Seminario 2: Bot de analisis del sentimiento de mercados financieros

Alejandro Martínez Riquelme
Jordi Blasco Lozano
Interacción Persona Máquina
Universidad de Alicante

26 de octubre de 2025

Resumen

En un mundo financiero cada vez más dinámico y sobrecargado de información, tomar decisiones de inversión ágiles e informadas es un desafío constante. Este proyecto presenta el desarrollo de un asistente conversacional inteligente a través de un bot de Telegram, diseñado para realizar análisis de sentimiento de mercado para cualquier acción o ETF bajo demanda. El sistema integra múltiples tecnologías como n8n, Docker y Cloudflare Tunnel para crear una solución robusta y accesible que simplifica el acceso a inteligencia de mercado.

Índice

1 Introducción al Proyecto: Bot de Análisis de Sentimiento Bursátil

En un mundo financiero cada vez más dinámico y sobrecargado de información, tomar decisiones de inversión ágiles e informadas es un desafío constante. Los inversores, tanto novatos como experimentados, se enfrentan al reto de procesar grandes volúmenes de noticias, informes y opiniones para evaluar el sentimiento del mercado sobre un activo financiero. Este proceso no solo consume mucho tiempo, sino que también requiere un análisis complejo para extraer conclusiones prácticas.

Para abordar este problema, hemos desarrollado una solución innovadora: un **asistente conversacional inteligente a través de un bot de Telegram**, diseñado para realizar análisis de sentimiento de mercado para cualquier acción o ETF bajo demanda.

Nuestro proyecto nace de la necesidad de simplificar y automatizar el acceso a inteligencia de mercado. El objetivo principal es proporcionar a cualquier usuario una herramienta accesible desde su móvil que, con un simple comando, sea capaz de:

1. **Recopilar las noticias más recientes** y relevantes sobre un activo financiero específico.
2. **Utilizar un Modelo de Lenguaje Avanzado (LLM)** para analizar esta información y determinar el sentimiento general del mercado.
3. **Generar una recomendación de inversión clara y concisa** a medio plazo, categorizada en acciones como compra fuerte, compra, mantener, vender o venta fuerte.

Para lograrlo, hemos diseñado y construido una arquitectura robusta que integra múltiples tecnologías. Utilizando **n8n** como plataforma central de automatización para orquestar el flujo de trabajo, **Docker** para crear el entorno de ejecución, y un túnel de **Cloudflare** para exponer nuestro servicio local a Internet de forma segura y asi poder permitir la entrada de mensajes mediante nuestro bot de telegram.

Este proyecto no solo resuelve un problema real, sino que también sirve como una demostración práctica de cómo la combinación de la automatización de flujos de trabajo, la inteligencia artificial conversacional y la infraestructura nos permite crear potentes aplicaciones con recursos limitados. A continuación, explicaremos en detalle cada uno de los pasos que hemos seguido para dar vida a este bot, desde la configuración del entorno hasta el diseño del flujo de inteligencia.

2 Configuración del Túnel y Scripts de Inicio

Para que nuestro bot de Telegram pueda recibir y procesar mensajes en tiempo real, es fundamental exponer el servidor local de n8n a Internet, ya que Telegram requiere una URL pública y accesible para configurar el webhook del trigger. Sin esta exposición, el trigger de Telegram no podría funcionar, ya que n8n opera por defecto en localhost:5678, un puerto interno no visible desde fuera. Elegimos Cloudflare Tunnel (cloudflared) por su simplicidad y gratuidad para entornos de desarrollo, aunque sus URLs generadas no son estáticas y cambian en cada ejecución, lo que impide hardcodearlas directamente en la configuración de n8n.

El desafío principal radica en esta variabilidad: si intentáramos definir una URL fija en el nodo de Telegram o en el docker-compose.yml, el webhook fallaría al reiniciar el servicio. Por ello, optamos por un enfoque dinámico basado en variables de entorno (.env), que permite inyectar la URL generada automáticamente en el contenedor de n8n sin intervención manual cada vez. Este método asegura que el trigger de Telegram apunte siempre a la URL correcta, usando variables como WEBHOOK_URL, N8N_HOST y TUNNEL_URL para configurar el protocolo HTTPS y el host externo.

2.1 Paso 1: Instalación y Preparación de Cloudflare Tunnel

Primero, instalamos cloudflared en nuestro sistema operativo (en este caso, Linux o WSL para compatibilidad con Docker). Este herramienta crea un túnel seguro desde nuestro puerto local (5678) hacia una URL temporal en el dominio trycloudflare.com, sin necesidad de cuenta ni configuración compleja. El comando base para iniciarlo es `cloudflared tunnel --url http://localhost:5678`, que genera una

salida como “https://xxxx.trycloudflare.com” y la registra en un log para su captura posterior. Este túnel enruta el tráfico entrante de Telegram directamente a n8n, manteniendo la seguridad al no exponer puertos directamente en el firewall.

2.2 Paso 2: Creación del Script de Inicio (start-n8n.sh)

Para automatizar todo el proceso, creamos el script `start-n8n.sh`, que sigue un orden específico y secuencial para garantizar la integración sin errores. El script comienza deteniendo cualquier instancia previa de `cloudflared` o Docker para evitar conflictos: `pkill cloudflared 2>/dev/null` y `docker-compose down 2>/dev/null`. Luego, lanza `cloudflared` en segundo plano con redirección de salida a un archivo `tunnel.log`: `cloudflared tunnel --url http://localhost:5678 >tunnel.log 2>&1 &`, capturando el PID para control posterior.

A continuación, el script espera hasta 30 segundos para que se genere la URL, monitoreando el log con un bucle: `grep -oP 'https://[a-zA-Z0-9-]+.trycloudflare.com' tunnel.log | head -1`. Una vez extraída la URL (por ejemplo, https://offline-argued-drops.trycloudflare.com), la valida y la almacena en una variable TUNNEL_URL. Si no se obtiene en el tiempo límite, el script falla con un error y mata el proceso de `cloudflared`. Este paso es crítico porque sin la URL dinámica, el webhook de Telegram no se configuraría correctamente en n8n.

2.3 Paso 3: Actualización de Variables de Entorno (.env)

Con la URL en mano, el script actualiza el archivo .env, que sirve como fuente de variables para docker-compose.yml. Genera o sobrescribe el .env con contenido como:

- WEBHOOK_URL=${TUNNEL_URL}
- N8N_HOST=${TUNNEL_URL} (adaptado para HTTPS)
- TUNNEL_URL=${TUNNEL_URL}

Esto inyecta la URL en el entorno de n8n, permitiendo que el nodo de Telegram use automáticamente WEBHOOK_URL como endpoint público. El script muestra el contenido del .env para verificación antes de proceder. De esta forma, evitamos editar manualmente el .env cada reinicio, haciendo el despliegue reproducible con un solo comando: `./start-n8n.sh`.

2.4 Paso 4: Lanzamiento del Contenedor de n8n

Finalmente, con el .env actualizado, el script ejecuta `docker-compose up -d`, iniciando el contenedor de n8n con las variables cargadas. El `docker-compose.yml` define el servicio n8n con puertos expuestos en 5678, volúmenes para persistencia de datos (`/home/node/.n8n`), y variables como `N8N_PROTOCOL=https` y `N8N_EDITOR_BASE_URL=${TUNNEL_URL}` para alinear la interfaz web y los webhooks. Una vez arriba, n8n configura internamente el webhook de Telegram apuntando a la URL del túnel, completando el ciclo de exposición. Para detener todo, usamos un script complementario `stop-n8n.sh` que hace `down` en Docker y mata `cloudflared`, limpiando logs opcionalmente.

Esta configuración no solo resuelve la exposición dinámica, sino que también mantiene el proyecto portable y escalable, ideal para demostraciones como esta. En la siguiente sección, detallaremos la creación del bot en Telegram y su integración en el workflow de n8n.

3 Creación del Bot de Telegram e Integración del Trigger

Una vez configurado el túnel para exponer n8n a Internet, el siguiente paso esencial es crear el bot en Telegram y conectarlo al workflow mediante el nodo de trigger de Telegram, que actúa como punto de entrada para recibir mensajes del usuario. Este trigger se basa en la API de Telegram Bot, que requiere un token de autenticación para configurar un webhook que envíe actualizaciones de mensajes a nuestra URL pública (la generada por cloudflared). Sin esta integración, el bot no podría interactuar con el flujo de n8n, ya que Telegram solo notifica a endpoints accesibles externamente.[1]

3.1 Paso 1: Creación del Bot con BotFather

Para iniciar, abrimos Telegram y buscamos el usuario oficial @BotFather, que es la herramienta de Telegram para gestionar bots. Enviamos el comando `/newbot` seguido del nombre del bot (por ejemplo, "Stock Market Analytics Bot") y un username único terminando en "bot" (como `@stock_market_analytics_bot`), que debe ser público y accesible para todos los usuarios. BotFather responde confirmando la creación y proporcionando un **token de API** en formato `123456789:ABCDEF...`, que actúa como clave secreta para autenticar todas las peticiones del bot. Este token no debe compartirse públicamente, ya que otorga control total sobre el bot, incluyendo la capacidad de enviar y recibir mensajes.

3.2 Paso 2: Configuración del Nodo de Trigger en n8n

Con el token en mano, accedemos a la interfaz de n8n (vía la URL del túnel) y creamos un nuevo workflow, arrastrando el nodo **Telegram Trigger** al canvas. En las credenciales del nodo, seleccionamos "Create New" para la autenticación de Telegram, ingresando el token de API obtenido de BotFather y configurando el tipo como "Bot Token Authentication". El nodo también requiere la URL del webhook, que se resuelve automáticamente usando la variable de entorno WEBHOOK_URL del .env (injected vía docker-compose), apuntando a `https://[túnel].trycloudflare.com/webhook/[workflow-id]`, donde n8n maneja la ruta interna para actualizaciones. Al activar el workflow, n8n registra el webhook con la API de Telegram, confirmando que recibirá notificaciones push para cada mensaje enviado al bot. Esto asegura que el trigger se active solo para nuestro bot, filtrando interacciones irrelevantes por defecto.[1]

3.3 Paso 3: Estructura del Mensaje Entrante y Preparación para el Flujo

Cuando un usuario envía un mensaje al bot, como `/analysis sp500`, el trigger de Telegram recibe un JSON estructurado con la actualización completa del mensaje, que incluye metadatos del usuario y el chat para contextualizar la interacción. La estructura típica es un array de objetos con campos como `update_id` (identificador único de la actualización), `message` (detalles del mensaje), que contiene `message_id`, `from` (información del emisor con `id` numérico), `chat` (detalles del chat con el mismo `id`), `date` (timestamp), `text` (contenido del mensaje, e.g., "/analysis sp500") y `entities` (anotaciones como comandos de bot). Por ejemplo, un mensaje de prueba genera algo como:


```
[
  {
    "update_id": 177484711,
    "message": {
      "message_id": 20,
      "from": {
        "id": 1065676350,
        "is_bot": false,
        "first_name": "Jordi",
        "last_name": "Blasco Lozano",
        "username": "jbloz",
        "language_code": "es"
      },
      "chat": {
        "id": 1065676350,
        "first_name": "Jordi",
        "last_name": "Blasco Lozano",
        "username": "jbloz",
        "type": "private"
      },
      "date": 1761410615,
      "text": "/analysis sp500",
      "entities": [
        {
          "offset": 0,
          "length": 9,
```

```
          "type": "bot_command"
        }
      ]
    }
  }
]

```

Este formato detallado es útil para tracking, pero para nuestro flujo de análisis de sentimiento, necesitamos simplificarlo extrayendo solo los elementos esenciales: el texto después del comando (e.g., "sp500") y el ID del usuario (e.g., 1065676350) para respuestas posteriores. Por lo tanto, transformamos el JSON entrante en una estructura minimalista como:

```
[
  {
    "text": "sp500",
    "id": 1065676350
  }
]

```

Esta reformateación se logra en los nodos subsiguientes (filtro y código JavaScript), eliminando ruido como nombres, timestamps y entidades, para pasar datos limpios al resto del workflow sin sobrecargar el procesamiento. De esta manera, el trigger inicializa el flujo de forma eficiente, preparando el terreno para la validación y extracción de comandos específicos como `/analysis`.

En la próxima sección, detallaremos cómo filtramos mensajes irrelevantes y refinamos el texto extraído mediante nodos de código.

4 Transformación y filtrado de los datos de entrada: Nodos Filter y JavaScript en n8n

Una vez que el trigger de Telegram recibe los mensajes enviados al bot, el siguiente paso en el workflow de n8n es limpiar y transformar los datos para quedarnos solo con la información relevante para nuestro análisis bursátil. El objetivo aquí es filtrar solo los mensajes que contengan el comando `/analysis` y extraer el texto posterior al comando (el ticker o nombre del activo) junto al identificador de usuario para personalizar las respuestas en el resto del flujo.

4.1 Paso 1: Nodo de Filter

El nodo **Filter** funciona como una puerta de acceso, permitiendo únicamente los mensajes que cumplen una condición clave: que el mensaje incluya la palabra `/analysis`. Así, si un usuario escribe cualquier otro texto o comando, el flujo lo descartará automáticamente y no avanzará a las siguientes etapas. Esta lógica reduce ruido y asegura que solo procesamos peticiones válidas y estructuradas para el análisis del sentimiento de mercado.

Configuración:

- Campo a filtrar: `message.text` (dentro del JSON recibido).
- Condición: contiene `/analysis`.
- Acción: si pasa el filtro, el mensaje sigue el flujo; si no, se descarta.

4.2 Paso 2: Nodo Code (JavaScript)

El nodo **Code** o **Code in JavaScript** es donde transformamos el mensaje filtrado en un formato sencillo y funcional para el resto del workflow. El bloque de código que has incluido cumple tres funciones principales:

- Toma el texto completo del mensaje (`fullText`) y el identificador del usuario (`ID`).
- Localiza el comando `/analysis` y extrae únicamente el texto que viene detrás (lo que el usuario quiere analizar: un ticker, índice, etc.).
- Devuelve un objeto JSON minimalista con el formato exactamente necesario para la API siguiente: el `text` limpio y el `id` para enviar la respuesta al usuario correcto.

```
const fullText = $input.item.json.message.text;
const ID = $input.item.json.message.from.id;
const command = "/analysis ";

// Extrae todo lo que viene después de /analysis
const textAfterCommand = fullText.includes(command)
? fullText.substring(fullText.indexOf(command) + command.length).trim()
: fullText;

return {
json: {
    text: textAfterCommand,
    id: ID
}
};
```

Gracias a esto, transformamos un mensaje de entrada complejo, como el que hemos visto en el paso anterior en una salida simple y limpia como esta:

```
[
  {
    "text": "sp500",
    "id": 1065676350
  }
]
```

Esto prepara el mensaje para ser enviado a la API de análisis de sentimiento o cualquier otro módulo de procesamiento posterior, facilitando al máximo la integración y manteniendo limpio el flujo de trabajo en n8n.

5 HTTP Request para buscar noticias con Perplexity API

Una vez que hemos procesado el mensaje del usuario para obtener únicamente el texto relevante (el ticker o nombre del activo financiero), el siguiente paso del workflow consiste en consultar la API de Perplexity para recopilar las noticias más recientes sobre ese activo. Esta búsqueda es fundamental para que el LLM realice el análisis de sentimiento con información actualizada y relevante.

5.1 Perplexity, subscripción y API

Perplexity ofrece varias maneras de acceder a su API Pro, incluyendo un periodo de prueba de 12 meses si asocias una cuenta de PayPal, así como promociones puntuales para ciertas cuentas o estudiantes. En nuestro caso, disponemos de una suscripción Pro, que nos da acceso prioritario tanto a modelos avanzados como a la propia API. Esta suscripción incluye 5 € de crédito API cada mes, suficiente para mantener proyectos ligeros y pruebas recurrentes.

5.2 Acceso a la API de búsqueda

Con la suscripción Pro, tenemos acceso directo a la API, incluida la función más potente: la **API de búsqueda avanzada**. La documentación oficial de Perplexity proporciona un ejemplo de integración

rápida mediante cURL (que podemos copiar y pegar al módulo de HTTP Request en n8n), autenticando cada llamada con nuestra clave personal. Allí mismo se detallan parámetros de uso como el límite de resultados o el máximo de tokens por noticia.

5.3 Ejemplo práctico de la integración en n8n

Creamos un módulo HTTP Request justo después del bloque de JavaScript, usando estos ajustes:

- **URL:** `https://api.perplexity.ai/search`
- **Método:** POST
- **Autenticación:** Bearer Token (clave API de Perplexity)
- **Cuerpo (JSON):**

```
{
  "query": "Last news about the stock of {{ $json.text }}",
  "max_results": 8,
  "max_tokens_per_page": 2048
}
```

Aquí, `{{ $json.text }}` toma el ticker del paso anterior para personalizar la búsqueda.

El input a este módulo será el JSON minimalista del paso previo (por ejemplo, `{"text": "sp500", "id": 1065676350}`), asegurando así que la consulta esté siempre alineada con el comando que ha solicitado el usuario.

Con esto, cada vez que el bot lo requiera, buscará siempre las 8 noticias más relevantes (máximo 2048 tokens cada una), proporcionando a nuestro agente LLM los datos actualizados y preparados para la toma de decisiones de inversión y el análisis de sentimiento.