CREACIÓN DE UN AGENTE RAG EN N8N

¿Qué es un agente RAG?

Un agente RAG (Retrieval-Augmented Generation) es un tipo de sistema de inteligencia artificial que combina generación de texto (como la que hacen los modelos tipo GPT) con búsqueda de información en una base de datos o repositorio para producir respuestas más precisas, actualizadas y basadas en datos reales.

¿Qué hace un agente RAG?

1. Recupera información, busca documentos, textos, PDFs, páginas o fragmentos relevantes en una base de datos (por ejemplo, Qdrant, Pinecone, Weaviate, etc.).

2. Los envía al modelo de IA y el modelo genera una respuesta usando la información que recuperó, no solo lo que tiene entrenado.

3. Produce una respuesta más confiable. La IA responde basándose en datos reales, específicos y personalizados.

Pre-requisitos:

- Instalar Docker en el computador

Para esto vamos al enlace https://docs.docker.com/desktop/setup/install/windows-install/ donde elegimos la versión de Docker que deseemos instalar de acuerdo al SO del PC.

- Crear el archivo con extensión yml llamado “docker-compose” el cual va a tener:

  ✓ Los servicios de Qdrant en el puerto 6333.
  ✓ El servicio de Ollama en el puerto 11434
  ✓ Y el servicio de n8n en el puerto 5678
  ✓ Además, el URLs de Ngrok debe estar en el apartado de Webhook para redirigir los mensajes de Telegram al tigger de n8n.

  Este archivo debe estar ubicado en una carpeta local destinada para el flujo de trabajo.

```
version: '3.8'

services:
  # Servicio 1: Qdrant (Base de Datos Vectorial)
  qdrant:
    image: qdrant/qdrant:latest
    container_name: qdrant
    ports:
      - "6333:6333"
      - "6334:6334"
    volumes:
      - qdrant_storage:/qdrant/storage
    restart: unless-stopped
    networks:
      - rag-network

  # Servicio 2: Ollama (Servidor de Modelos LLM)
  ollama:
    image: ollama/ollama:latest
    container_name: ollama
    ports:
      - "11434:11434"
    volumes:
      # Mapeamos la carpeta de datos del contenedor
      - "./ollama-data:/root/.ollama"
    restart: unless-stopped
    networks:
      - rag-network

 # Servicio 3: n8n (Orquestador de Flujos)
 n8n:
    image: n8nio/n8n:latest
    container_name: n8n
    ports:
      - "5678:5678" # Puerto de la interfaz web de n8n (Acceso local: http://localhost:5678)
    volumes:
      - n8n_data:/home/node/.n8n
    environment:
      # --- CAMBIOS AÑADIDOS PARA EL WEBHOOK/HTTPS ---
      - N8N_HOST=0.0.0.0
      - N8N_PORT=5678
      # 1. Indicamos a n8n que use HTTPS (aunque sea a través de ngrok)
      - N8N_PROTOCOL=https
      # 2. Variable crucial: URL que n8n debe usar para crear el webhook en Telegram
      #    DEBES REEMPLAZAR ESTO con la URL HTTPS que te dé ngrok (o tu dominio si usas proxy)
      - WEBHOOK_URL=https://hazy-overgrossly-leandra.ngrok-free.dev
      # 3. Permite que n8n confíe en el encabezado X-Forwarded-For de un proxy/ngrok
      - N8N_BASIC_AUTH_ACTIVE=false
      - N8N_SECURE_COOKIE=false
      # ------------------------------------------------
    networks:
      - rag-network
    restart: always

# Configuración de los volúmenes para persistir datos
volumes:
  qdrant_storage:
  n8n_data:
  # El volumen 'ollama_data' se eliminó de aquí ya que usa una ruta de host (./ollama-data)
  # en su lugar.

# Red interna compartida
networks:
  rag-network:
    driver: bridge
```

- Instalar Ollama de forma local usando el enlace https://ollama.com/download/windows para luego proceder con la instalación del modelo de Ollama que sea adecuado de acuerdo a las especificaciones del PC.

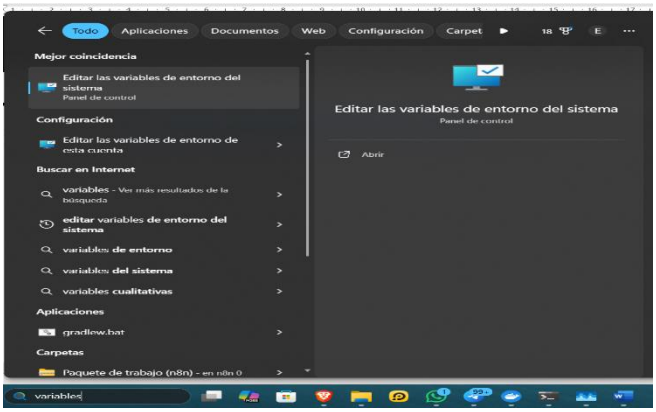
- Descargar el modelo de chat y embbengding

  ✓ Para esto abrimos la terminal y pegamos el comando ollama pull “modelo de ollama” en nuestro caso instalamos el modelo Llama 3.1:8b, así que usamos el comando “ollama pull llama3.1:8b” y se comenzará a descargar el modelo de chat.

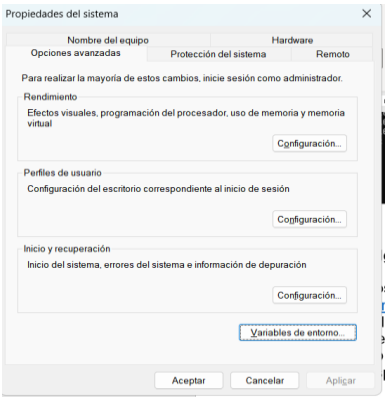
  ✓ Después podemos usar “ollama pull nomic-embed-text” para instalar un modelo de emmbending que usaremos luego en n8n, este modelo maneja 768 dimensiones.

- Descargar Ngrok

Para esto vamos a https://ngrok.com/download/windows?tab=download y descargamos el archivo.exe en alguna carpeta, puede ser en la que tienen para el proyecto o en cualquier otra, en nuestro caso lo descargamos en C:\Program Files\Ngrok después vamos a variables de entorno.

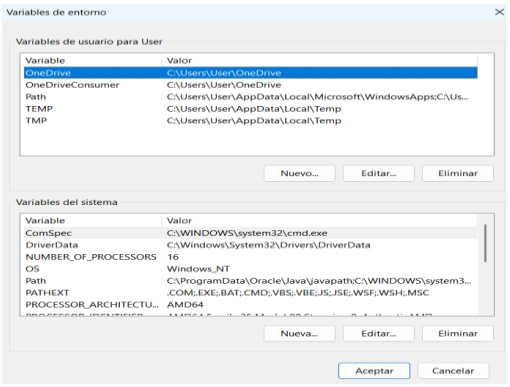


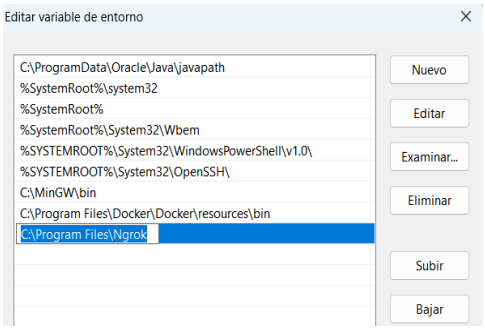
Luego a variables de entorno



Luego vamos al path de variables

del sistema y le damos en editar

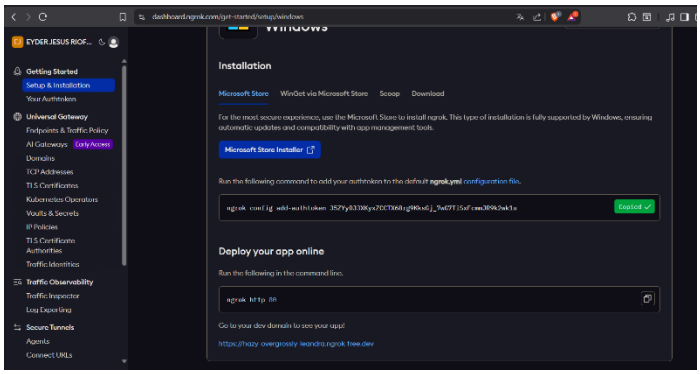




Luego agregamos uno nuevo y pegamos

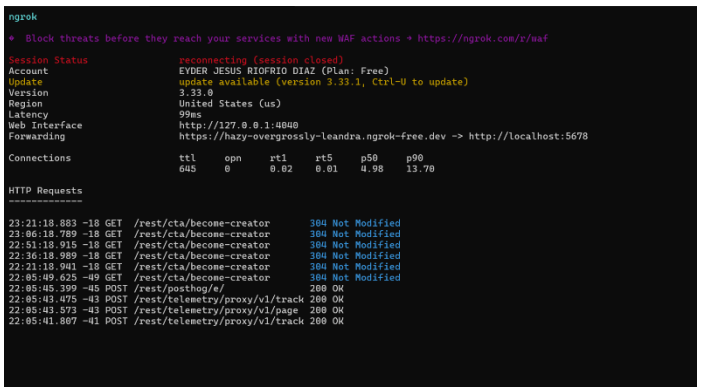
la dirección donde está el ejecutable de ngrok

Y ya con esto pdremos usar los comandos de ngrok desde la consola, luego ya podrémos ir a https://dashboard.ngrok.com/get-started/setup/windows iniciamos sesión y copiamos el token que nos da ngrok y lo

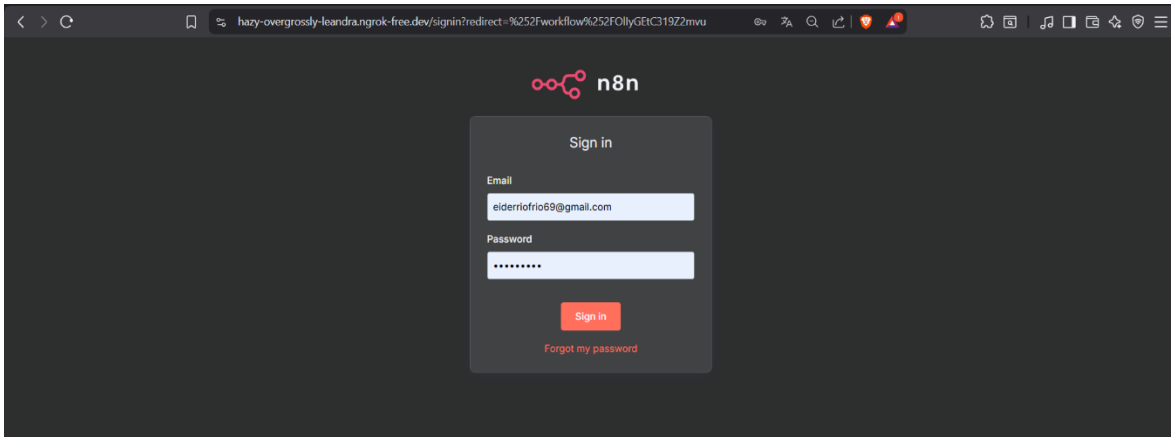


pegamos en la terminal.

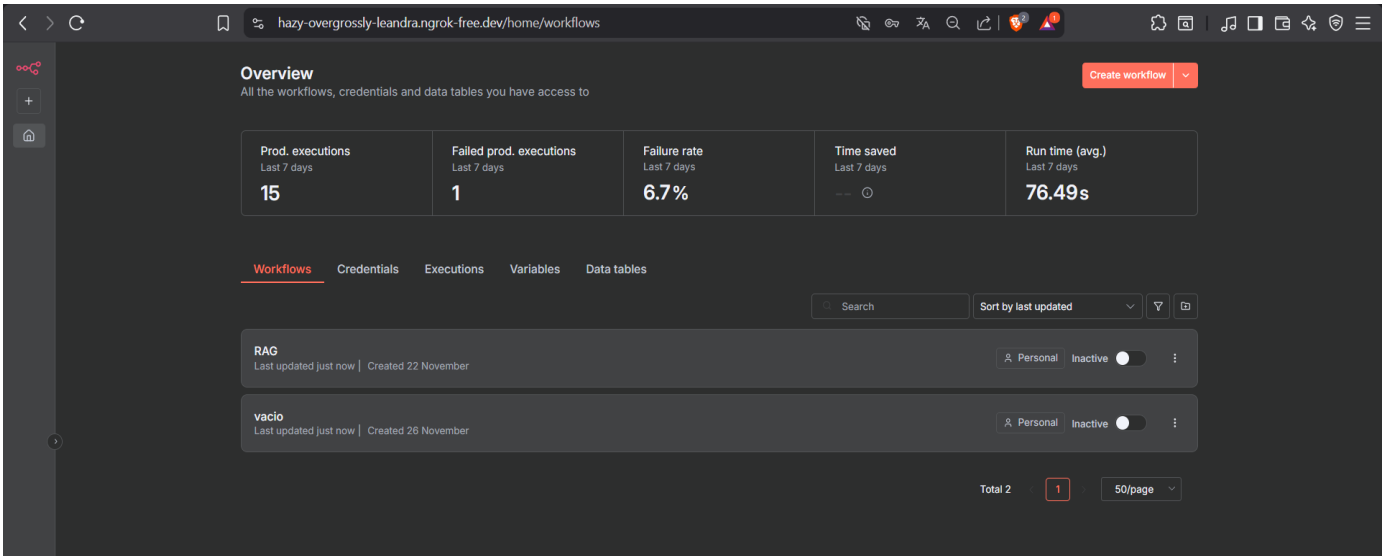
Luego ejecutamos el comando ngrok “puerto”, en nuestro caso usamos el puerto donde está n8n: ngrok 5678 y copiamos la dirección https que nos dé Ngrok ya que esa es la que deberemos usar para acceder a n8n y esta misma debe estar dentro de docker-compose.yml en el espacio de Webhook.



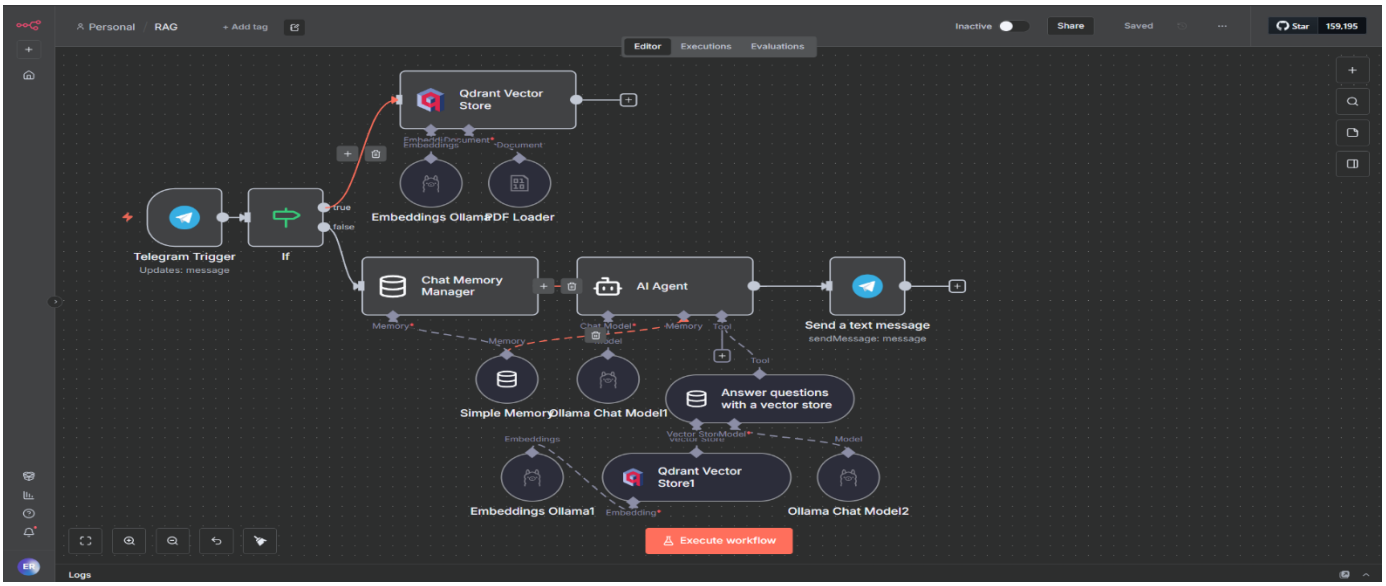
Ya aquí podemos pegar la dirección que nos dio Ngrok en el navegador y accedemos con nuestra cuenta de n8n.



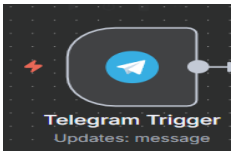
Y ya tenemos nuestro n8n ejecutándose de manera local pero abierta a internet para poder conectarse a Telegram.

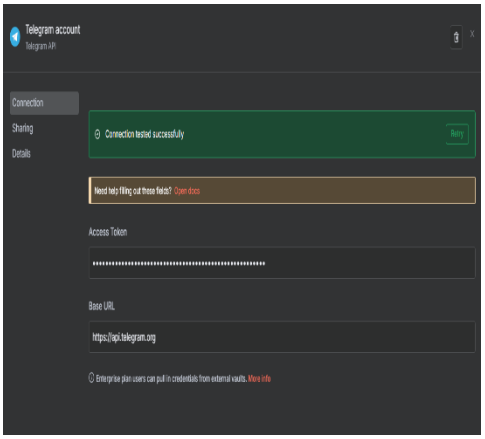


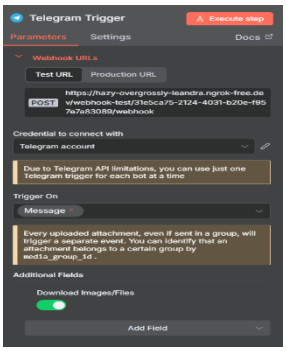
FLUJO



1. Tigger on message de Telegram → If node





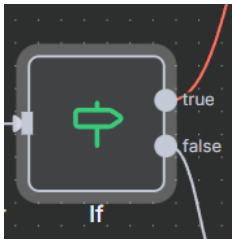


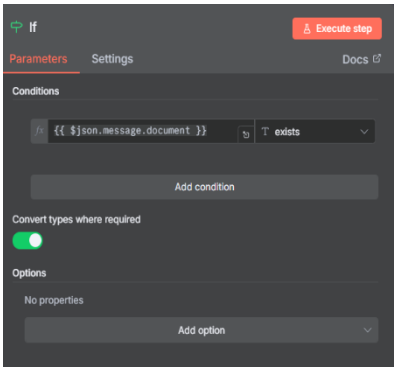
Este nodo es el que recibe lo que el usuario le envía al bot a travez del chat de Telegram y luego lo envía al nodo If.

En este nodo agregamos nuestra cuenta, la cual cuenta con el token que nos dio el botfather en telegram y dejamos la URL que está por defecto.

Y activamos la opción de descargar imágenes/archivos

2. If node → Qdrant Vector Store node/ Chat Memory Manager



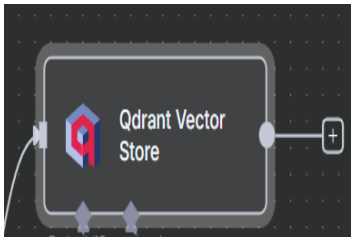


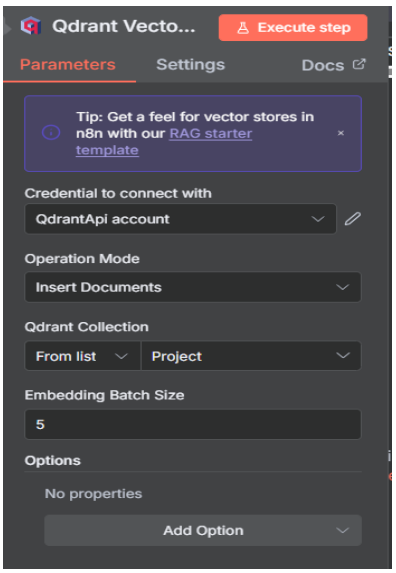
Este nodo permite que el flujo siga por un camino o por otro dependiendo de la condición que se le dé.

En este caso usamos la condición "{{ $json.message.document }} exist" la cual se traduce en que si la entrada que le llega al If desde el tigger es un documento entonces el flujo irá por la rama True, sino irá por la rama False. Además activamos la opción Convert types where required para que el flujo funcione correctamente.

- Rama True:

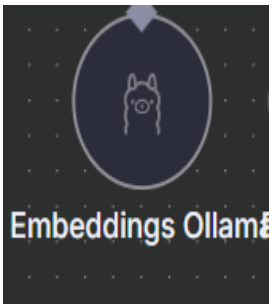
    1. Qdrant Vector Store

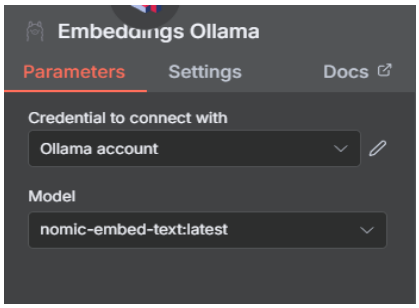




Este nodo está configurado como un insertor, el cúal lo que hace es recibir el documento y guardarlo en la colección de Qdrant que el usuario elija. En los parametros del nodo debemos añadir una cuenta de qdrant en la cual nos pedirá un api key y una URL, en la URL debemos poner la dirección del contenedor de qdrant que esta en docker; en nuestro caso: http://qdrant:6333 y el api key lo dejamos vacío ya que lo estamos haciendo de modo local. En Operation mode lo dejamos en Inssert Documents, en Qdrant Collection lo dejamos en From list y elejimos la colección de qdrant que utilizaremos, en nuestro caso usamos la colección "Project", En embenddinh Batch Size lo dejamos en un número de 3 a 5 que són el número de textos que se envían juntos al modelo para generar embeddings en una sola operación. Un valor mayor acelera el procesamiento, pero consume más memoria. Un valor menor es más estable, pero más lento.

Este nodo tiene 2 terminales:

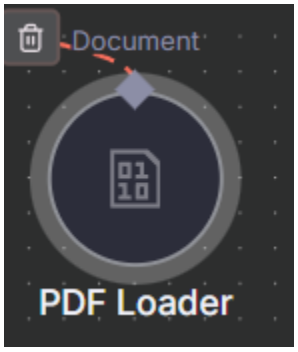


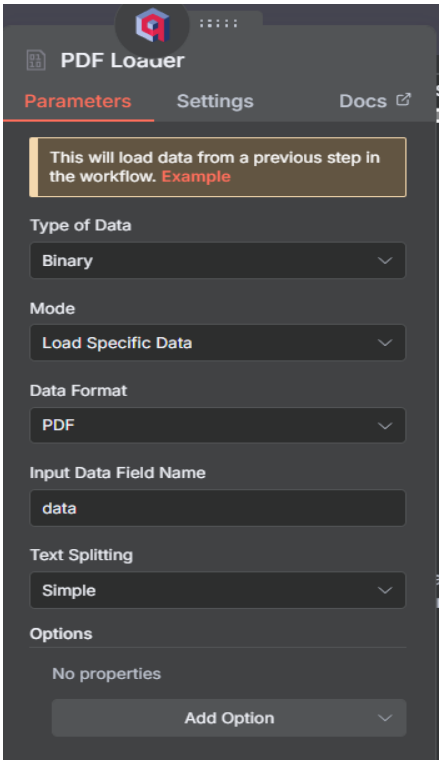


- Embenddings

  Este nódo contiene el modelo de embendding que usará el Vector Store para guardar la informacion en qdrant. En este caso usamos el modelo que descargamos “nomic-embed-text-lastest”

- Document





  Este nodo nosotros le cambiamos el nombre a PDF loader y lo que hace es tomar el texto puro que le llega y lo tansforma en un documento legible para qdrant, en un formato estandar de documento que incluye el contenido del texto, los metadatos, y la estructura que requiere el vector store, para que sea compatible con los embenddings y qdrant.

  Está configurado con:

  Type of data: Binary

  Mode: Load Specific Data

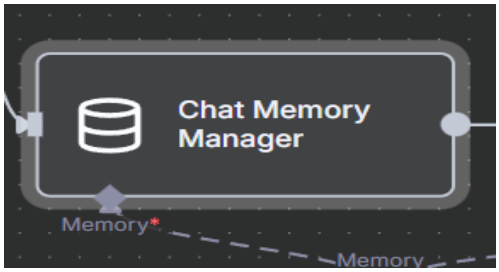
  Data Format: PDF

  Input Data Field Name: data

  Text Splitting: Simple

- Rama False:

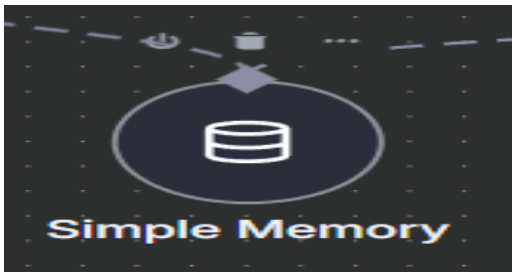
1. Chat Memory Manager → AI Agent



Este nodo se encarga de monitorear la memoria e historial de mensajes para llevar el hilo de la conversación y el agente tenga un buen contexto de la conversación y no sea un bot que simplemente responde a mensajes cómo si fueran la primera vez.

Este nodo va conectado a:

- Simple Memory



Esta es una memoria sencilla que almacena los mensajes de la conversación.

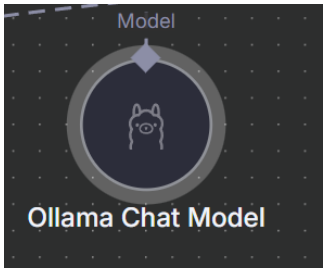
2. AI Agent → Tigger send a message de Telegram

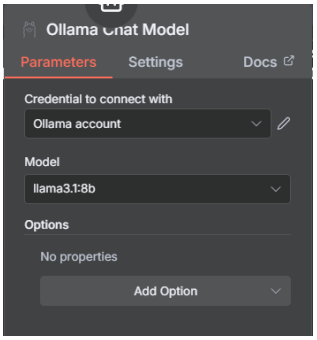
Este nodo es el agente que se encarga de procesar los mensajes que le envía el usuario y responde de acuerdo al contexto de la conversación, como en base a la información que haya en la colección de qdrant. El agente recibe el mensaje y contexto del Manager y luego procesa una respuesta, que luego va a la herramienta Answerer y revisa la collección con el nodo Qdrant Vcetor Store con una consulta en un embendding y luego obtiene una respuesta que devuelve el nodo de Qdrant, para luego ser procesada una respuesta con un modelo de chat que puede ser el mismo que se enviará al agente, cuya respuesta luego será enviada al agente y este ya procesará la respuesta final que será enviada al Tiiger send a message.

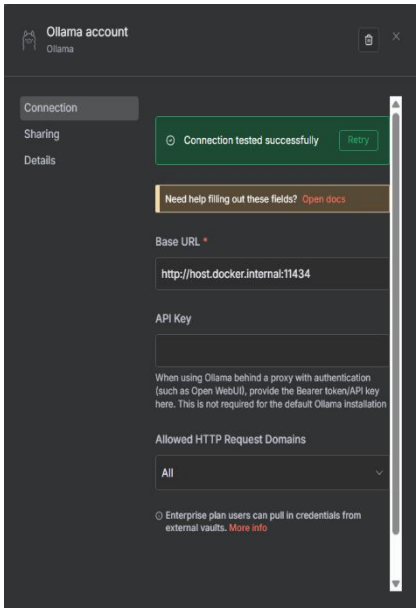
Este nodo está conectado a:

- Ollama chat Model

Este nodo contiene el Modelo de Chat que instalamos en el computador, así que ponemos nuestra cuenta de Ollama, en la cual si estamos ejecutando Ollama en Docker usamos la URL http//:localhost:5678, pero nosotros al estar corriendo Ollama de forma local en Windows usamos la URL http://host.docker.internal:11434 y dejamos el resto cómo está. Luego elejimos el modelo de chat que queramos usar en el caso de que tengamos más de uno. En nuestro caso usamos el modelo Llama 3.1:8b

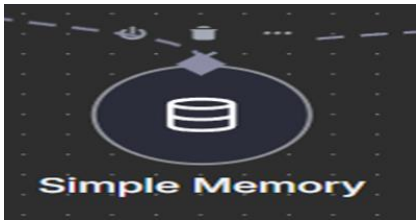






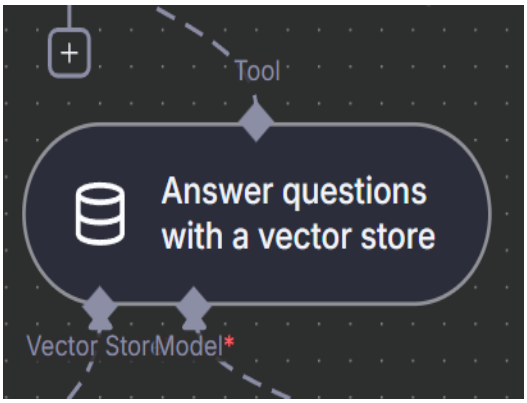
- Simple Memory

Esta memoria debe ser la misma que está conectada al manager para que compartan el contexto.



- Answer questions with a vectore store (tool)

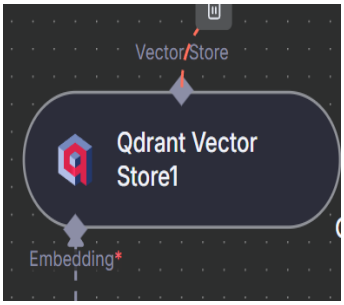
Esta herramienta es la que se encarga de buscar en la base de datos y luego generar una respuesta con lo encontrado y luego será enviada al egente para que la procese.

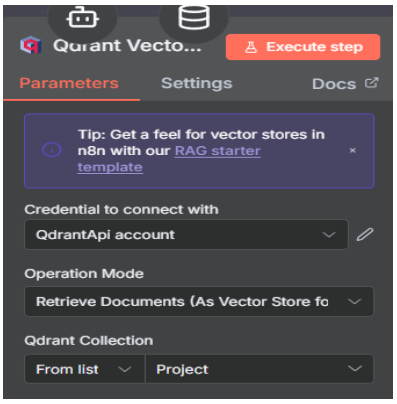


Esta herramienta está conectada a:

- ❖ Qdrant Vector Store1

Este nodo estará en modo Retrieve, que opera de manera contraria al Inser anterior, este nodo busca en Qdrant usando embenddings y luego envía lo encontrado al nodo Answer. Aquí ponemos la misma cuenta Qdrant y colección que pusimos en el nodo de Insert Documents.

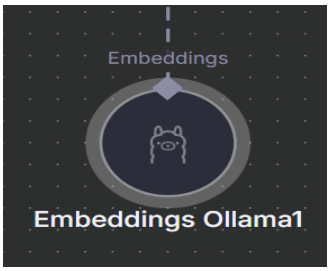


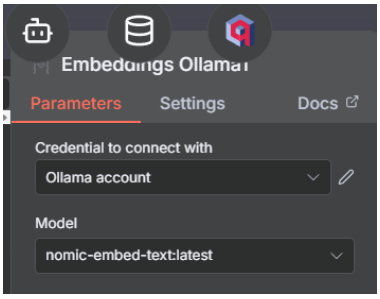


Este nodo está conectado a:

- ○ Embenddings Ollama

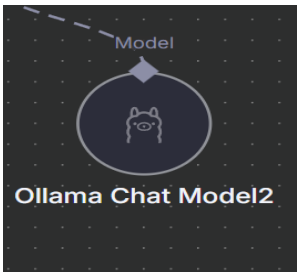
El mismo modelo de embendding usado
en el embendding del nodo Qdrant Vector Store que guarda los documentos usando la misma cuenta de Ollama.

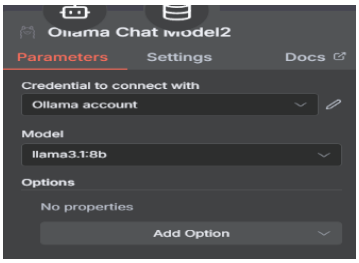




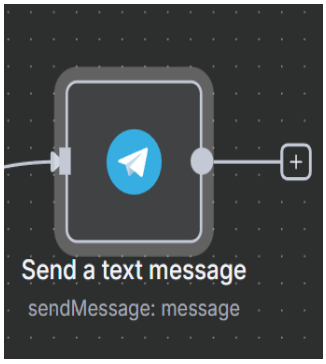
- ❖ Ollama Chat Model2

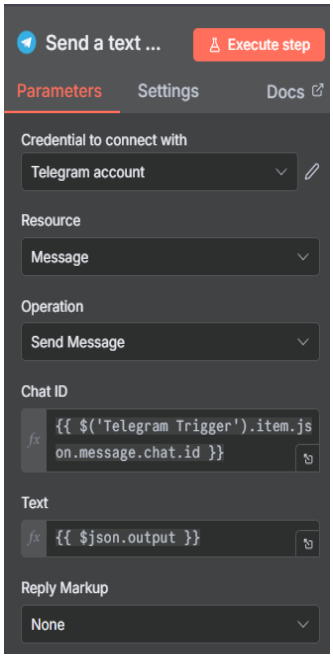
Un modelo de chat para procesar la respuesta que el Answer le enviará al agente, puede ser el mismo que el modelo del agente (lo recomendable).





3. Tigger send a message de Telegram
Este tigger es el que recibe la respuesta final del agente y la envía al bot de Telegram a travez del Webhook y este ya se lo muestra al usuario. En el nodo ponemos el Chat ID y el text de la salida del agente IA y dejamos la siguiente configuración:





Y con esto hemos terminado de configurar el flujo de trabajo de n8n y podremos usarlo desde el chat del bot de Telegram.