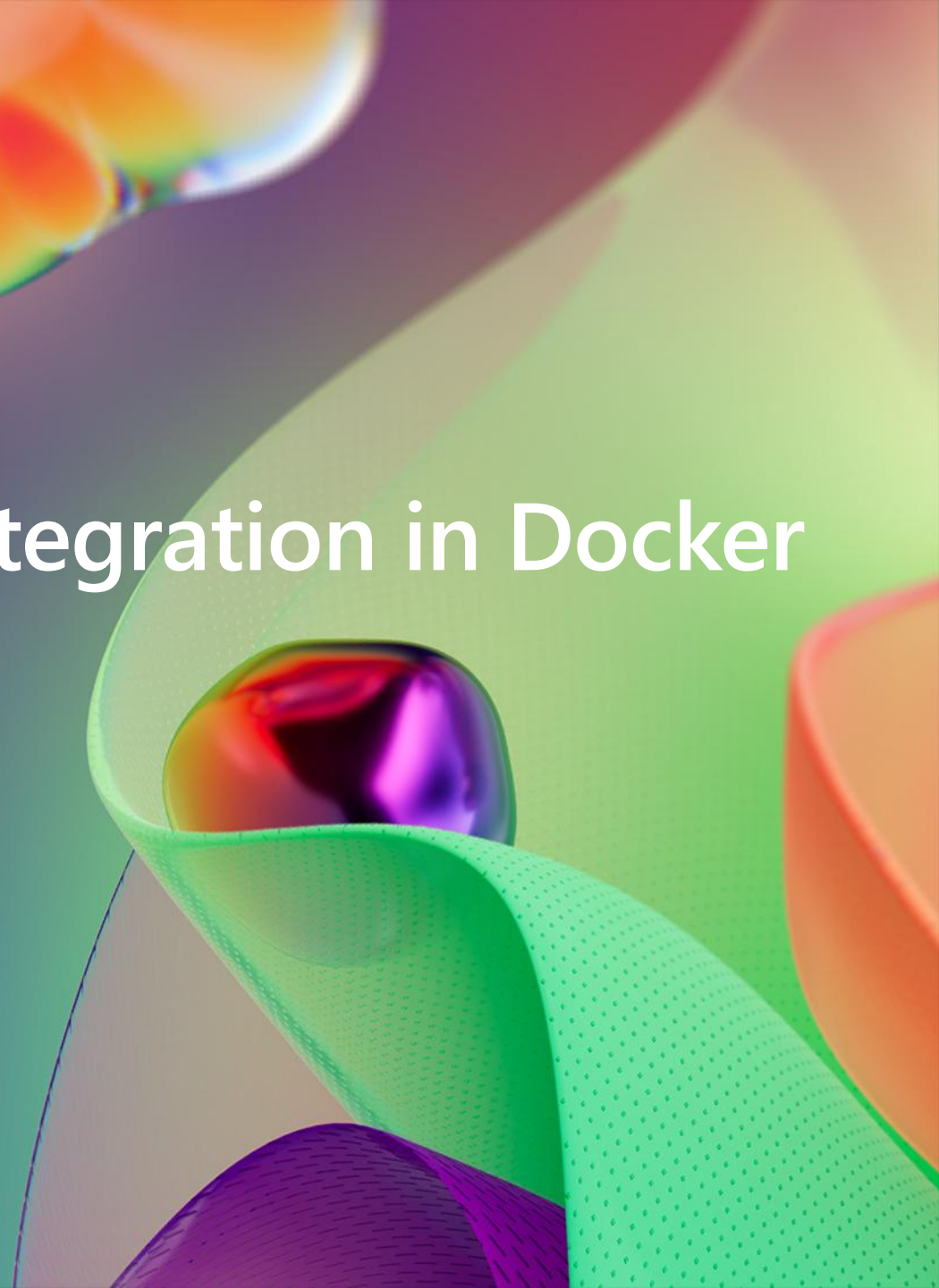
Microsoft

PostgreSQL and GraphRAG Integration in Docker

Presenter: Helen Zeng
AI GBB

May 27, 2025



Microsoft

Background and objectives:

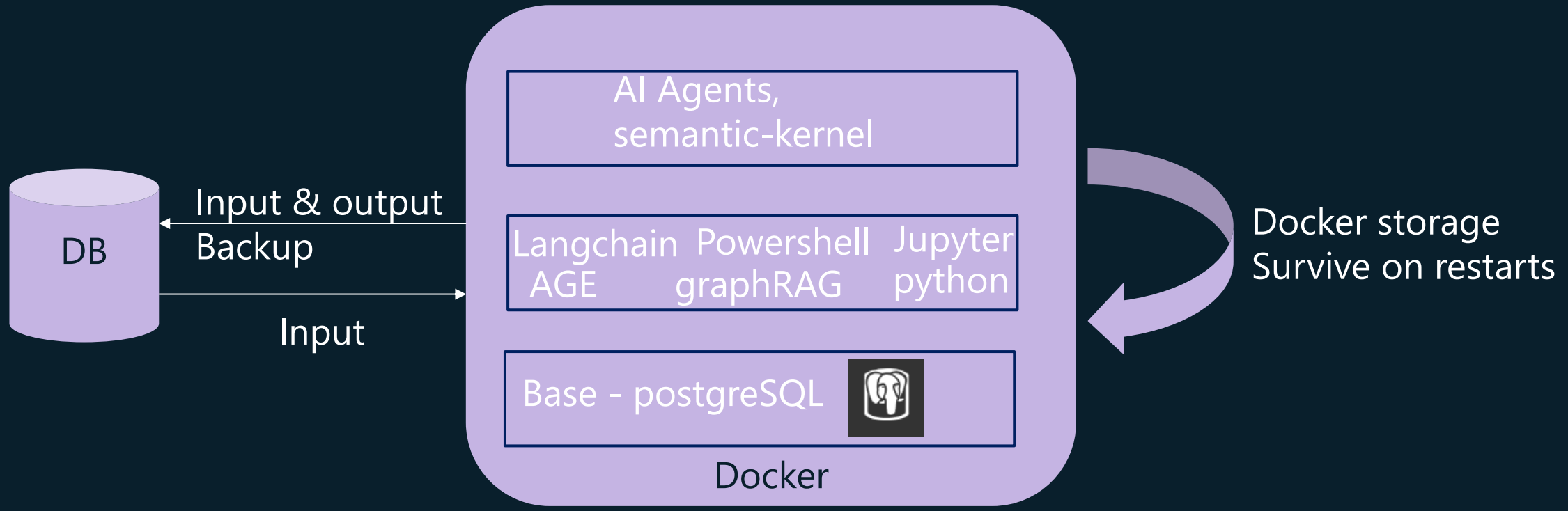
- Raw input data is already in a DB.
- Open Cypher query capability is needed.
- Run CLI version of graphRAG Conveniently.
- No need another data storage between the DB and graphRAG.

Solution design:

- Build graphARG index directly using data in the DB.
- Integrate PostgreSQL and graphRAG in a Docker container, add AGE (Apache Graph Extension), for running Cypher query.
- Bonus: add AI agent services on top.

Architecture

The docker image uses postgres as base, then added python, graphRAG, AGE, semantic-kernel and other needed packages. Two volumes are created to persist postgres data and app related data.




```
volumes:
  - local_postgres_data:/var/lib/postgresql/data
  - graphrag_data:/app/graphrag-folder
```

Preparation

1. Create Dockerfile
2. Prepare the .env file
3. Prepare setting.yaml
4. Prepare docker-compose.yaml
5. Prepare python code or jupyter notebook

.env example

Your DB properties

Your docker postgres property

If you use your docker postgres, then set to true

```
MY_DB_HOST=
MY_DB_PORT=
MY_DB_USER=
MY_DB_PASSWORD=
MY_DB_NAME=
MY_DB_SSLMODE=
PROMPTS=/app/graphrag-folder/prompts
CONFIG_PATH=/app/graphrag-folder/settings.yaml
AGE_HOST=postgres
AGE_PORT=5432
AGE_USER=postgres
AGE_PASSWORD=
AGE_DB=postgres
USE_LOCAL_AGE=true
GRAPHRAG_API_BASE=
GRAPHRAG_API_KEY=
GRAPHRAG_LLM_MODEL=gpt-4o
GRAPHRAG_LLM_API_VERSION="2025-01-01-preview"
GRAPHRAG_LLM_DEPLOYMENT=gpt-4o
GRAPHRAG_EMBEDDING_MODEL=text-embedding-3-small
GRAPHRAG_EMBEDDING_API_VERSION="2023-05-15"
GRAPHRAG_EMBEDDING_DEPLOYMENT=text-embedding-3-small
```

Step 0 – insert .txt input to DB if not already existing

Run python code to insert .txt files to DB:
> python insert-table.py

Step 1 – build docker image

This step builds docker images to include all modules needed to run services later.

```
$ docker build -t graphrag-img .
```

Step 2 – run postgres service

This step starts postgres service which other services depend on.

```
$ docker compose up postgres
```

```
[+] Running 2/2
 ✓ Network postgresql-age_default  Created                                    0.1s
 ✓ Container postgres              Created                                    0.1s
Attaching to postgres
postgres  |
postgres  | PostgreSQL Database directory appears to contain a database; Skipping initialization
postgres  |
postgres  | 2025-05-23 16:28:27.226 UTC [1] LOG:  starting PostgreSQL 16.9 (Debian 16.9-1.pgdg120+1) on x86_64-pc-linux-gnu, compil
ed by gcc (Debian 12.2.0-14) 12.2.0, 64-bit
postgres  | 2025-05-23 16:28:27.227 UTC [1] LOG:  listening on IPv4 address "0.0.0.0", port 5432
postgres  | 2025-05-23 16:28:27.227 UTC [1] LOG:  listening on IPv6 address "::", port 5432
postgres  | 2025-05-23 16:28:27.234 UTC [1] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
postgres  | 2025-05-23 16:28:27.250 UTC [29] LOG:  database system was shut down at 2025-05-23 16:05:41 UTC
postgres  | 2025-05-23 16:28:27.270 UTC [1] LOG:  database system is ready to accept connections
```

Step 3.1 – run ‘load-data’ service

This step loads data from DB to Docker folder /app/graphrag-folder/input as graphRAG input.

```
$ docker compose up load-data
```

```
[+] Running 2/2
 ✓ Container postgres       Running                                    0.0s
 ✓ Container load-data-app  Created                                    0.1s
Attaching to load-data-app
load-data-app  | All rows saved as .txt files in /app/graphrag-folder/input.
load-data-app exited with code 0
```

Step 3.2 – check data is present in postgres server

This sub-step checks docker folder /app/graphrag-folder/input to verify graphRAG input.

```
$ docker exec -it postgres bash
```

```
root@b754b0d605cb:/app/graphrag-folder# cd input
root@b754b0d605cb:/app/graphrag-folder/input# ls
input_10.txt  input_1.txt  input_2.txt  input_3.txt  input_4.txt  input_5.txt  input_6.txt  input_7.txt  input_8.txt  input_9.txt
root@b754b0d605cb:/app/graphrag-folder/input#
```

Step 4.1 – build graphrag index

This step runs ‘graphrag index’ using /app/graphrag-folder/input and generate output.

```
$ docker compose up graphrag-index
```

```
[+] Running 3/3
 ✓ Container postgres            Running                                    0.0s
 ✓ Container load-data-app       Created                                    0.0s
 ✓ Container graphrag-index-app  Created                                    0.1s
Attaching to graphrag-index-app
graphrag-index-app  | Total files: 10
```

```
graphrag-index-app  |
graphrag-index-app  | [119 rows x 2 columns]}
graphrag-index-app  | ⠋ GraphRAG Indexer
graphrag-index-app  | ├── Loading Input (InputFileType.text) - 10 files loaded (0 filtered) ━ 100%  0…
graphrag-index-app  | ├── create_base_text_units ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100% 0:00:00 0:00:00
graphrag-index-app  | ├── create_final_documents ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100% 0:00:00 0:00:00
graphrag-index-app  | ├── extract_graph ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100% 0:00:00 0:00:00
graphrag-index-app  | ├── finalize_graph ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100% 0:00:00 0:00:00
graphrag-index-app  | ├── create_communities ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100% 0:00:00 0:00:00
graphrag-index-app  | ├── create_final_text_units ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100% 0:00:00 0:00:00
graphrag-index-app  | ├── create_community_reports ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100% 0:00:00 0:00:00
graphrag-index-app  | ├── generate_text_embeddings ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100% 0:00:00 0:00:00🚀 All workflows completed su
ccessfully.
graphrag-index-app exited with code 0
```

Step 4.2 – check data is present in postgres server

This sub-step verifies ‘graphrag index’ output in /app/graphrag-folder/output.

```
$ docker exec -it postgres bash
```

```
root@b754b0d605cb:/app# cd graphrag-folder
root@b754b0d605cb:/app/graphrag-folder# ls
cache  input  logs  output  prompts  settings.yaml  update_output
root@b754b0d605cb:/app/graphrag-folder# cd input
root@b754b0d605cb:/app/graphrag-folder/input# ls
input_10.txt  input_1.txt  input_2.txt  input_3.txt  input_4.txt  input_5.txt  input_6.txt  input_7.txt  input_8.txt  input_9.txt
root@b754b0d605cb:/app/graphrag-folder/input# cd ../output
root@b754b0d605cb:/app/graphrag-folder/output# ls
communities.parquet        embeddings.community.full_content.parquet  graph.graphml          text_units.parquet
community_reports.parquet  embeddings.entity.description.parquet      lancedb
context.json               embeddings.text_unit.text.parquet          relationships.parquet
documents.parquet          entities.parquet                           stats.json
root@b754b0d605cb:/app/graphrag-folder/output# cd lancedb
root@b754b0d605cb:/app/graphrag-folder/output/lancedb# ls
default-community-full_content.lance  default-entity-description.lance  default-text_unit-text.lance
root@b754b0d605cb:/app/graphrag-folder/output/lancedb#
```

Step 5 – write index output to DB

This step stores the graph output data in DB, as docker backup.
The content in /app/graphrag-folder/output will be saved into DB. This example uses Azure hosted PostgreSQL as DB.

```
$ docker compose up graphrag-writer
```

```
[+] Running 4/4
 ✓ Container postgres                Running                                0.0s
 ✓ Container load-data-app           Created                                0.0s
 ✓ Container graphrag-index-app      Created                                0.0s
 ✓ Container graphrag-writer-app     Created                                0.1s
Attaching to graphrag-writer-app
graphrag-writer-app  | Skipping JSON file: /app/graphrag-folder/logs/logs.json
graphrag-writer-app  | All relevant files have been processed and stored in PostgreSQL.
graphrag-writer-app exited with code 0
```

Step 6 – build graph, prepare for Cypher query

This step builds graph using AGE on PostgreSQL in docker, to prepare for Cypher query.

```
$ docker compose up build-graph
```

```
[+] Running 5/5
 ✓ Container postgres              Running                                    0.0s
 ✓ Container load-data-app         Created                                    0.0s
 ✓ Container graphrag-index-app    Created                                    0.0s
 ✓ Container graphrag-writer-app   Created                                    0.0s
 ✓ Container build-graph-app       Created                                    0.1s
Attaching to build-graph-app
build-graph-app  | Existing graph 'graphRAG' dropped.
build-graph-app  | Graph 'graphRAG' will be created.
build-graph-app  | Inserted 10 rows into Document
build-graph-app  | Index(['id', 'human_readable_id', 'title', 'text', 'text_unit_ids',
build-graph-app  |        'creation_date', 'metadata'],
build-graph-app  |       dtype='object')
build-graph-app  | Inserted 449 rows into Entity
build-graph-app  | Index(['id', 'human_readable_id', 'title', 'type', 'description',
build-graph-app  |        'text_unit_ids', 'frequency', 'degree', 'x', 'y'],
build-graph-app  |       dtype='object')
build-graph-app  | Inserted 690 rows into relationship
build-graph-app  | Index(['id', 'human_readable_id', 'source', 'target', 'description', 'weight',
build-graph-app  |        'combined_degree', 'text_unit_ids'],
build-graph-app  |       dtype='object')
build-graph-app  | Data loaded into AGE graph successfully.
build-graph-app exited with code 0
```

View the graph content

This step verifies graph in PostgreSQL ag_catalog in docker, name graphRAG.

```
root@de59c1cc762b:/app# psql -U postgres -d postgres <<EOF > output.txt
> LOAD 'age';
> SET search_path = ag_catalog, "\$user", public;
> SELECT * FROM cypher('graphRAG', \$\$ MATCH (n) RETURN n LIMIT 5 \$\$) AS (n agtype);
> EOF
```

```
root@de59c1cc762b:/app# less output.txt
```

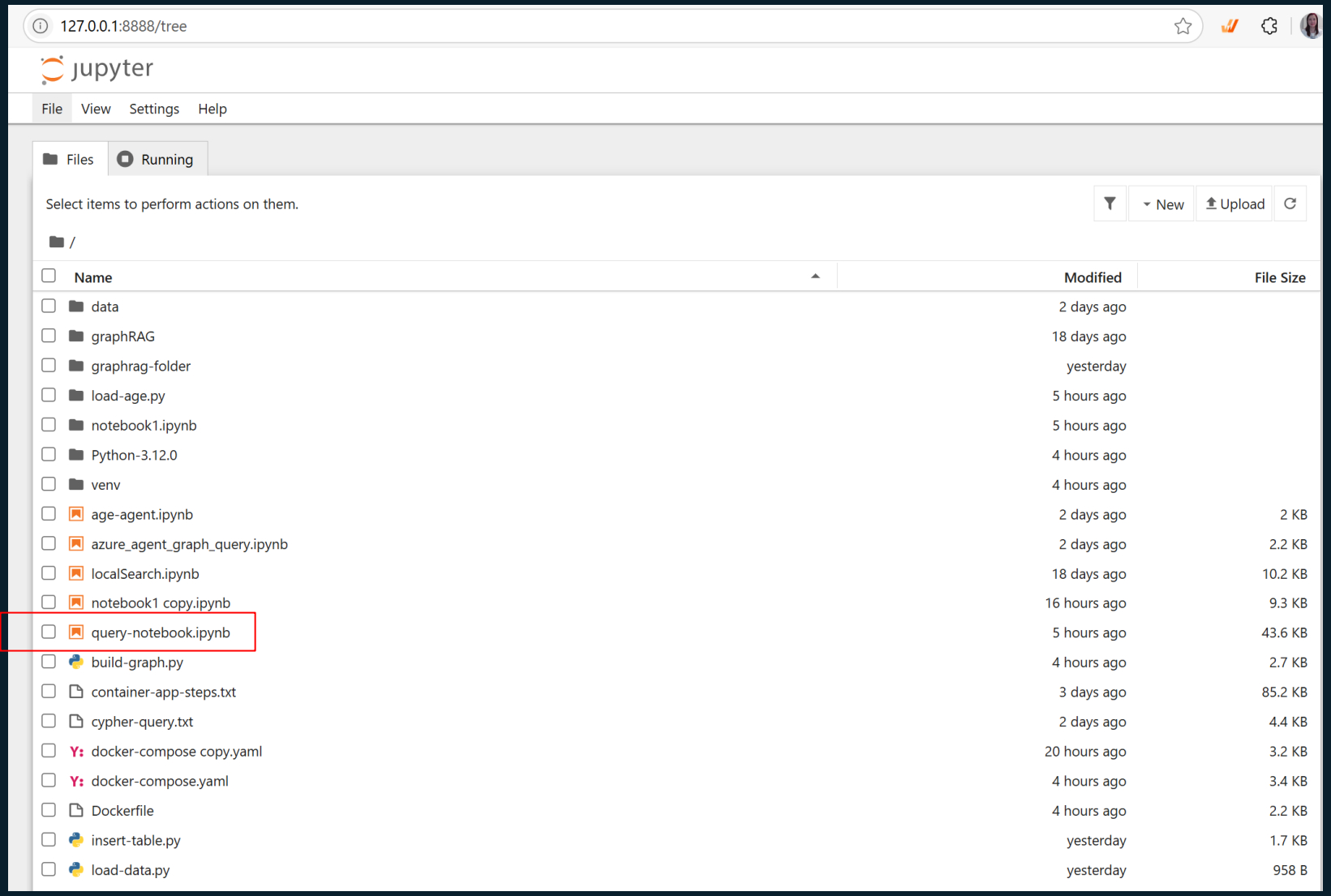
Step 7 – run query in jupyter notebook

This step runs jupyter notebook in docker.

```
$ docker compose up query-notebook
```

```
[+] Running 6/6
 ✓ Container postgres              Running                                    0.0s
 ✓ Container load-data-app         Created                                    0.0s
 ✓ Container graphrag-index-app    Created                                    0.0s
 ✓ Container graphrag-writer-app   Created                                    0.0s
 ✓ Container build-graph-app       Created                                    0.0s
 ✓ Container query-notebook-app    Recreated                                  0.1s
Attaching to query-notebook-app
query-notebook-app  | [I 2025-05-23 16:51:30.721 ServerApp] jupyter_lsp | extension was successfully linked.
query-notebook-app  | [I 2025-05-23 16:51:30.727 ServerApp] jupyter_server_terminals | extension was successfully linked.
query-notebook-app  | [W 2025-05-23 16:51:30.728 LabApp] 'shutdown_no_activity_timeout' has moved from NotebookApp to ServerApp. Th
is config will be passed to ServerApp. Be sure to update your config before our next release.
query-notebook-app  | [I 2025-05-23 16:51:30.730 ServerApp] jupyterlab | extension was successfully linked.
query-notebook-app  | [I 2025-05-23 16:51:30.734 ServerApp] notebook | extension was successfully linked.
query-notebook-app  | [I 2025-05-23 16:51:30.742 ServerApp] Writing Jupyter server cookie secret to /root/.local/share/jupyter/runt
ime/jupyter_cookie_secret
query-notebook-app  | [I 2025-05-23 16:51:31.018 ServerApp] notebook_shim | extension was successfully linked.
query-notebook-app  | [I 2025-05-23 16:51:31.060 ServerApp] notebook_shim | extension was successfully loaded.
query-notebook-app  | [I 2025-05-23 16:51:31.062 ServerApp] jupyter_lsp | extension was successfully loaded.
query-notebook-app  | [I 2025-05-23 16:51:31.063 ServerApp] jupyter_server_terminals | extension was successfully loaded.
query-notebook-app  | [I 2025-05-23 16:51:31.078 LabApp] JupyterLab extension loaded from /app/venv/lib/python3.12/site-packages/ju
pyterlab
query-notebook-app  | [I 2025-05-23 16:51:31.078 LabApp] JupyterLab application directory is /app/venv/share/jupyter/lab
query-notebook-app  | [I 2025-05-23 16:51:31.080 LabApp] Extension Manager is 'pypi'.
query-notebook-app  | [I 2025-05-23 16:51:31.136 ServerApp] jupyterlab | extension was successfully loaded.
query-notebook-app  | [I 2025-05-23 16:51:31.138 ServerApp] notebook | extension was successfully loaded.
query-notebook-app  | [I 2025-05-23 16:51:31.139 ServerApp] Serving notebooks from local directory: /app
query-notebook-app  | [I 2025-05-23 16:51:31.139 ServerApp] Jupyter Server 2.16.0 is running at:
query-notebook-app  | [I 2025-05-23 16:51:31.139 ServerApp] http://
query-notebook-app  | [I 2025-05-23 16:51:31.139 ServerApp]     http://127.0.0.1:8888/tree?token=
query-notebook-app  | [I 2025-05-23 16:51:31.139 ServerApp] Use Control-C to stop this server and shut down all kernels (twice to s
kip confirmation).
query-notebook-app  | [C 2025-05-23 16:51:31.150 ServerApp]
```

Jupyter notebook UI



Cypher query examples

Several ways of querying are provided:

- Cypher query
  - Example given to check nodes and relationships, and their properties.
- Multi-hop cypher query
  - One of the advantages of using knowledge graph is the multi-hop query.
- GraphRAG LocalSearch
  - Examples given to show the impact of community level to the query results.
- GraphRAG GlobalSearch
  - Both python function call and CLI methods are provided.
- Vector search
  - Example shows that vector search result is restricted to the original text; it typically returns the top-k most similar original chunks. In other words, vector search is retrieval only, it returns what was said, not what it means. However, graphRAG combines vector search with a knowledge graph that encodes relationships between entities. The graph is built from LLM-extracted triples, which helps contextual thinking; thus, graphRAG provides more synthesized and thoughtful response.

Build and run AI agents example

Task: summarize the podcasts.

```
await summarize_all_text("/app/graphrag-folder/input", agent)
```

```
--- Combined Summary ---
 Sure, here's a concise summary of the lengthy text you provided:

---

**Summary:**

The podcast *Behind the Tech*, hosted by Microsoft CTO Kevin Scott, explores diverse stories about innovators in technology. In this ser
ies, Scott interviews influential figures such as Jaron Lanier, Reid Hoffman, Danielle Feinberg, Alice Steinglass, Andrew Ng, Anders Hej
lsberg, Surya Ganguli, and others, showcasing their unique journeys into tech and their impact on the industry.

Common themes across episodes include:
1. **Career Journeys:** Interviewees shared personal anecdotes about how curiosity, chance, and determination shaped their paths—from ea
rly exposure to computing to tackling complex challenges in their careers, including machine learning, VR, programming languages, and an
imation.
2. **Intersection of Creativity and STEM:** Guests like Danielle Feinberg and Anders Hejlsberg describe blending artistic and technical
craftsmanship in their work, whether creating visually stunning Pixar films or designing programming tools like Turbo Pascal and C#.
3. **Advocacy for Diversity & Education:** Many guests emphasize the importance of inspiring younger generations, especially women and m
inorities, to pursue computer science. Initiatives like *Code.org* and Stanford's Human-Centered AI Institute are addressing systemic ga
ps to improve diversity and accessibility in tech.
4. **Ethical & Societal Implications:** Episodes discuss AI's rapid evolution, needing thoughtful regulation and ethical responsibility.
Insights from academics like Surya Ganguli highlight concerns ranging from biases in data to the energy inefficiencies of digital comput
ation.
5. **Technological Impact:** Guests explore transformative industries such as VR, healthcare, fintech, and agriculture, demonstrating wa
ys technology solves real-world problems and shapes global economies.

Interview highlights include:
- Jaron Lanier's critique of the Internet's influence on society.
- Reid Hoffman's insights on entrepreneurial success and China's ability to "blitzscale."
- Danielle Feinberg's role in leveraging math and storytelling in Pixar animation.
- Alice Steinglass advocating CS education for diverse classrooms.

Through their candid experiences, these technologists inspire listeners to appreciate the challenges, opportunities, and ever-expanding
possibilities within STEM.

---

Let me know if you'd like to dive deeper into any individual story from the text.
```

Optional step: reconstruct-graph

In case the graph related data in docker /app/graphrag-folder/output lost, then the graph can be reconstructed using the data stored in DB, which was saved in step 5.

```
$ docker compose up reconstruct-graph
```

```
[+] Running 2/2
 ✓ Container postgres                 Running                                    0.0s
 ✓ Container reconstruct-graph-app    Created                                    0.3s
Attaching to reconstruct-graph-app
reconstruct-graph-app  | All files have been reconstructed from Azure-hosted PostgreSQL to /app/graphrag-folder/re
store.
reconstruct-graph-app exited with code 0
```

```
$ docker exec -it postgres bash
```

```
root@076170b43c6f:/app# cd graphrag-folder
root@076170b43c6f:/app/graphrag-folder# ls
cache  graph.graphml  input  logs  output  plugins  prompts  restore  settings.yaml  update_output
root@076170b43c6f:/app/graphrag-folder# cd restore
root@076170b43c6f:/app/graphrag-folder/restore# ls
communities.parquet        embeddings.community.full_content.parquet  graph.graphml          text_units.parquet
community_reports.parquet  embeddings.entity.description.parquet      relationships.parquet
context.json               embeddings.text_unit.text.parquet          stats.json
documents.parquet          entities.parquet                           summaries.json
root@076170b43c6f:/app/graphrag-folder/restore#
```

Query in docker postges

```
$ docker exec -it postgres bash
```

```
root@307e79cfb708:/app# psql -U postgres -d postgres
psql (16.9 (Debian 16.9-1.pgdg120+1))
Type "help" for help.

postgres=# LOAD 'age';
LOAD
postgres=# SET search_path = ag_catalog, "\$user", public;
SET
postgres=#     SELECT * FROM cypher('graphRAG', $$
       MATCH (n) RETURN labels(n), count(n)
    $$) AS (labels agtype, count agtype);
    labels    | count
--------------+-------
 ["Entity"]   | 449
 ["Document"] | 10
(2 rows)

postgres=#     SELECT * FROM cypher('graphRAG', $$
       MATCH ()-[r]->() RETURN type(r), count(r)
    $$) AS (relationship agtype, count agtype);
 relationship | count
--------------+-------
 "RELATED_TO" | 690
(1 row)

postgres=# \dt
           List of relations
   Schema   |   Name   | Type  |  Owner
------------+----------+-------+----------
 ag_catalog | ag_graph | table | postgres
 ag_catalog | ag_label | table | postgres
(2 rows)

postgres=#
```

Graph storage

Graph metadata, like nodes, edges stored in Docker postgres

```
$ docker exec -it postgres bash
```

```
root@299fc1439d4d:/app# psql -U postgres -d postgres
psql (16.9 (Debian 16.9-1.pgdg120+1))
Type "help" for help.

postgres=# \dt
Did not find any relations.
postgres=# LOAD 'age';
LOAD
postgres=# SET search_path = ag_catalog, "$user", public;
SET
postgres=# SELECT * FROM cypher('graphRAG', $$
    MATCH ()--() RETURN count(*)
$$) AS (count agtype);
 count
-------
 1380
(1 row)

postgres=# SELECT * FROM cypher('graphRAG', $$
    MATCH ()-[r]->() RETURN count(r)
$$) AS (count agtype);
 count
-------
 690
(1 row)
```

Graph input and output stored in DB (in this example, it's Azure hosted PostgreSQL)

```
postgres=> \dt
                List of relations
 Schema |       Name       | Type  |   Owner
--------+------------------+-------+-----------
 public | graphrag_inputs  | table | XXXX
 public | graphrag_outputs | table | XXXX
(2 rows)
```