

Complete Guide - Graph RAG Agent Workflow with n8n

Ahmed Aziz Ammar

Table des matières

1	Introduction	3
2	Prerequisites	3
3	Docker Desktop Installation	3
4	Installing n8n in Self-Hosted Mode with Docker	3
5	Accessing the n8n Interface	4
6	Configuring Neo4j in the Cloud	4
7	Installing Neo4j in n8n	6
8	Overview of the Graph RAG Workflow	7
8.1	General Architecture	8
9	Importing the First Workflow into n8n	9
10	Workflow No. 1 : Graph Database Workflow	9
10.1	Node 1.1 : Execute Workflow	10
10.2	Node 1.2 : Read/Write Files from Disk	10
10.3	Node 1.3 : Extract from File	11
10.4	Node 1.4 : Edit Fields1	11
10.5	Node 1.5 : Execute Command	12
10.6	Code Node 1.6 : Extract All Information (!!Important)	13
10.7	Code Node 1.7 : Bad Format Cleanup	13
10.8	Node 1.8 : Edit Fields	14
10.9	Important : Creating the Neo4j Credential	15
10.10	Filling in the Connection Fields	16
10.11	Node 1.9 : Create Database (Neo4j)	17
10.12	Relationship Nodes : Unique Pairs → AI Agent → Bidirectional Pairs → ExecuteQuery graphDb	18
11	Click the Execute Workflow Button	24
12	Import the Second Workflow into n8n	24

13 Workflow No. 2 : Graph RAG Agent	25
13.1 Architecture	25
13.2 Node 2.1 : When Chat Message Received	25
13.3 Node 2.2 : AI Agent	25
13.4 Node 2.3 : OpenAI Chat Model	27
13.5 Node 2.4 : Postgres Chat Memory	27
13.6 Node 2.5 : ExecuteQuery graphDb in Neo4j	28
14 Test the Graph RAG	29
15 Conclusion	29

1 Introduction

This guide details the steps to install and run a Graph RAG workflow locally with a cloud-based graph database, using **n8n**. n8n is a no-code/low-code automation tool for orchestrating tasks such as document analysis, API calls, file management, etc.

2 Prerequisites

Before getting started, make sure you have the following elements on your machine :

- GIT installed
- 2.7 GB space for Docker Desktop
- 4.5 GB space for self-hosted-ai-starter-kit container
- Terminal or Command Prompt

3 Docker Desktop Installation

1. Download Docker Desktop from the official link : <https://www.docker.com/products/docker-desktop>
2. Install Docker Desktop following the instructions for Windows.
3. Verify that Docker is operational with the command :

```
docker --version
```

4 Installing n8n in Self-Hosted Mode with Docker

1. Clone the official **self-hosted-ai-starter-kit** repository : **git clone https://github.com/n8n-io/self-hosted-ai-starter-kit.git** Check if there's a space between "https" and " :" — if so, concatenate them.
2. Access the cloned directory : **cd self-hosted-ai-starter-kit**
3. Copy the example **.env** file and rename it : **copy .env.example .env**
4. Launch the installation with Docker Compose :

```
docker-compose --profile cpu up
```

Check if there's a space between "-" and "profile" — if so, concatenate them.

5. Check the location of the **self-hosted-ai-starter-kit** folder on your machine. We need the **.env** file for PostgreSQL configuration and the **shared** folder for resource location.

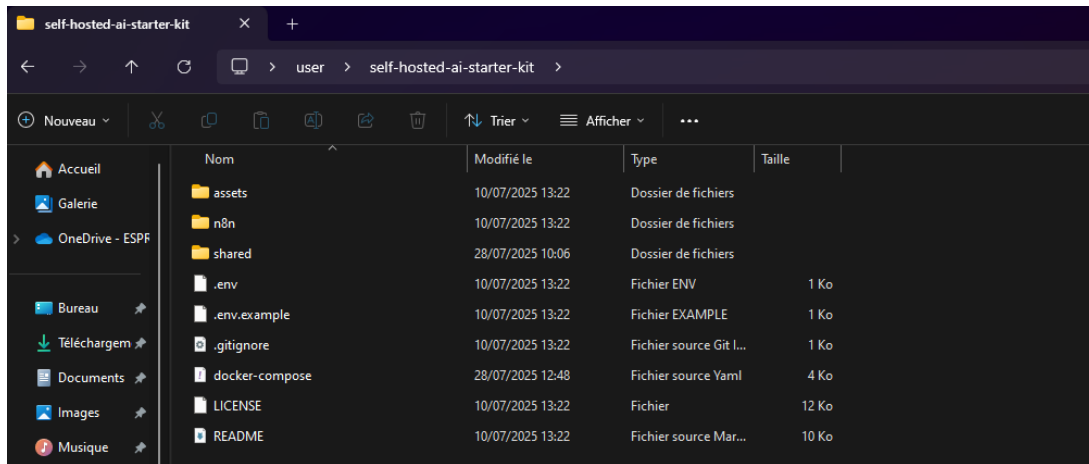


FIGURE 1 – Structure of the self-hosted-ai-starter-kit folder

5 Accessing the n8n Interface

1. After installation, open Docker Desktop and verify that the container `self-hosted-ai-starter-kit` is running.
2. Click the generated link : `5678:5678` to access the web interface, or open : <http://localhost:5678>

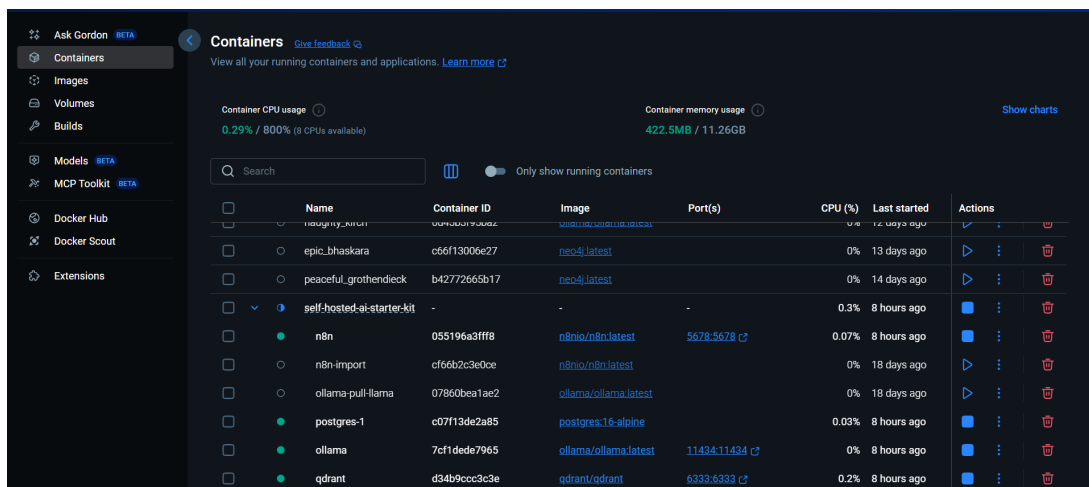


FIGURE 2 – self-hosted-ai-starter-kit container in Docker Desktop

3. Complete the registration form in n8n (First name, Last name, Email, etc.).

References :

- Official n8n guide : <https://github.com/n8n-io/self-hosted-ai-starter-kit>
- Download Docker Desktop : <https://www.docker.com/products/docker-desktop>

6 Configuring Neo4j in the Cloud

To set up Neo4j in cloud mode, we use the **Neo4j AuraDB** platform, which hosts graph databases and provides secure API access.

- Visit the official website : <https://console.neo4j.io/>.
- Sign in using your Gmail or Outlook account.
- Once logged in, you will be redirected to an interface prompting you to download a file containing your credentials :
 - **NEO4J_USERNAME**
 - **NEO4J_PASSWORD**
 - **NEO4J_URI**
- Download and store this file securely. Wait a few minutes for the free instance to be created (named “free instance” in Neo4j Aura).

```
# Wait 60 seconds before connecting using these details, or login to https://console.neo4j.io to validate the Aura Instance is available
NEO4J_URI=neo4j+s://bf4fb4b6.databases.neo4j.io
NEO4J_USERNAME=neo4j
NEO4J_PASSWORD=wEaht8y9mhN1u09VtNkLmxMawXwOMJdm-pFddyYS8w
NEO4J_DATABASE=neo4j
NEO4J_INSTANCEID=bf4fb4b6
NEO4J_INSTANCENAME=trial instance
```

FIGURE 3 – Neo4j AuraDB credentials file

- You will then be redirected to the Neo4j Aura management interface.

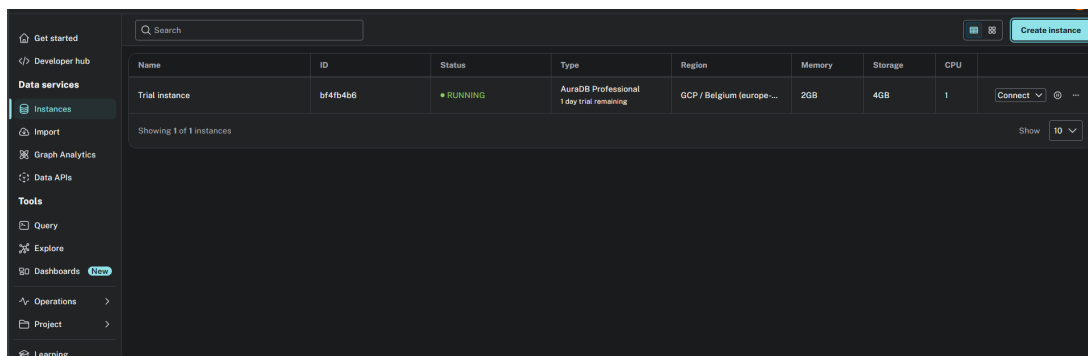


FIGURE 4 – Neo4j Aura interface

- Click on “Connect” and select “Query”. You will be redirected to the Cypher Query interface.

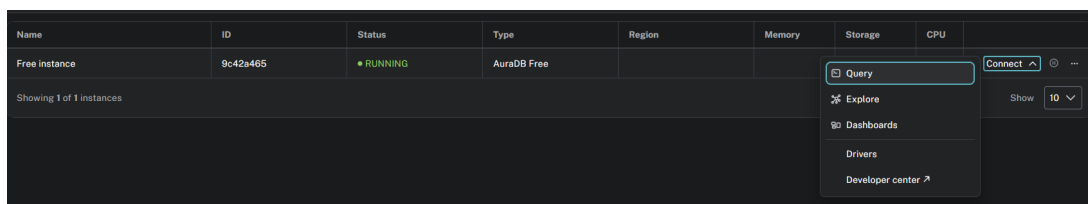


FIGURE 5 – Neo4j connection options

- You can execute Cypher commands to visualize nodes and their relationships.
- **Example query :**
`MATCH (n:YourGraphDatabaseName) RETURN n LIMIT 50;`

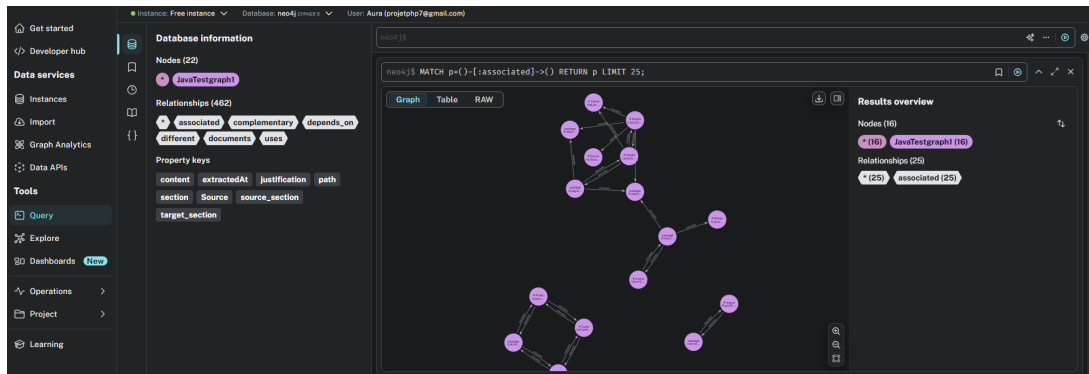


FIGURE 6 – Visualizing relationships between nodes in Neo4j

7 Installing Neo4j in n8n

To integrate Neo4j with n8n, follow these steps :

1. Go to **Settings** in n8n.
2. Select the **Community Nodes** tab.
3. Click the **Install** button.
4. In the search bar, type : **n8n-nodes-neo4j**.
5. Select the package found, then click **Install**.

Illustrative Screenshots

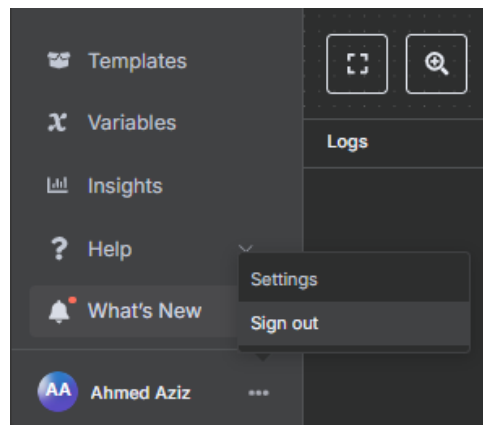


FIGURE 7 – Settings menu in n8n

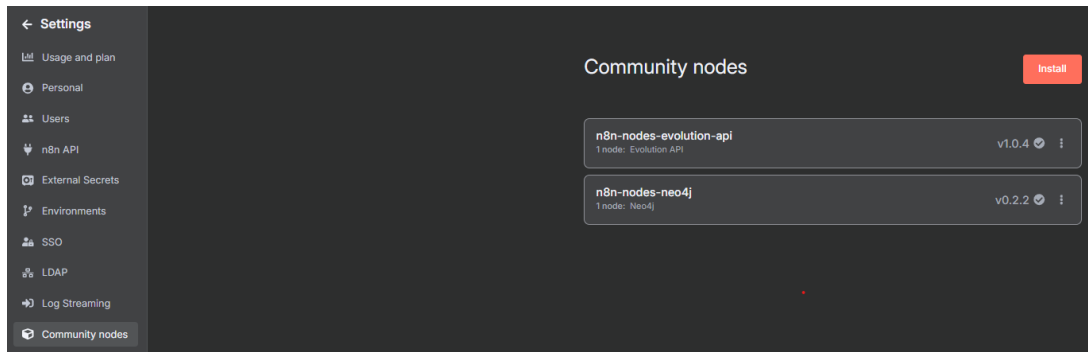


FIGURE 8 – Community Nodes tab for extension installation

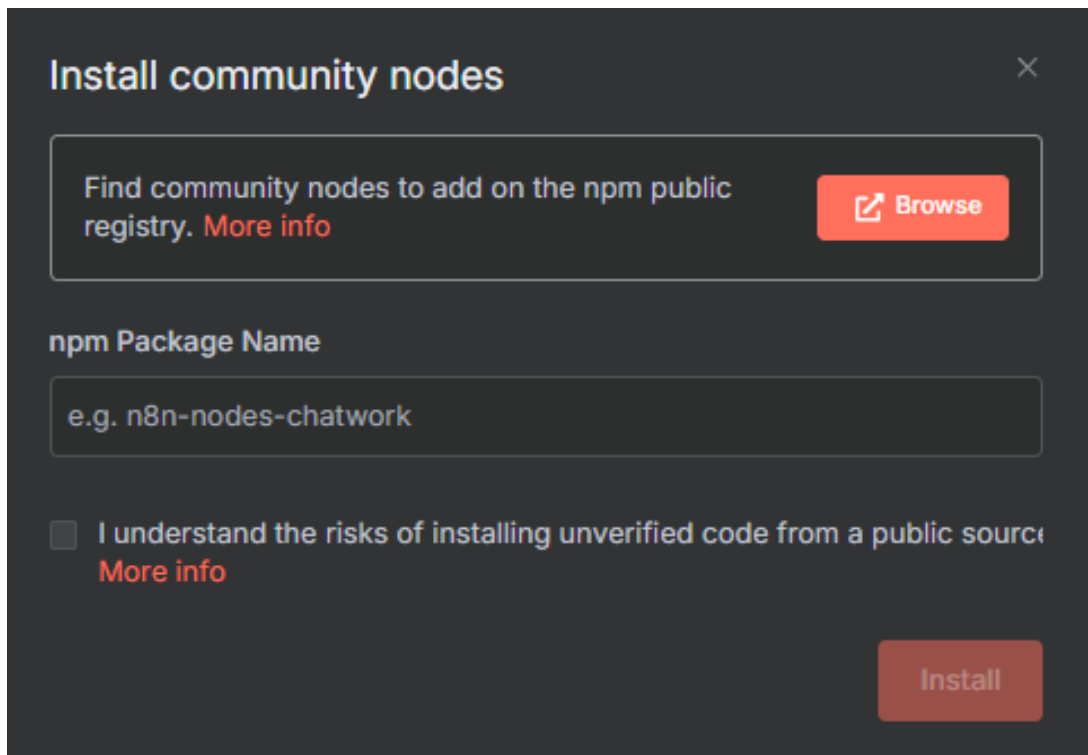


FIGURE 9 – Installing the Neo4j node (n8n-nodes-neo4j)

8 Overview of the Graph RAG Workflow

These two workflows serve the following main purposes :

- Automate the ingestion of documents (Java, Markdown) from local directories.
- Index and store these documents as nodes in a cloud-hosted Neo4j graph database.
- Automatically generate semantic relationships between documents (e.g., *documents*, *complementary*, *different*).
- Enable advanced queries and synthetic responses based on the graph (**Graph RAG** approach).
- Provide justified answers with metadata from source documents (title, section, extraction date, content snippet).

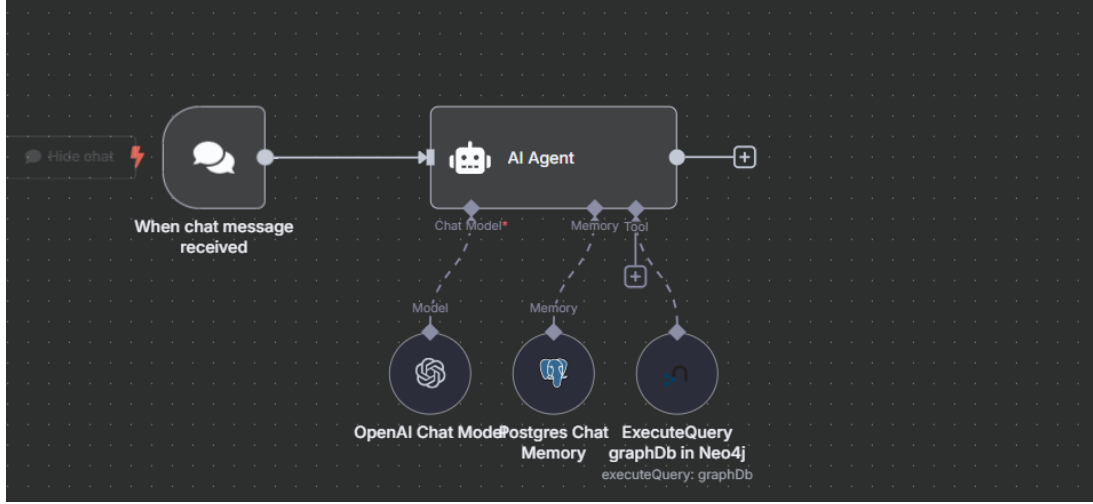


FIGURE 10 – Graph RAG Agent Workflow

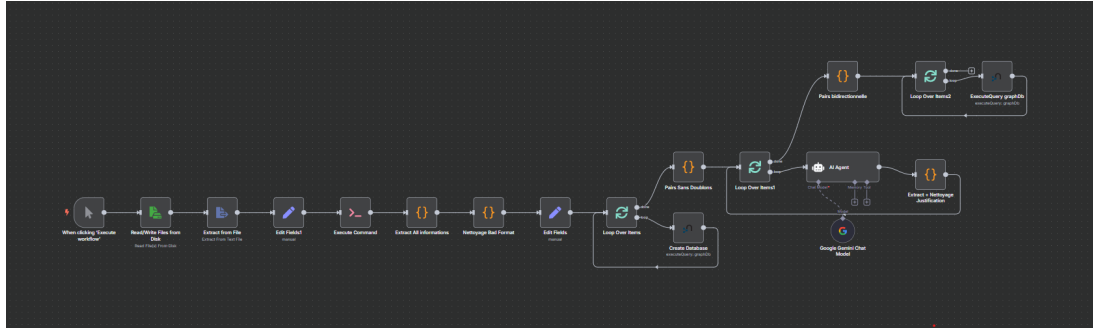


FIGURE 11 – Ingestion and setup workflow for Neo4j graph database

8.1 General Architecture

The architecture is based on two main interconnected workflows built around the Neo4j graph database :

1. Graph Construction Workflow This first workflow aims to ingest and structure knowledge from source files (Java, Markdown). It includes :

- **Reading and extracting documents** : retrieving file content and metadata.
- **Cleaning and normalization** : preparing data to avoid insertion errors.
- **Creating nodes and relationships in Neo4j** : inserting files as enriched nodes and automatically generating semantic relationships between them.

2. Graph RAG Agent Workflow This second workflow enables conversational interaction with the graph :

- **Receiving user queries** through a chat interface.
- **Retrieving relevant information** by querying Neo4j with a RAG agent.
- **Multi-document reasoning** to provide a synthesized response based on files and their relationships, without hallucination.
- **Conversational memory** to ensure continuity in dialogue.

Knowledge Base (Neo4j) Neo4j is the core of the architecture, organizing documents and their links as a graph, enabling advanced searches and contextual reasoning.

9 Importing the First Workflow into n8n

1. Launch n8n
2. Go to <http://localhost:5678>
3. Navigate to the Workflows menu > Create Workflow

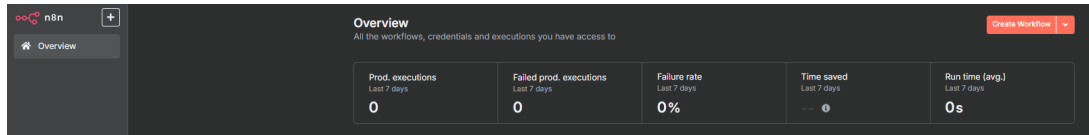


FIGURE 12 – n8n home interface

4. Click on > Import from File

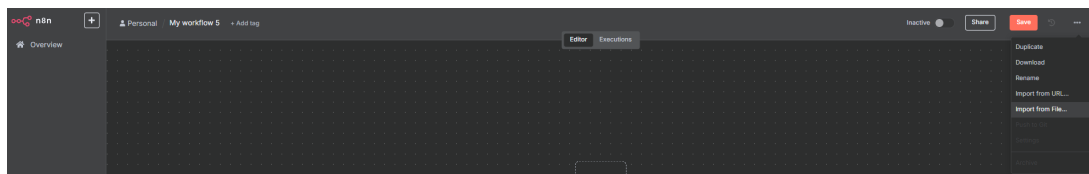


FIGURE 13 – Import from a file

5. Select the file Graph Database Resources:JAVA+MD.json
6. Click Save

10 Workflow No. 1 : Graph Database Workflow

Node Architecture

Manual Trigger → Read Files → Extract Content → Clean and Normalize →
Generate Pairs → AI-Based Relationship Analysis → Insert into Neo4j
(Nodes + Relationships)

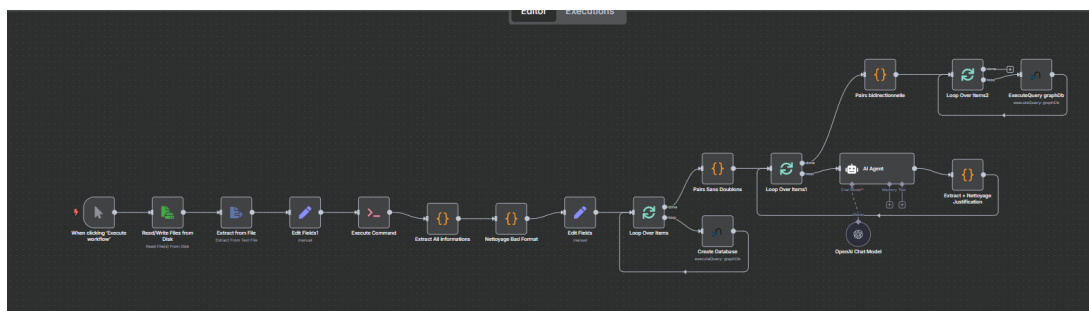


FIGURE 14 – Graph creation workflow in Neo4j

Node Configuration

10.1 Node 1.1 : Execute Workflow

Role : Manually triggers the file indexing and relationship generation process.

Configuration :

- **Type :** Manual Trigger
- **Usage :** Click to start the workflow that reads the files, cleans the content, and inserts the data into Neo4j.

10.2 Node 1.2 : Read/Write Files from Disk

Role : Recursively reads files from the local filesystem.

Configuration :

- **Operation :** Read Files
- **File(s) Selector :** To read all ‘.java’ and ‘.md’ files, including from subfolders, make sure n8n is installed via Docker. Add your resources to the **shared** directory inside **self-hosted-ai-starter-kit** :
`/data/shared/YourPATH/**/*.{java,md}.JAVA,md,markdown`

Explanation of the glob pattern :

- **** :** Recursively traverses subdirectories
- *** :** Any file
- **{JAVA,md,markdown} :** Targeted file extensions

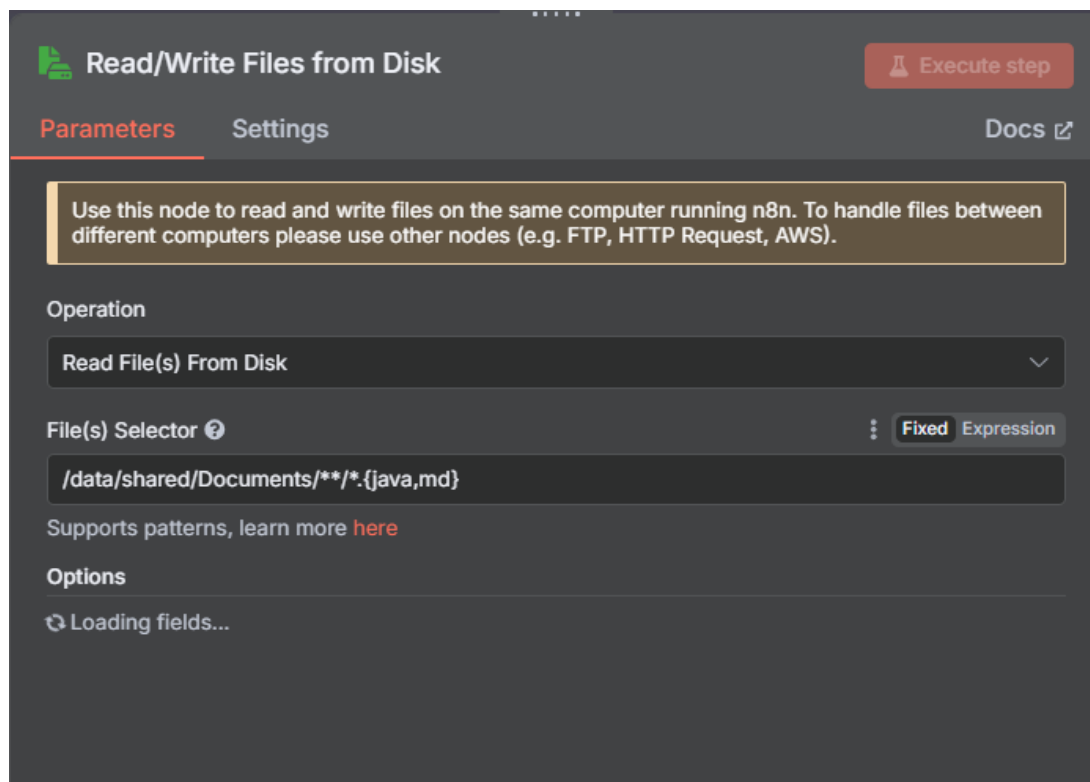
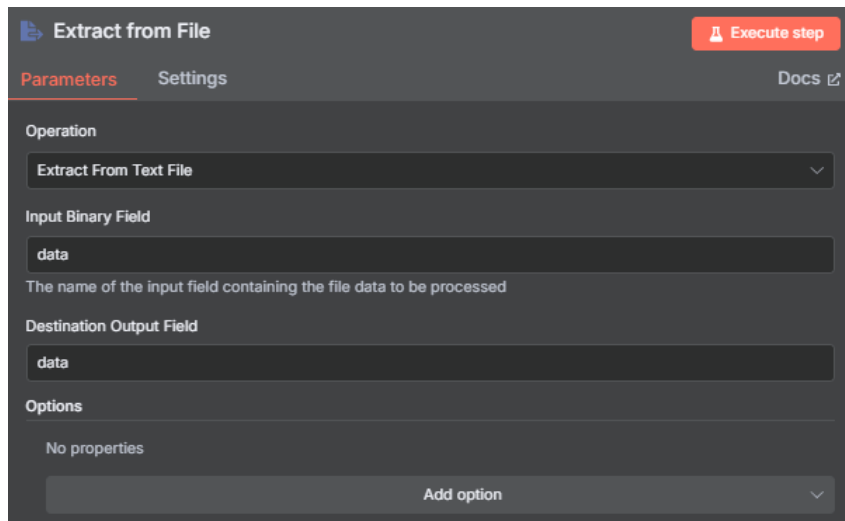


FIGURE 15 – File reading configuration

10.3 Node 1.3 : Extract from File

Role : Extracts raw content from files (TXT, MD, Markdown, Java).



The screenshot shows the configuration panel for the 'Extract from File' node. At the top, there is a title bar with a file icon, the text 'Extract from File', and an 'Execute step' button. Below the title bar are two tabs: 'Parameters' (active) and 'Settings'. A 'Docs' link is also present. The main configuration area is divided into sections: 'Operation' with a dropdown menu set to 'Extract From Text File'; 'Input Binary Field' with a text input containing 'data' and a description 'The name of the input field containing the file data to be processed'; 'Destination Output Field' with a text input containing 'data'; and 'Options' with the text 'No properties' and an 'Add option' button.

FIGURE 16 – File extraction configuration

10.4 Node 1.4 : Edit Fields1

Role : Normalizes the extracted fields to create properties such as `file_name` and `content`.

Edit Fields1 Execute step

Parameters Settings Docs

Mode
Manual Mapping

Fields to Set

file_name	A String	= {{ \$('Read/Write Files from Disk').item.json.fileName }}
content	A String	= {{ \$json.data }}

Drag input fields here or Add Field

Include Other Input Fields ☐

Options
No properties
Add option

FIGURE 17 – Field editing configuration

10.5 Node 1.5 : Execute Command

Role : Lists all file paths using the shell command :

```
find /data/shared/YourPATH/*
```

Execute Command Execute step

Parameters Settings Docs

Execute Once ☒

Command
find /data/shared/resource/*

FIGURE 18 – Shell command execution configuration

10.6 Code Node 1.6 : Extract All Information (!!Important)

Role : Associates each detected file path with the extracted metadata and content in preparation for insertion into Neo4j.

!!WARNING : It is crucial to ****modify line 32 of the code**** to specify the ****root path of your resources folder****. Without this change, no data will be correctly linked in the graph database.

Example :

```
const relativePath = fullPath.replace('/data/shared/resource/', '');
```

Tip : This path must point to your **/data/shared/** directory because it is the mounted volume used by n8n via Docker.

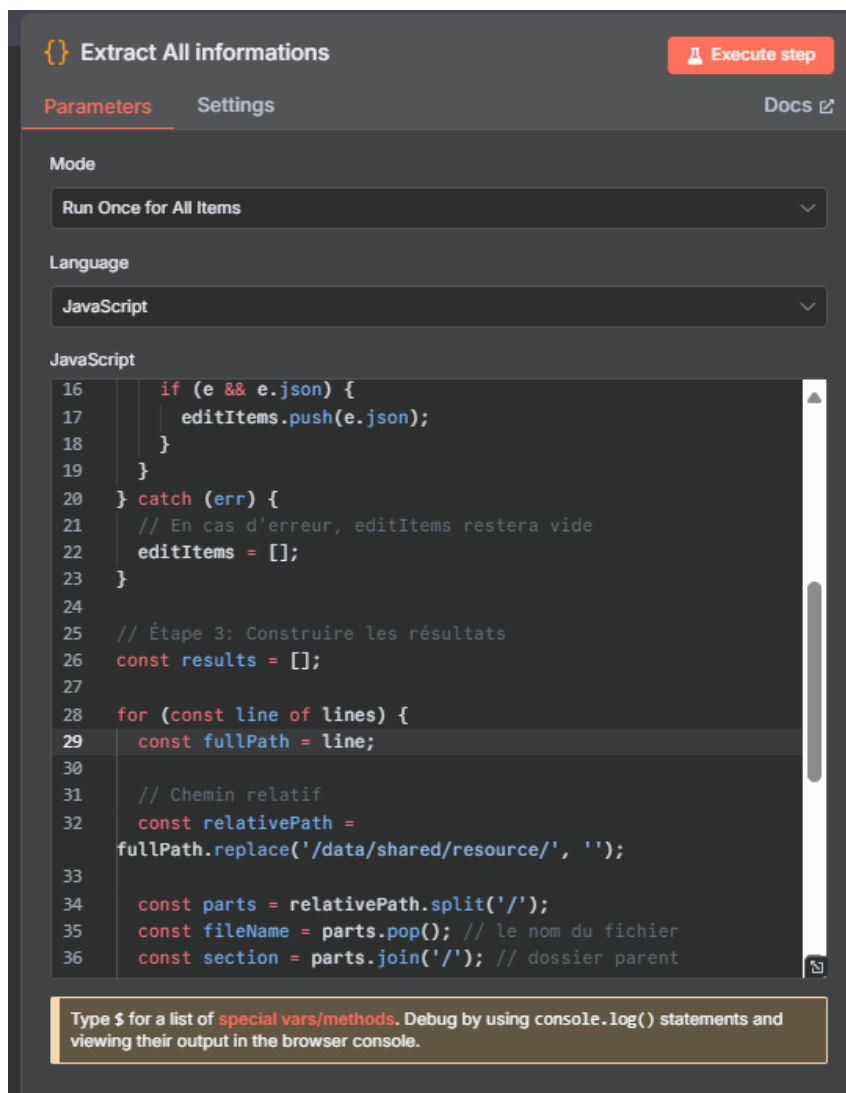


FIGURE 19 – Code for extracting information

10.7 Code Node 1.7 : Bad Format Cleanup

Role : Cleans the extracted content by removing unwanted characters (e.g., “”, line breaks) and escaping special characters before inserting into Neo4j.

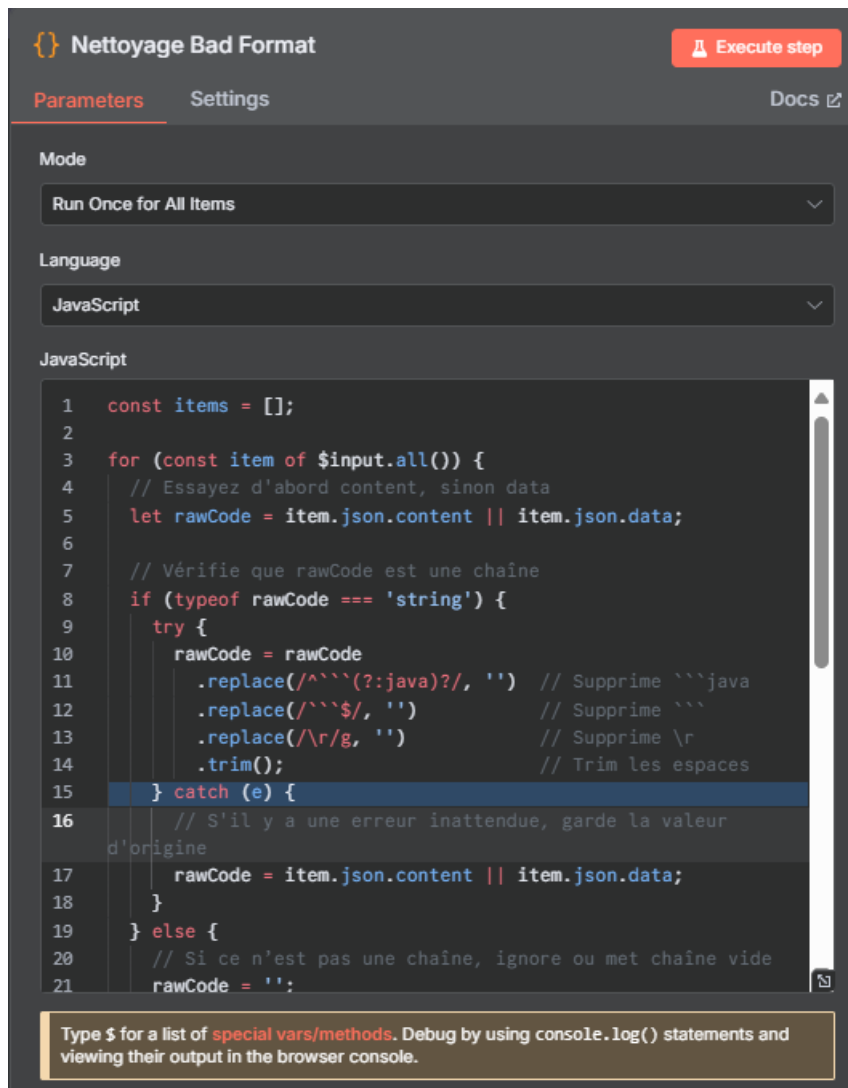


FIGURE 20 – Format cleanup code

10.8 Node 1.8 : Edit Fields

Role : Prepares final properties for Neo4j (file name, section, path, cleaned content).

The screenshot shows the 'Edit Fields' configuration window with the 'Parameters' tab selected. The 'Mode' is set to 'Manual Mapping'. Under 'Fields to Set', five fields are configured:

Field Name	Value	Type
file_name	{{ \$json.file }}	A String
section	{{ \$json.section }}	A String
PATH	{{ \$json.path }}	A String
content	{{ \$json.cleanedCode }}	A String
extractedAT	{{ \$json.extractedAt }}	A String

At the bottom, there is a dashed box containing the text: 'Drag input fields here or **Add Field**'.

FIGURE 21 – Final field configuration

10.9 Important : Creating the Neo4j Credential

In the first Neo4j node, click the first field + **Create new Credential**.

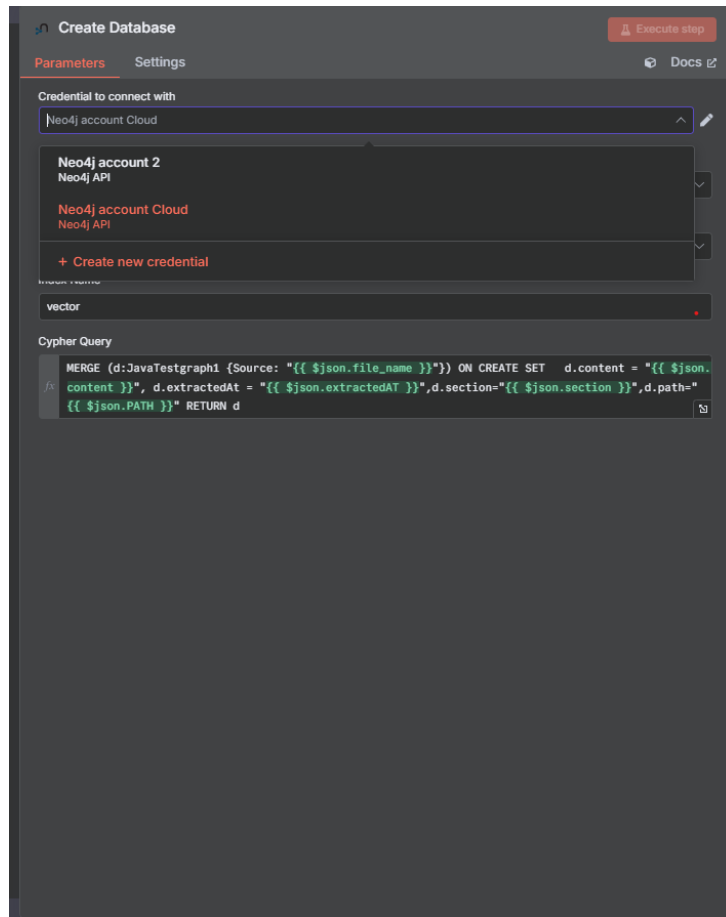


FIGURE 22 – Creating a Neo4j credential in n8n

10.10 Filling in the Connection Fields

Fill in the following fields using the values from the credentials file you downloaded when creating the Neo4j Aura instance :

- **Host** : NEO4J_URI (e.g., neo4j+s://xxxx.databases.neo4j.io)
- **Username** : NEO4J_USERNAME (typically neo4j)
- **Password** : NEO4J_PASSWORD (generated during instance creation)
- **Database** : NEO4J_DATABASE (often neo4j)

The screenshot shows the 'Neo4j account Cloud' configuration window in n8n. The window has a sidebar on the left with tabs: 'Connection' (selected), 'Sharing', and 'Details'. The main area contains the following fields:

- Connection URI ***: neo4j+s://9c42a465.databases.neo4j.io
- Username ***: neo4j
- Password ***: (masked with dots)
- Database ***: neo4j

At the bottom, there is a note: "Enterprise plan users can pull in credentials from external vaults. [More info](#)".

FIGURE 23 – Neo4j connection configuration in n8n

Important : Once created, the credential will automatically be used in other Neo4j nodes, and the same logic applies to other n8n nodes.

10.11 Node 1.9 : Create Database (Neo4j)

Role : Inserts nodes (files) into the Neo4j database with the following properties :

- **Source :** File name
- **content :** Cleaned content
- **section :** Parent folder
- **path :** Full file path

Cypher Query :

```
MERGE (d:YourGraphDatabaseName {Source: "{{ $json.file_name }}", path: "{{ $json.PATH
ON CREATE SET
  d.content = "{{ $json.content }}",
  d.section="{{ $json.section }}",
  d.extractedAt = "{{ $json.extractedAT }}",
  d.path = "{{ $json.PATH }}"
RETURN d
```

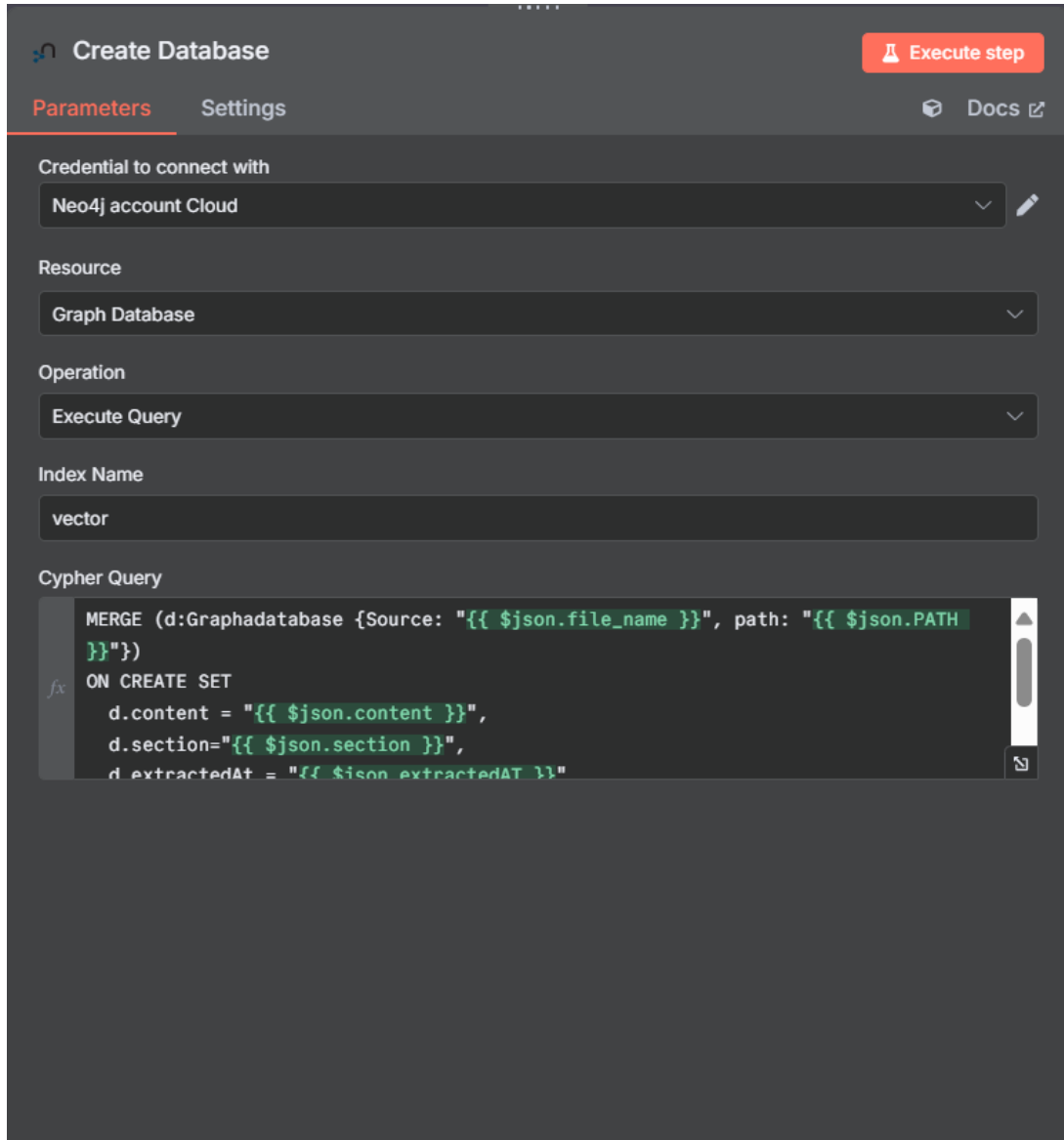


FIGURE 24 – Execute Query node for creating nodes

10.12 Relationship Nodes : Unique Pairs \rightarrow AI Agent \rightarrow Bidirectional Pairs \rightarrow ExecuteQuery graphDb

Overall Role : Build semantic relationships between files based on their content and sections, and insert them into Neo4j.

Processing Chain :

1. **Unique Pairs (Code Node)** Generates all file pairs (A, B) without repetition (no $A \rightarrow A$ and no inverse duplicates $B \rightarrow A$). Each pair includes :
 - source : {file, path, section, content}
 - target : {file, path, section, content}

Pairs Sans Doublons

Execute step

Parameters

Settings

Docs

Mode

Run Once for All Items

Language

JavaScript

JavaScript

```
1  const files = $input.all().map(item => item.json.d);
2
3  const results = [];
4
5  // Générer les paires sans doublons et sans auto-paires
6  for (let i = 0; i < files.length; i++) {
7    for (let j = i + 1; j < files.length; j++) {
8      const fileA = files[i];
9      const fileB = files[j];
10
11      results.push({
12        json: {
13          source: {
14            file: fileA.Source,
15            path: fileA.path,
16            section: fileA.section,
17            content: fileA.content
18          },
19          target: {
20            file: fileB.Source,
21            path: fileB.path,
22            section: fileB.section,
```

Type \$ for a list of **special vars/methods**. Debug by using `console.log()` statements and viewing their output in the browser console.

FIGURE 25 – Code to generate unique pairs

2. **AI Agent (LangChain)** Analyzes each (*source*, *target*) pair to determine :
 - type : Relationship type (complementary, uses, documents, associated, etc.)
 - justification : Concise explanation of the relationship
 - source_section, target_section
 - source, target (file names)
 - extractedAt : Generation timestamp

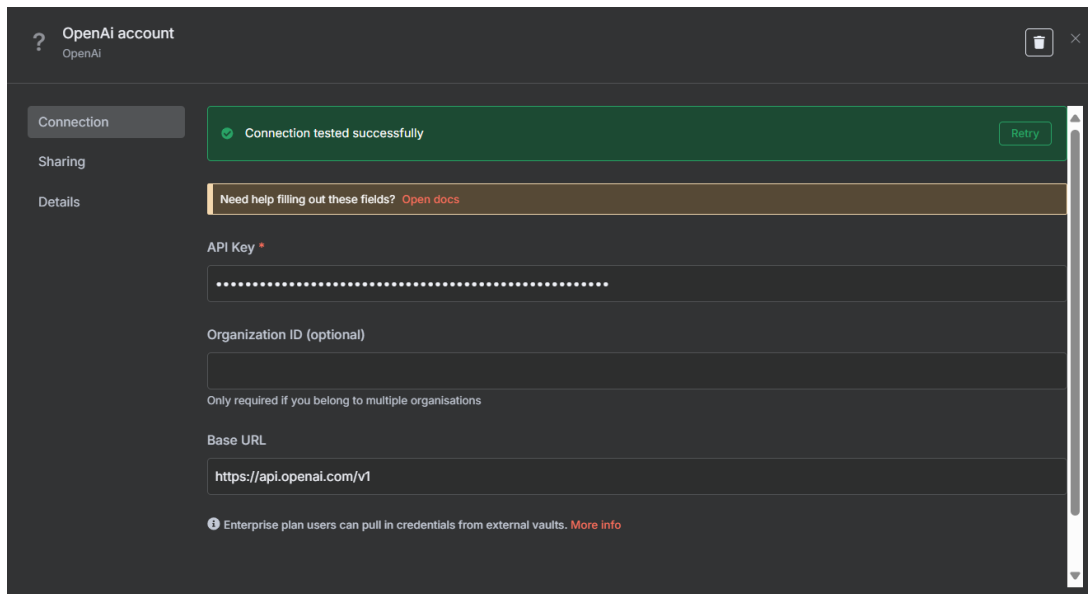
FIGURE 26 – AI agent configuration for relationship analysis

3. OpenAI GPT 4.1 mini Chat Model (Configuration)

- Go to the OpenAI Embeddings Node and click + **Create new Credentials**

FIGURE 27 – Creating a connection with OpenAI

- Fill in the connection fields using your OpenAI API Key : <https://platform.openai.com/api-keys>



The screenshot shows the 'OpenAI account' connection interface. At the top, there's a header with a question mark icon, 'OpenAI account', and 'OpenAI'. Below this is a sidebar with 'Connection', 'Sharing', and 'Details' tabs. The 'Connection' tab is active. A green banner at the top of the main area says 'Connection tested successfully' with a 'Retry' button. Below this is a brown banner that says 'Need help filling out these fields? Open docs'. The main form has three fields: 'API Key' (with a red asterisk and a masked input), 'Organization ID (optional)' (with a note 'Only required if you belong to multiple organisations'), and 'Base URL' (with the value 'https://api.openai.com/v1'). At the bottom, there's a small info icon and text: 'Enterprise plan users can pull in credentials from external vaults. More info'.

FIGURE 28 – OpenAI Connection Fields

4. **Extract + Clean Justification (Code Node)** This node has two main roles :
- (a) **Extraction** : Converts the raw output of the AI model (a single JSON block wrapped in text) into a structured object containing :
 - `source` and `target` : file names
 - `source_section` and `target_section`
 - `relationshipType`
 - `justification`
 - `extractedAt`
 - (b) **Cleaning** : Removes special characters, escapes quotes, and formats the justification for secure insertion into Neo4j.

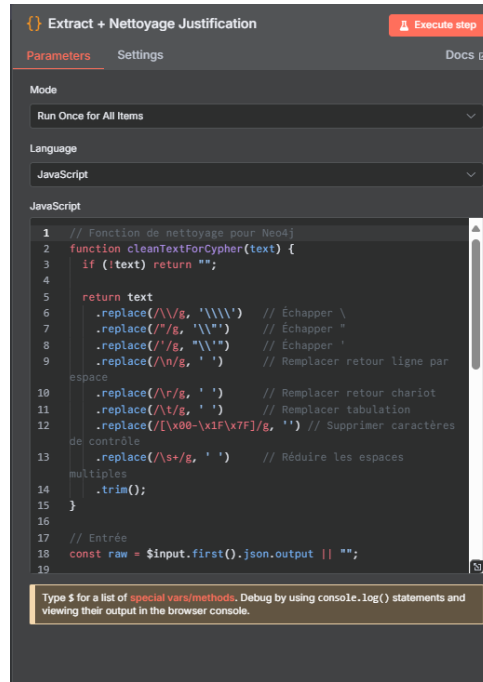


FIGURE 29 – Code for extraction and cleaning of justifications

5. **Bidirectional Pairs (Code Node)** From the AI output (A, B) , this node generates the inverse (B, A) to ensure bidirectionality. Each inverse relationship keeps :
- **relationshipType** : Same as original
 - **justification** : Same or annotated as inverse

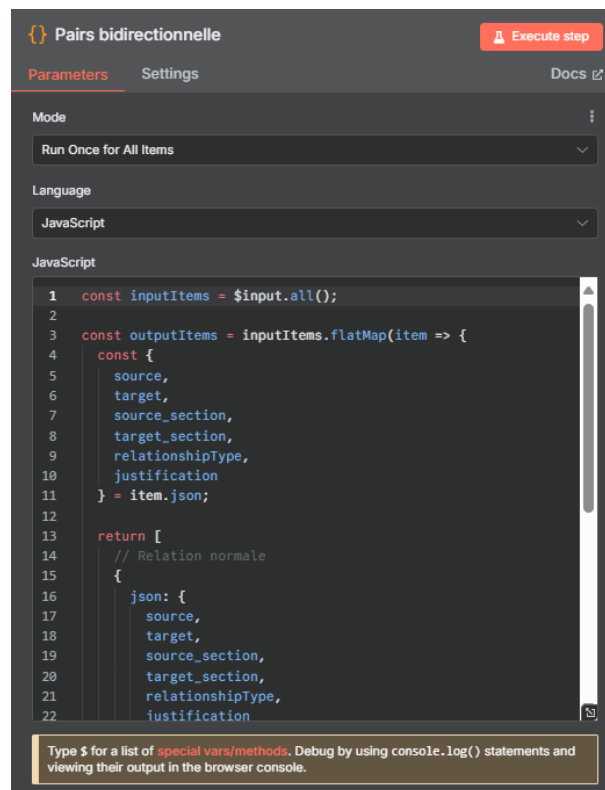


FIGURE 30 – Code for generating bidirectional pairs

6. **ExecuteQuery graphDb (Neo4j)** Inserts relationships between existing nodes using :

```
MERGE (src:YourGraphDatabaseName {Source: "{{ $json.source }}",section: "{{ $json.source_section }}"})
MERGE (tgt:YourGraphDatabaseName {Source: "{{ $json.target }}",section : "{{ $json.target_section }}"})
MERGE (src)-[rel:'{{ $json.relationshipType }}']->(tgt)
ON CREATE SET
    rel.justification = "{{ $json.justification }}",
    rel.source_section = "{{ $json.source_section }}",
    rel.target_section = "{{ $json.target_section }}"
RETURN src.Source, type(rel), tgt.Source
```

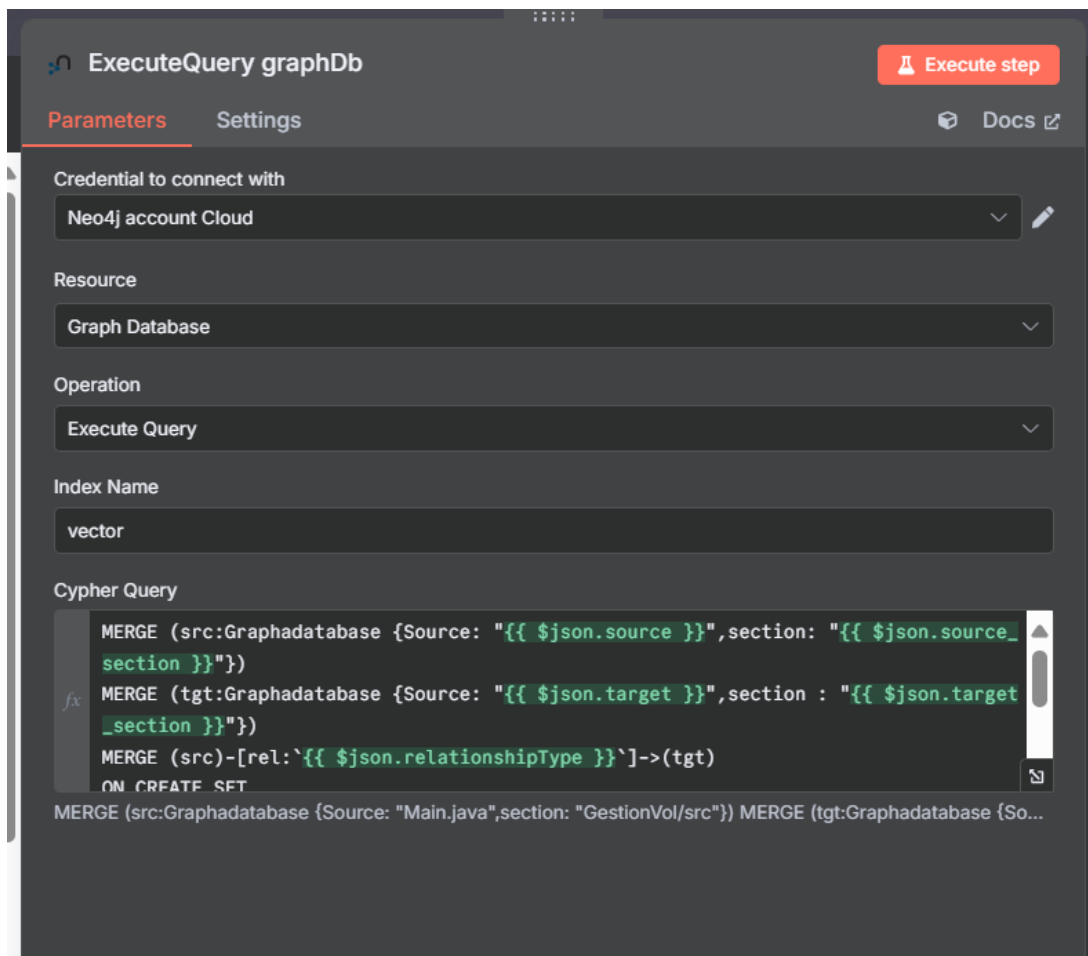


FIGURE 31 – Execute Query node configuration for creating relationships

Summary of inserted data :

- **source** → Source file name
- **target** → Target file name
- **relationshipType** → Relationship type
- **justification** → AI model reasoning
- **source_section, target_section** → Corresponding sections
- **extractedAt** → Generation date

11 Click the Execute Workflow Button

!!Important : Wait until the workflow finishes executing — this may take time depending on the number of resources.

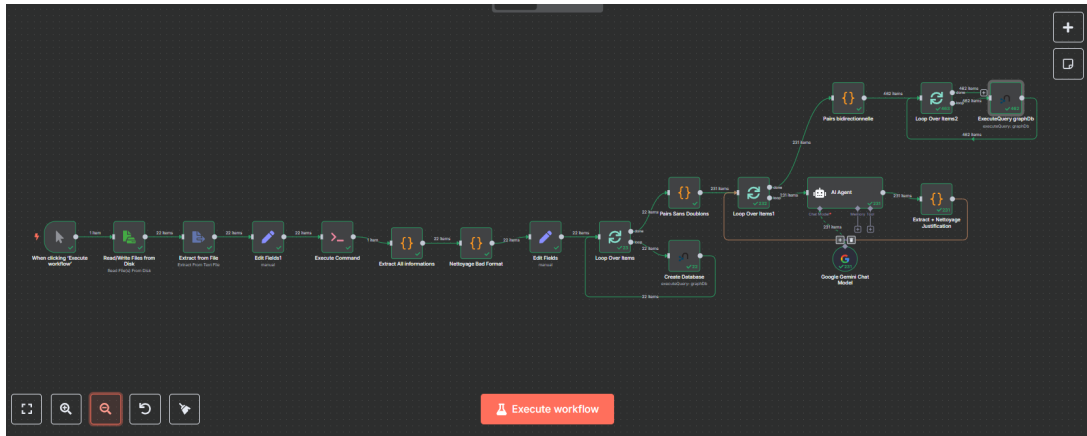


FIGURE 32 – Graph Database Workflow executed

12 Import the Second Workflow into n8n

1. Launch n8n
2. Go to <http://localhost:5678>
3. Menu Workflows > Create Workflow

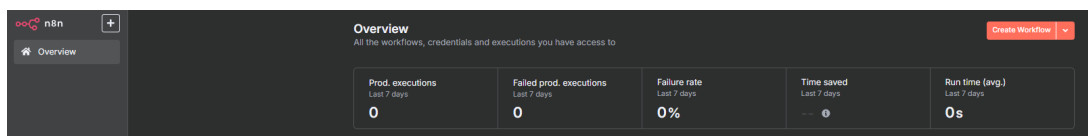


FIGURE 33 – n8n Home Interface

4. Click > Import from File

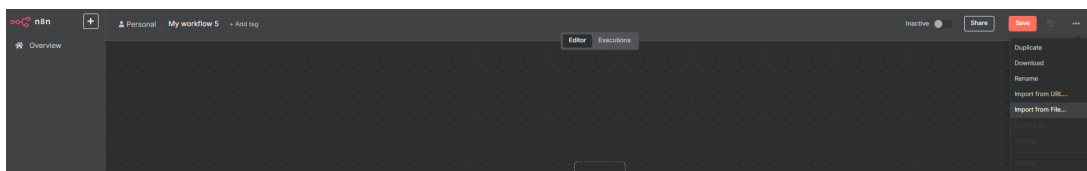


FIGURE 34 – Import from file

5. Select the file Graph RAG AGENT new resource JAVA,MD.json
6. Click Save

13 Workflow No. 2 : Graph RAG Agent

13.1 Architecture

Chat Message → AI Agent → OpenAI Chat Model → Postgres Memory → Neo4j Graph Database (Tool)

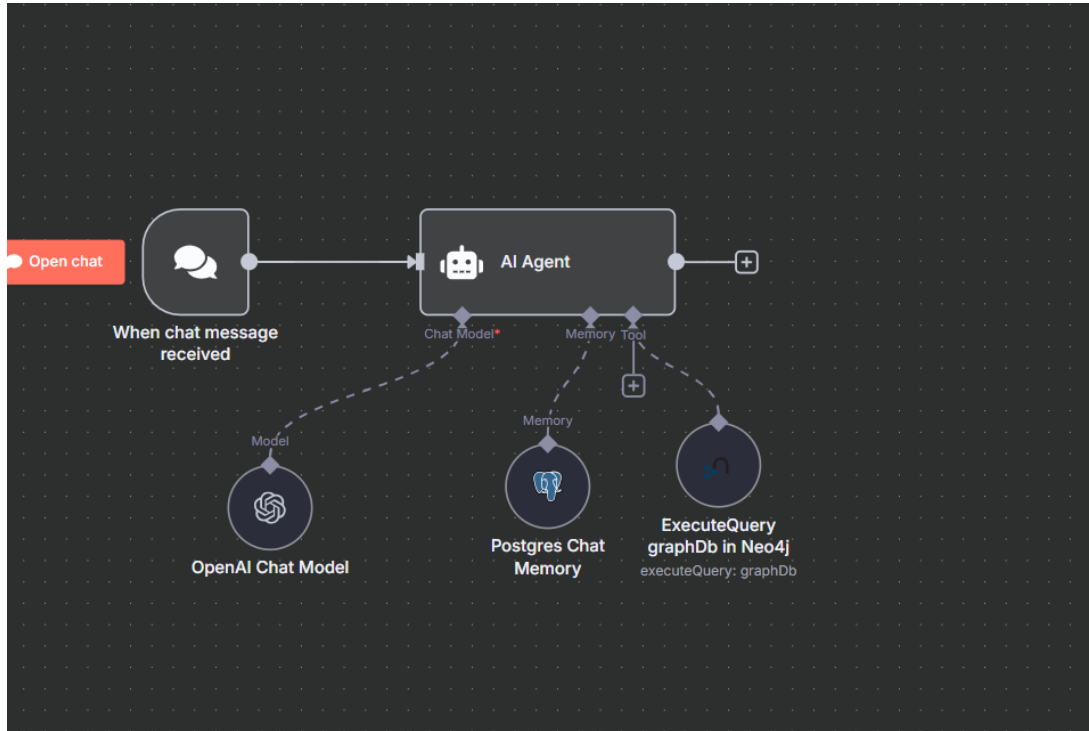


FIGURE 35 – Conversational interface for the Neo4j-based Graph RAG Agent

General Description : The intelligent agent (AI Agent) receives a user question, queries the Neo4j graph database to retrieve related Java files and documentation, and generates an enriched response explaining the relationships between the files.

13.2 Node 2.1 : When Chat Message Received

Role : Starting point of the workflow. **Function :** Triggers the flow each time a user sends a message in the chat.

13.3 Node 2.2 : AI Agent

Role : Main Graph RAG agent. **Function :** Analyzes the user query, interacts with Neo4j, and composes a response.

Uses :

- A language model to understand and generate the response.
- A memory (Postgres) to store the conversation context.
- A Neo4j tool to retrieve files and their relationships.

Combination :

- **Chat Model** → understanding and generation
- **Memory** → conversational context

- **Tool (Neo4j)** → retrieve relevant nodes and relationships

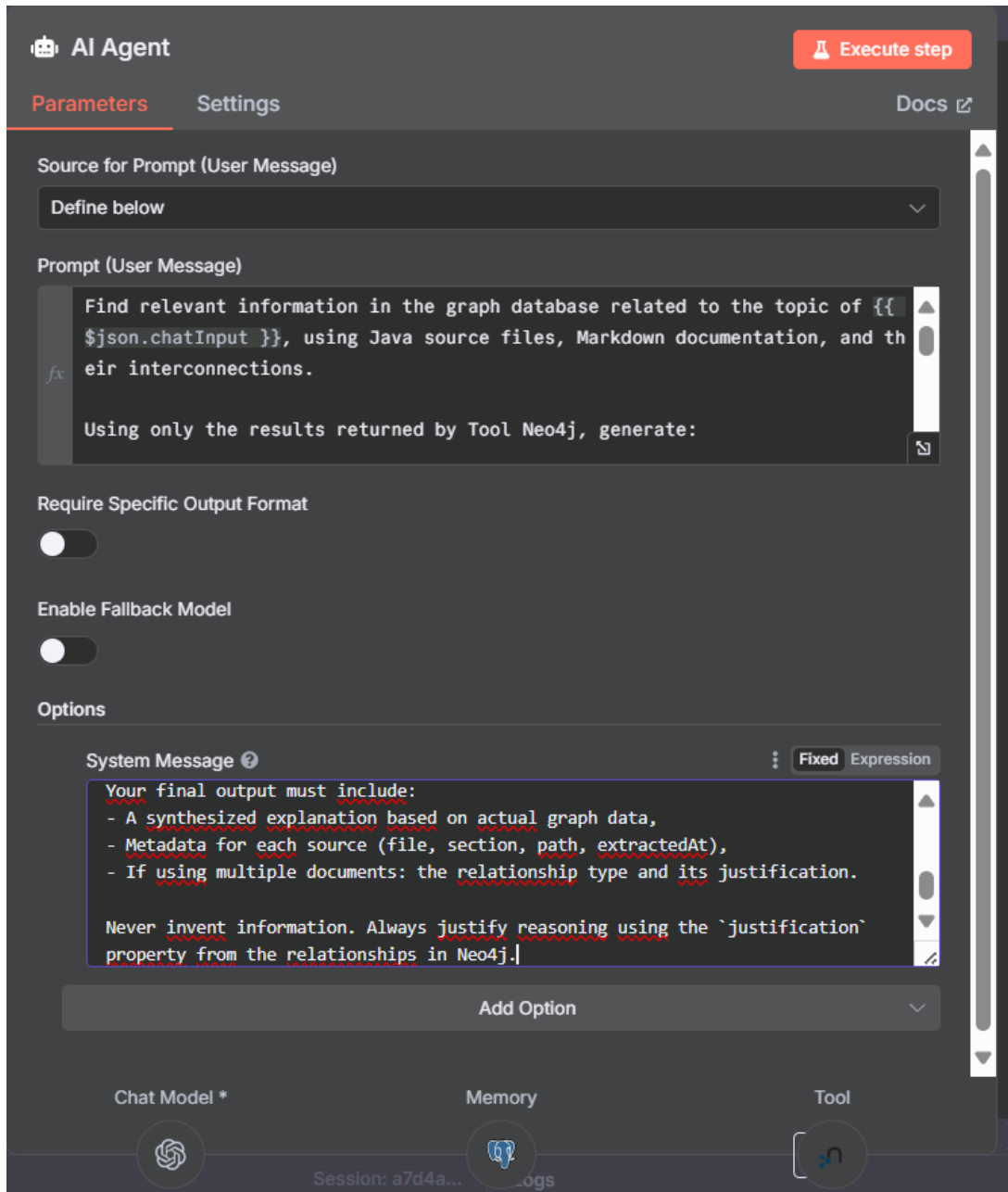


FIGURE 36 – AI Agent configuration connected to Neo4j, Postgres, and OpenAI

System Message :

You are a Graph RAG AI agent specialized in analyzing Java source files and their related Markdown documentation stored in a Neo4j knowledge graph.

Your main tool is the Neo4j database (Tool Neo4j), which you must query to extract relevant files and their semantic relationships.

Rules :

- Only use data returned from Neo4j (no hallucinations).
- Prioritize relationships : **documents**, **complementary**.
- Include metadata : file name, section, path, extractedAt, snippet of content.
- Respond in the language of the query.

Expected format :

- Concise and reasoned response.
- List of files used with their relationship and justification.
- Multi-document citations if needed.

13.4 Node 2.3 : OpenAI Chat Model

Role : Chat GPT 4o mini **Function :** Generates the response based on graph data.
Connected to :

- AI Agent via Chat Model.

Configuration :

- Add the option *Sampling Temperature* set to 0.0 to eliminate hallucination risk.
- Use the previously created OpenAI credential.

13.5 Node 2.4 : Postgres Chat Memory

Role : Conversational memory. **Function :** Stores conversation history to maintain context.

Configuration :

- **Memory Type :** Postgres
- **Connection :** Requires creating a PostgreSQL database credential
- **Required Parameters** from the `.env` file in `self-hosted-ai-starter-kit` :
 - **Host :** postgres (service name in `docker-compose.yml`)
 - **Port :** 5432 (default PostgreSQL port)
 - **Database :** \$POSTGRES_DB
 - **User :** \$POSTGRES_USER
 - **Password :** \$POSTGRES_PASSWORD

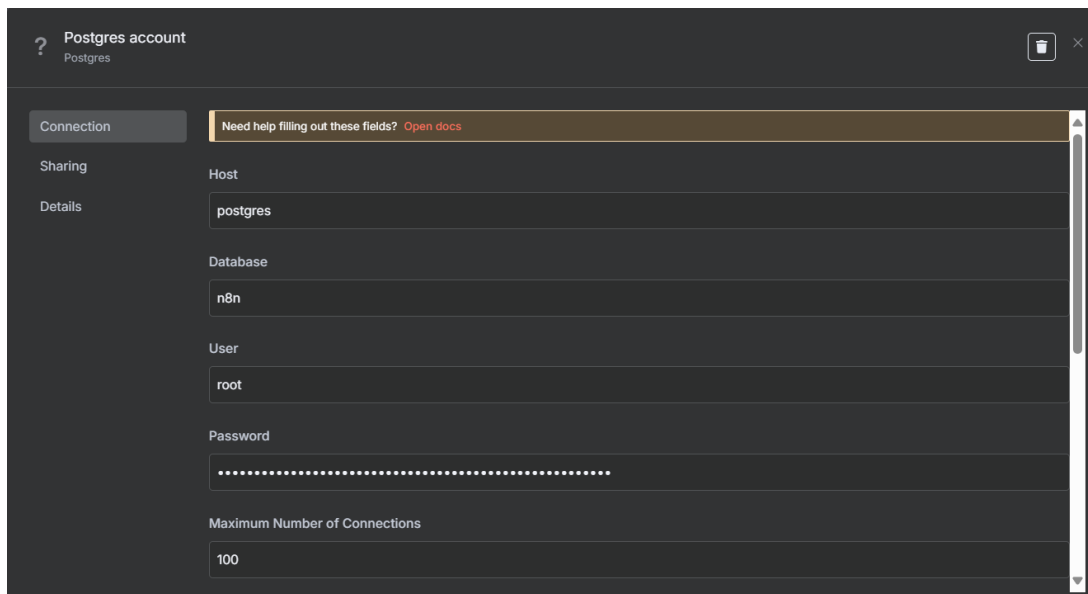
The image shows a dark-themed web interface for configuring a PostgreSQL connection. At the top, there's a header with a question mark icon, the text 'Postgres account', and a 'Postgres' label. To the right of the header is a trash icon and a close 'x' button. Below the header, there's a sidebar on the left with three tabs: 'Connection' (selected), 'Sharing', and 'Details'. The main content area has a top bar with the text 'Need help filling out these fields?' and a link 'Open docs'. Below this, there are several input fields: 'Host' with the value 'postgres', 'Database' with the value 'n8n', 'User' with the value 'root', and 'Password' which is masked with dots. At the bottom, there's a field for 'Maximum Number of Connections' with the value '100'. A vertical scrollbar is visible on the right side of the main content area.

FIGURE 37 – PostgreSQL connection configuration

Example .env content :

```

POSTGRES_USER=root
POSTGRES_PASSWORD=password
POSTGRES_DB=n8n

N8N_ENCRYPTION_KEY=super-secret-key
N8N_USER_MANAGEMENT_JWT_SECRET=even-more-secret
N8N_DEFAULT_BINARY_DATA_MODE=filesystem

# For Mac users running OLLAMA locally
# See https://github.com/n8n-io/self-hosted-ai-starter-kit?tab=readme-ov-file#for-mac--apple-silicon-users
# OLLAMA_HOST=host.docker.internal:11434

```

FIGURE 38 – Default .env file content

Connection : This node is connected to the AI Agent via the **Memory** option.

13.6 Node 2.5 : ExecuteQuery graphDb in Neo4j

Role : Query the Neo4j database to extract files and their relationships. **Function** : Executes the Cypher query to retrieve nodes and relations.

Cypher Query :

```

MATCH (n:YourGraphDatabaseName) OPTIONAL MATCH (n)-[r]->(m)
OPTIONAL MATCH (m)-[r2]->(n) RETURN n, r, m, r2

```

The screenshot shows the configuration interface for the 'ExecuteQuery graphDb in Neo4j' node. The interface is divided into two tabs: 'Parameters' (selected) and 'Settings'. A red 'Execute step' button is in the top right corner. The 'Parameters' tab contains the following fields:

- Credential to connect with**: A dropdown menu showing 'Neo4j account Cloud'.
- Tool Description**: A dropdown menu showing 'Set Automatically'.
- Resource**: A dropdown menu showing 'Graph Database'.
- Operation**: A dropdown menu showing 'Execute Query'.
- Index Name**: A text input field containing 'vector'.
- Cypher Query**: A text area containing the query:


```

MATCH (n:JavaTestgraph1) OPTIONAL MATCH (n)-[r]->(m) OPTIONAL MATCH (m)-[r2]
->(n) RETURN n, r, m, r2

```

Below the Cypher Query field, there is a preview of the query: 'MATCH (n:JavaTestgraph1) OPTIONAL MATCH (n)-[r]->(m) OPTIONAL MATCH (m)-[r2]->(n) RETURN n, r, ...'.

FIGURE 39 – Neo4j node configuration to execute queries

14 Test the Graph RAG

After completing the configuration, you can test the Graph RAG by typing a query in the n8n interface and sending it to receive a response.

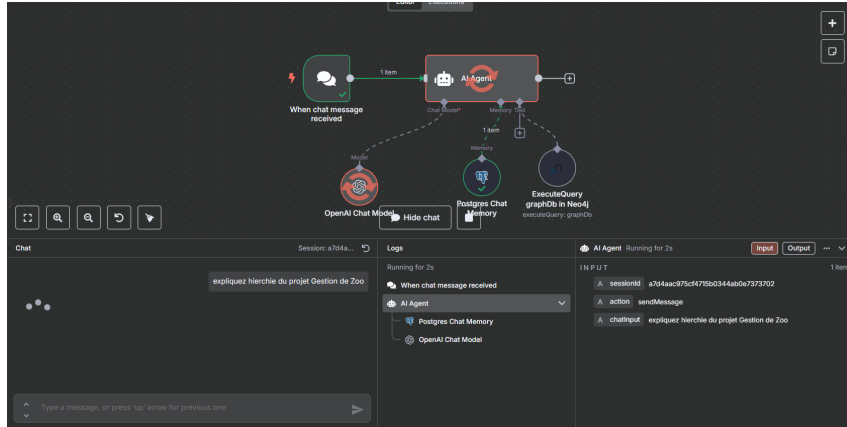


FIGURE 40 – Example query sent in n8n to test the Graph RAG

15 Conclusion

This guide helped you set up a complete Graph RAG (Retrieval-Augmented Generation) system using **n8n** for orchestration, **Neo4j** for graph data management, and advanced AI models for semantic analysis and response generation.

Benefits of the Graph RAG approach :

- **Advanced contextualization** : Semantic relationships between documents allow for deeper understanding and interconnections.
- **Enriched responses** : Responses leverage relationships across multiple documents for coherent answers.
- **Complete traceability** : Each response includes source metadata (file, section, justification).
- **Scalability** : The modular architecture makes it easy to add new document types and relationships.

Practical applications :

- **Smart documentation** : Automatic analysis of Java codebases and their associated Markdown documentation.
- **Developer assistant** : Helps understand relationships between software components.
- **Technology watch system** : Analyze links across technical and research documents.
- **Enterprise knowledge base** : Smart organization and querying of internal documentation.

Future extensibility : The system can be extended to include other file types (PDF, Word, Excel), other graph databases, or specialized AI models based on your organization's needs.

This solution provides a solid foundation for developing advanced AI applications while maintaining full control over your data and leveraging the power of graph-based

reasoning.