

Manual de Usuario: Agente de IA para RAG en n8n

Juan David Torres Avila

06/06/2025

1. Propósito

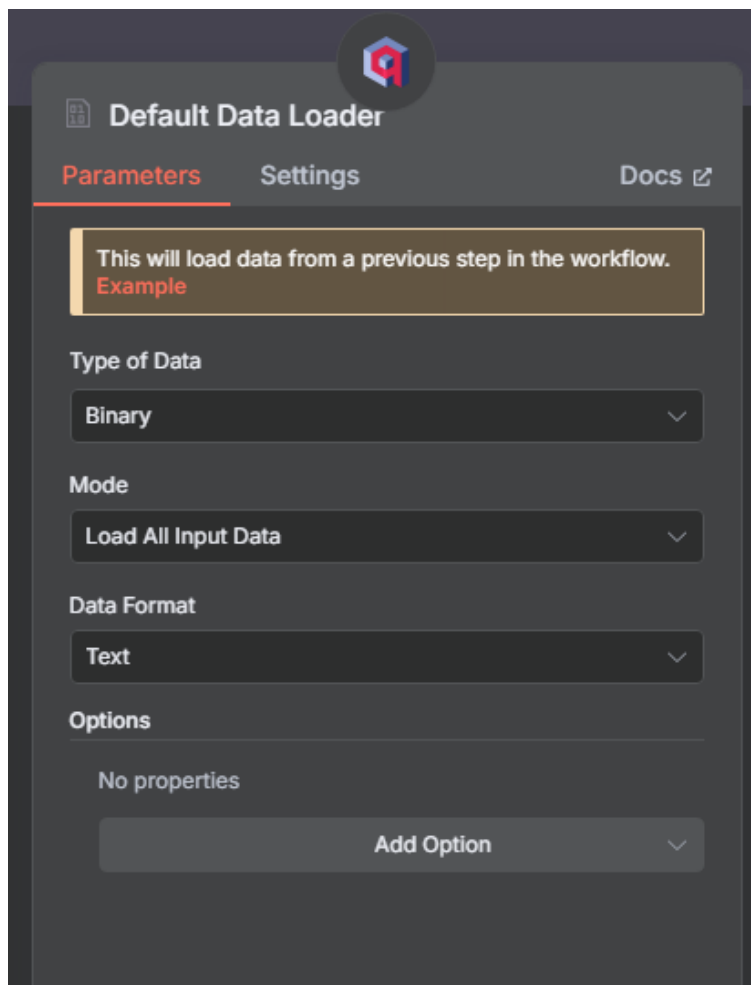
Este flujo de trabajo permite a un agente de inteligencia artificial (IA) responder preguntas en base al contenido de un documento de texto utilizando Recuperación Aumentada por Generación (RAG). Se emplean las siguientes tecnologías:

- Qdrant como vector store.
- Ollama para generación de texto e incrustaciones (embeddings).
- n8n como orquestador del flujo.

2. Requisitos Previos

Para la correcta ejecución del flujo, asegúrese de tener:

- Una cuenta y credenciales configuradas para:
 - Qdrant
 - Ollama
 - OpenIA (Opcional)
- Archivo de texto cargado localmente o archivo con el formato seleccionado en el Default Data Loader (nodo en el flujo de trabajo de n8n).

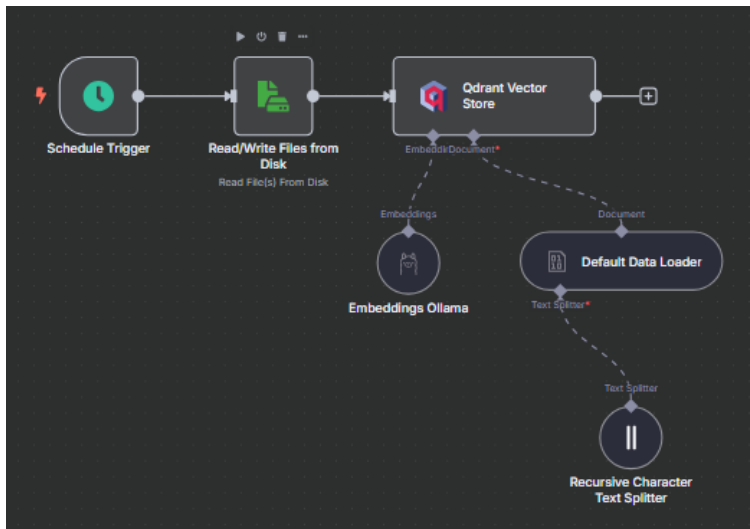


(Nodo donde se configura el tipo de dato esperado)

- Las credenciales conectadas en n8n:
 - QdrantApi account
 - Ollama account
 - Modelo de Ollama que dese utilizar
 - Modelo para el embedding con Ollama

3. Estructura del Flujo

3.1. Carga y procesamiento del documento



- **Schedule Trigger:** Dispara el flujo automáticamente en un intervalo definido.
- **Read/Write Files from Disk:** Lee el archivo desde el disco en la ruta: D:\Descargas\Codigo 2\Prueba-datos\inteligencia_artificial.txt.
- **Embeddings Ollama:** Convierte el texto en vectores usando el modelo `nomic-embed-text`.
- **Default Data Loader:** Carga los datos como documentos compatibles.
- **Recursive Character Text Splitter:** Divide el texto en fragmentos de 800 caracteres con un solapamiento de 100.
- **Qdrant Vector Store:** Inserta los vectores en la colección Prueba-datos.

3.2. Carga y procesamiento del documento (forma alterna, fuera del flujo)

Importación de librerías

```
import openai
from qdrant_client import QdrantClient
from qdrant_client.models import VectorParams, Distance, PointStruct
import PyPDF2
```

Estas librerías permiten leer PDFs, conectarse a Qdrant y usar los embeddings de OpenAI.

Configuración de API y parámetros globales

```
openai.api_key = "sk-proj-..."
```

```
QDRANT_URL = "http://localhost:6333"
```

```
COLLECTION_NAME = "documentos_pdf"
```

```
EMBEDDING_MODEL = "text-embedding-3-small"
```

```
VECTOR_SIZE = 1536
```

Se define la clave de API de OpenAI, URL de Qdrant, nombre de colección, modelo y tamaño de vectores.

Lectura de PDF

```
def leer_pdf(path):
    with open(path, "rb") as f:
        reader = PyPDF2.PdfReader(f)
        texto = "-".join([p.extract_text() for p in reader.pages if p.extract
        return texto
```

Extrae el texto de cada página de un archivo PDF.

División del texto

```
def dividir_texto(texto, tamaño=500):
    return [texto[i:i+tamaño] for i in range(0, len(texto), tamaño)]
```

Divide el texto completo en fragmentos de tamaño especificado (500 por defecto).

Obtención de embeddings

```
def obtener_embeddings(textos):
    response = openai.embeddings.create(model=EMBEDDING_MODEL, input=textos)
    return [r.embedding for r in response.data]
```

Usa OpenAI para convertir texto en vectores numéricos (embeddings).

Subida a Qdrant

```
def subir_a_qdrant(vectores, textos):
    client = QdrantClient(url=QDRANT_URL)
    if not client.collection_exists(COLLECTION_NAME):
        client.recreate_collection(
            collection_name=COLLECTION_NAME,
```

```

        vectors_config=VectorParams(size=VECTOR_SIZE, distance=Distance.COSINE)
    )
    puntos = [PointStruct(id=i, vector=v, payload={"texto": textos[i]}) for i, v in enumerate(vectors)]
    client.upsert(collection_name=COLLECTION_NAME, points=puntos)
    print(f"{len(puntos)} fragmentos subidos a Qdrant.")

```

Crea una colección si no existe, estructura los puntos y los sube a Qdrant.

Bloque principal

```

if __name__ == "__main__":
    texto = leer_pdf("documento.pdf")
    fragmentos = dividir_texto(texto)
    embeddings = obtener_embeddings(fragmentos)
    subir_a_qdrant(embeddings, fragmentos)

```

Ejecuta todas las funciones: lectura, división, embeddings y subida a Qdrant.

Comando de ejecución

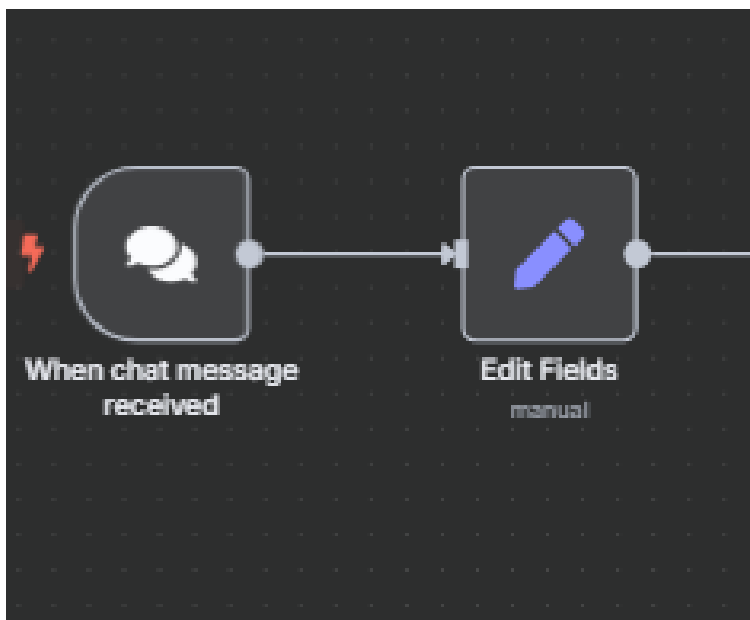
Para ejecutar el código anterior, desde PowerShell con Dirección de la carpeta:

```
cd "C:\Users\Cornifer\Downloads"(Ejemplo de una dirección de carpeta)
```

Archivo de código pythom con el progrma anteriormente explicado:

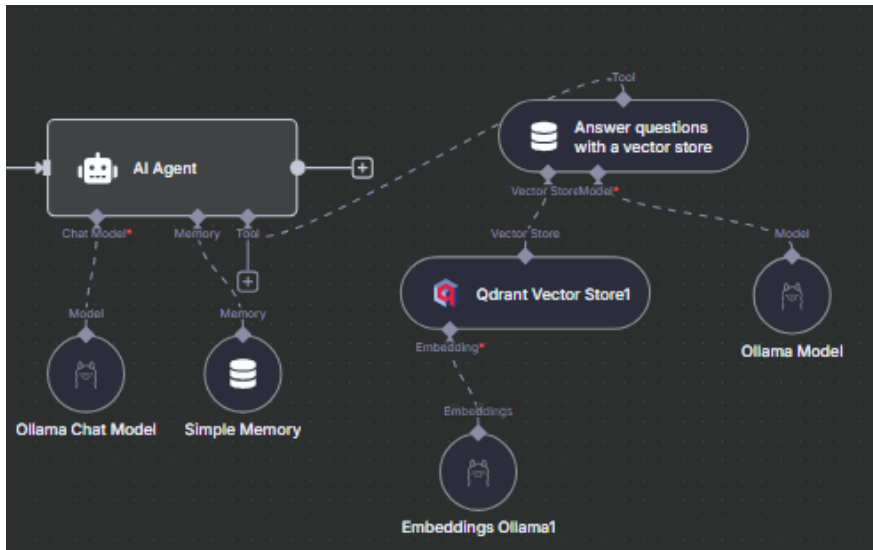
```
python cargar_txt_a_qdrant_ruta_personalizada.py
```

3.3. Interacción del usuario



- **When chat message received:** Captura el input del usuario.
- **Edit Fields:** Asigna y normaliza los campos `chatInput` y `sessionId`.

3.4. 3. Agente Inteligente RAG



- **Ollama Chat Model:** Modelo conversacional.
- **Simple Memory:** Guarda el historial de conversación con clave personalizada.
- **Qdrant Vector Store1:** Realiza búsqueda de contexto en la colección `ia_txt`.
- **Embeddings Ollama1:** Genera vectores para la consulta del usuario.
- **Ollama Model:** LLM que responde usando el contexto encontrado.
- **Answer questions with a vector store:** Herramienta que consulta el vector store.
- **AI Agent:** Une todos los componentes para generar respuestas con contexto.

4. Ejecución del Flujo

A. Manual

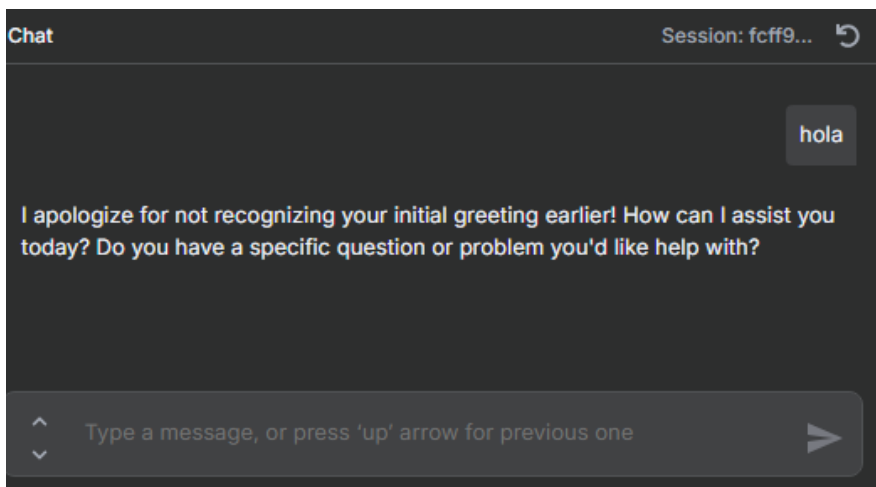
Ejecute abriendo el chat del flujo y empezando con la interacción o desde cualquier punto del flujo (para probar los diferentes nodos).

B. Automática (API o Webchat)

Envíe una solicitud POST al webhook generado por el nodo **When chat message received** con la siguiente estructura:

```
{  
  "chatInput": " Que es la inteligencia artificial?",  
  "sessionId": "usuario123"  
}
```

5. Resultado Esperado



El agente IA responderá basándose en la información contenida en el documento, consultando el vector store con embeddings y generando una respuesta contextual con Ollama.

6. Notas Técnicas

- El texto se divide en fragmentos para mejorar la eficiencia semántica.
- Se emplea memoria conversacional para mantener el contexto.
- Asegúrese que la colección `ia.txt` estén creadas en Qdrant.