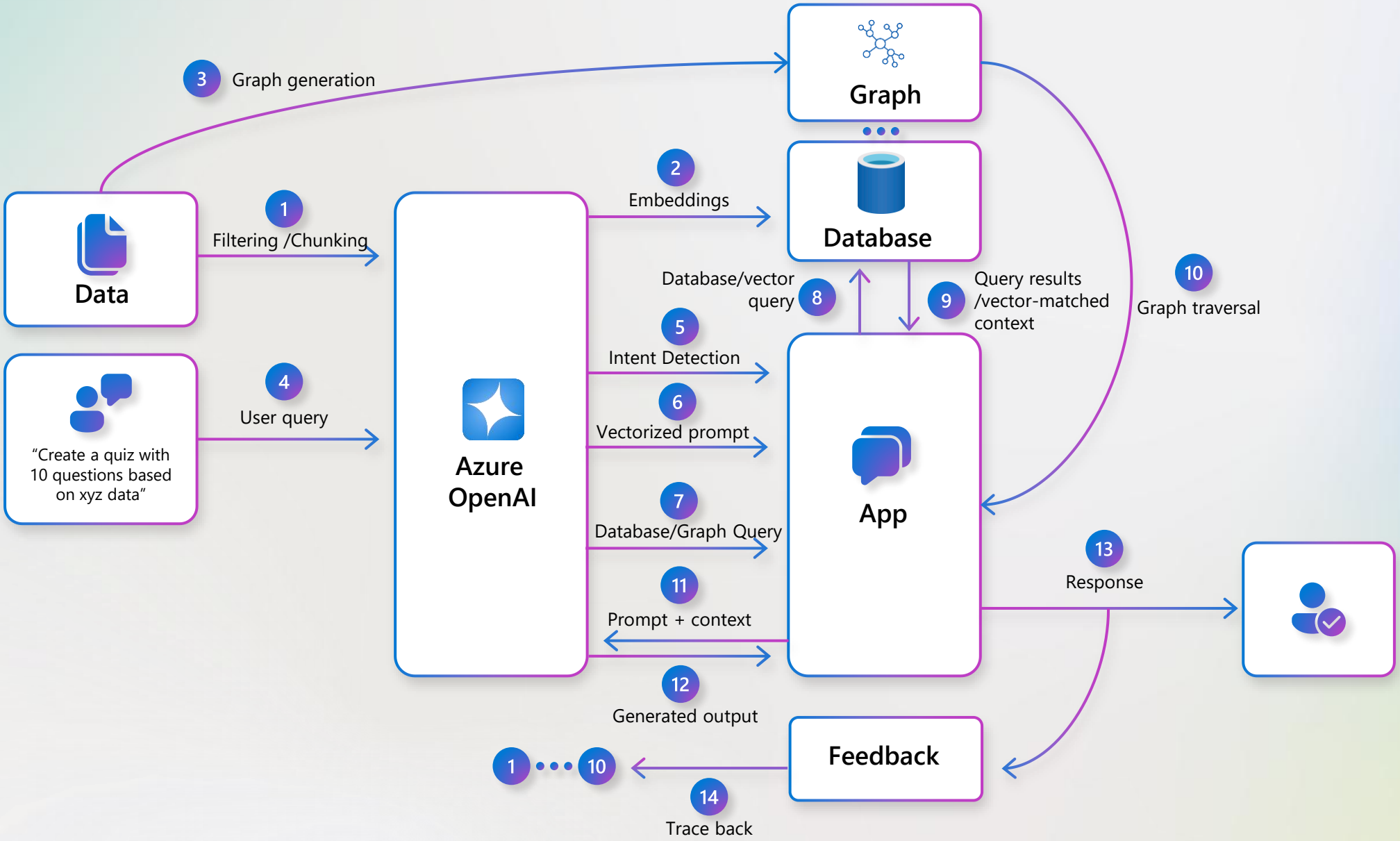
Only contains summary of the original source data without aggregates-> lower accuracy of AI solution

Leads to using all available model context window for (possibly irrelevant) custom data -> higher cost for inferencing

Requires generating embeddings on the entire dataset -> higher cost for compute/storage

Requires pre-population of semantic index (facets/column-level stats) by guessing what the questions will be -> lower accuracy, higher costs for search engine

OmniRAG Pattern



OmniRAG Core Tenets

- **Omni-source with data virtualization**
  - Not limited to vector store, utilizing ALL data sources that can bring value to the context for AI
  - Use data wherever it is, in the original format, minimizing data movement and transformation
- **Knowledge graph**
  - Contains entities/relationships from existing data to make it readily available for AI to reason over (along with original data)
- **User intent detection**
  - Allows automatic routing of user's query to the right source, leverages AI
- **NL2Query**
  - Convert user query to the source's query language using simple utterance analysis and/or AI
- **Session analytics**
  - Required to fine-tune golden dataset of questions+intent so intent detection improves. Could be used for semantic cache generation/curation

Basic RAG vs GraphRAG vs OmniRAG

Rough comparison

	Basic RAG	GraphRAG	OmniRAG (Graph+Vector+Raw)
Sources	Vector	Graph	Graph, Vector, Database, etc.
Intent detection	No	No	Yes
Graph generation	Generic (embeddings)	Generic (costly/takes long)	Custom (with code)
NL2Query	No	Yes (proprietary, graph only)	Yes (built-in + custom code)
Scalable	No	No	Yes

CosmosAIGraph as OmniRAG implementation

What is it providing?

- OmniRAG pattern reference implementation using knowledge graph/vector database/raw database
- Semi-automatic knowledge graph generation from existing data store (Cosmos DB, flat files, etc.)
- User intent detection (strategy builder) powered by AI
- Scalable knowledge graph database/index with in-memory model (such as RDF) and fast persistent store for graph and raw data (Cosmos DB)
- “Natural language to graph query” generator powered by AI with optional feedback loop

CosmosAIGraph is based on RDF Technologies

- **Resource Description Framework (RDF)**
  - A set of W3C standards
  - Typically used for Knowledge Graphs
- **Turtle (TTL)**
  - A concise YAML-like syntax to define:
    - The ontology (schema consisting of Classes and ObjectProperties),
    - The data of the graph (entities and relationships or nodes and edges)
- **Triples**
  - The **building block** of the graph- a tuple of (subject, predicate, object)
    - For example: (customer → purchased → product)
  - An RDF graph consists of many of these simple triples, conforming to the ontology
- **SPARQL 1.1**
  - A **query language** similar to SQL
- **Apache Jena**
  - An in-memory indexed RDF store with support of SPARQL, SHACL, TTL, etc.

Graph Search in CosmosAIGraph

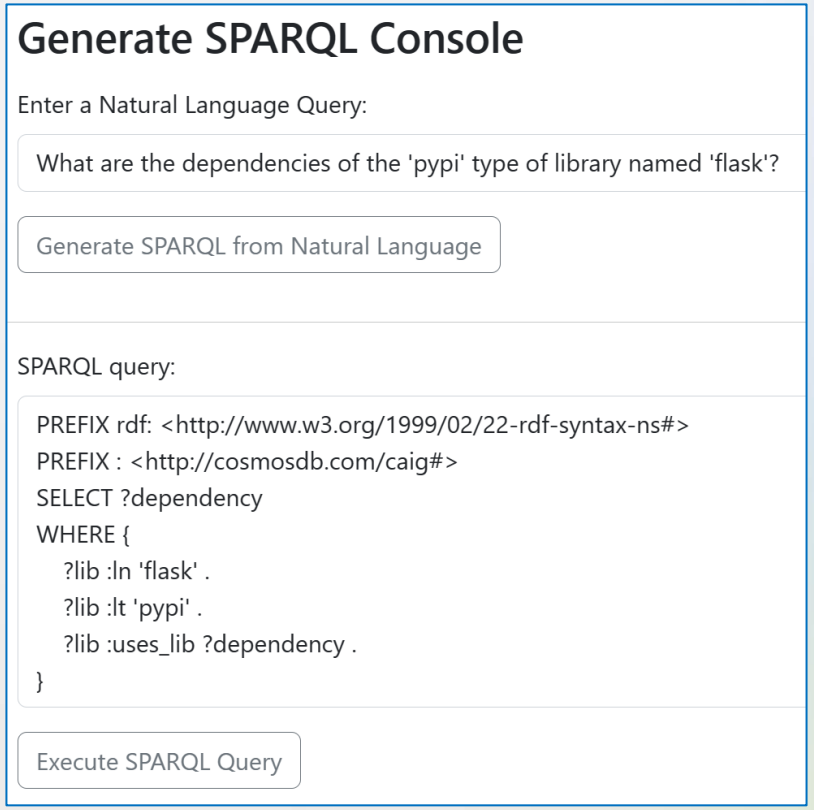
- Design and load your Cosmos DB for NoSQL account
  - Use typical NoSQL design patterns
- Define your ontology (graph schema) in TTL
- Load the in-memory RDF database
  - Read only the necessary attributes of the Cosmos DB documents, create one or many triples for each one of them
  - The in-memory graph is mutable, changes can be propagated bi-directionally between Cosmos DB and the graph
- Query the in-memory RDF store with SPARQL
  - It's very fast because it's in memory
  - It's low cost, because no vectorization is needed, and only open-source libraries are used

Vector Search in CosmosAIGraph

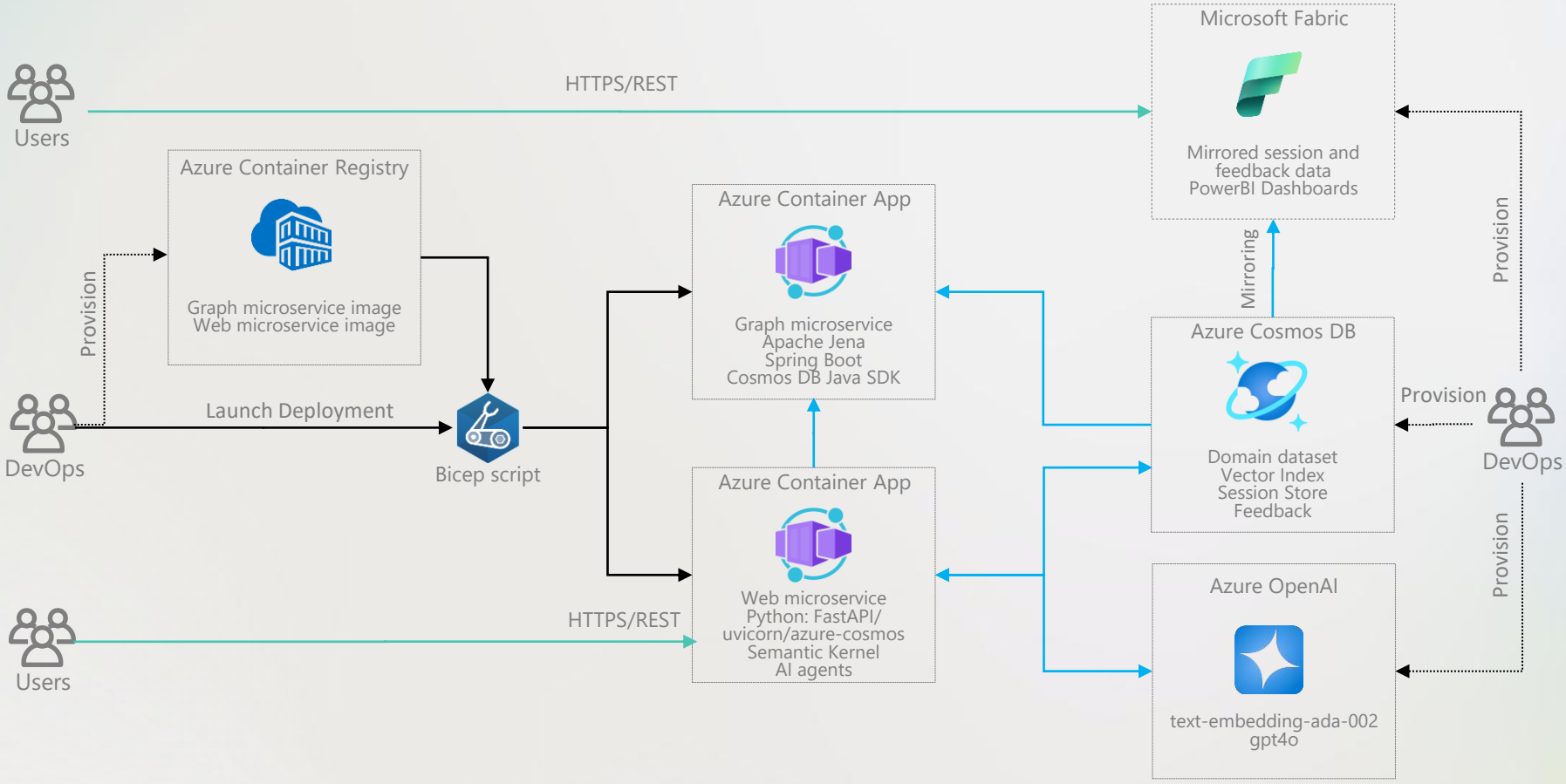
- "Vectorize" your data
  - Use the Azure OpenAI SDK with your Azure OpenAI service
  - Pass in a text value, receive back an "embedding" – an array of 1,536 floats
  - The embedding captures the semantic meaning of the text
  - Use the text-embedding-ada-002 or similar model in Azure OpenAI
  - Store that vector, along with document contents, in your vector database
- Vector Search
  - Pass in a vectorized query/command (i.e. embedding) as the argument to a search in the database
  - Receive N number of documents which match the given vector with semantic similarity

NL2SPARQL in CosmosAIGraph

- Uses GPT-4o in Azure OpenAI
- The OWL ontology is the “System Prompt”
- The Natural Language is the “User Prompt”
- The result is a working SPARQL query



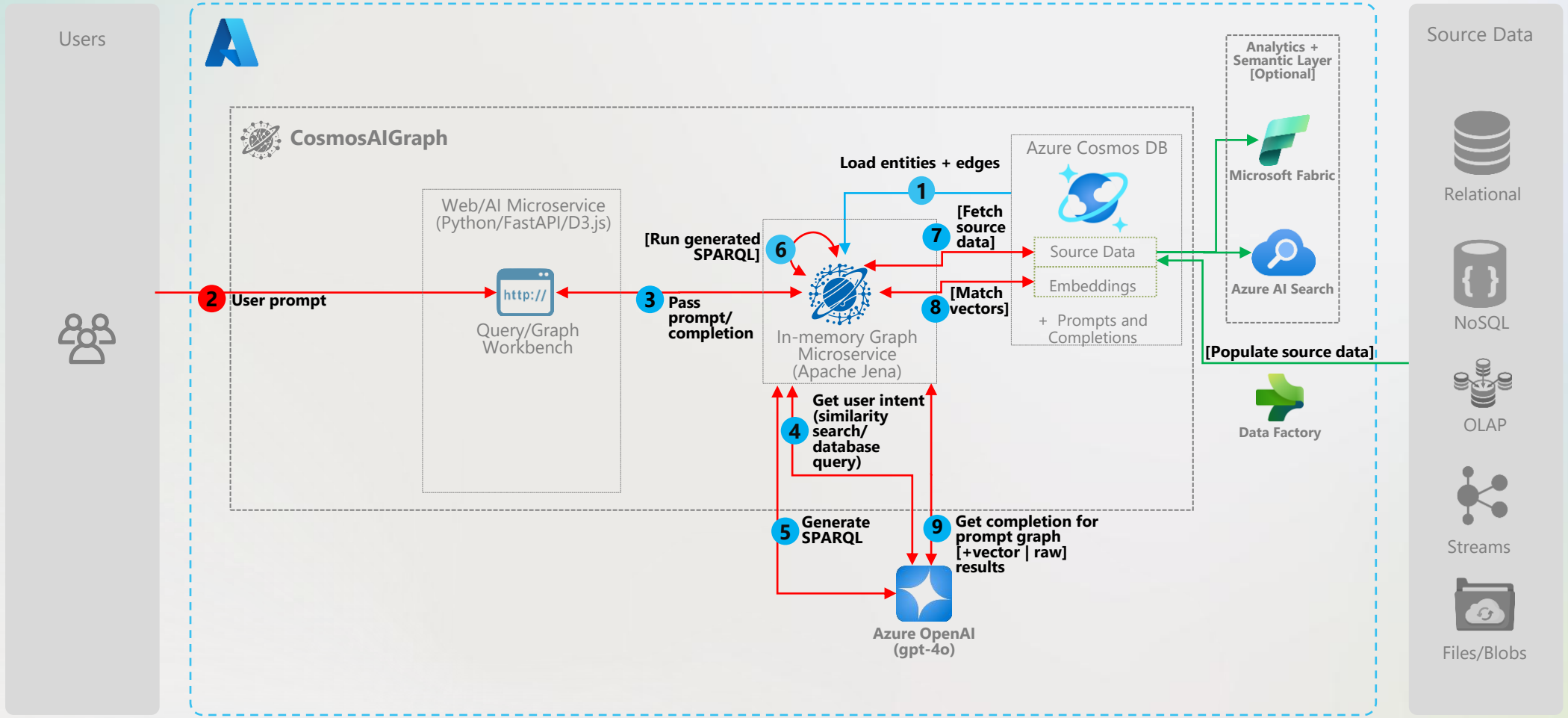
CosmosAIGraph Deployment Architecture



Legend: Pre-provisioning | Deployment | Runtime data movement | End-user interactions

CosmosAIGraph Solution Architecture

Public GitHub repo: aka.ms/caig



Legend: Primary Flow (hot path) | Background Flow (cold path) | Data Ingestion/Mirroring | [Optional actions] | 1 User actions | 2 Service actions

CosmosAIGraph Summary

- It's a reusable reference design and implementation. **It's not a product**
- It is built on the following:
  - ✓ **Cosmos DB for NoSQL** service, natively supporting vector search (DiskANN)
  - ✓ **RDF** technologies – triples, ontologies, SPARQL queries
  - ✓ **Apache Jena** – in-memory indexed RDF graph for fast performance and low costs
  - ✓ **Python 3** – fastapi, pydantic, azure-cosmos or pymongo, semantic-kernel libraries
  - ✓ **Azure OpenAI** service with industry-leading LLMs
  - ✓ **Semantic Kernel** – for pluggable and extensible orchestration
- Designed as set of microservices
- Deployed to Azure Container Apps with Bicep

Call to Action!

- ✓ Go to public repo aka.ms/caig and explore
- ✓ Deploy it to your Azure subscription
- ✓ Give us your feedback on OmniRAG and CosmosAIGraph!