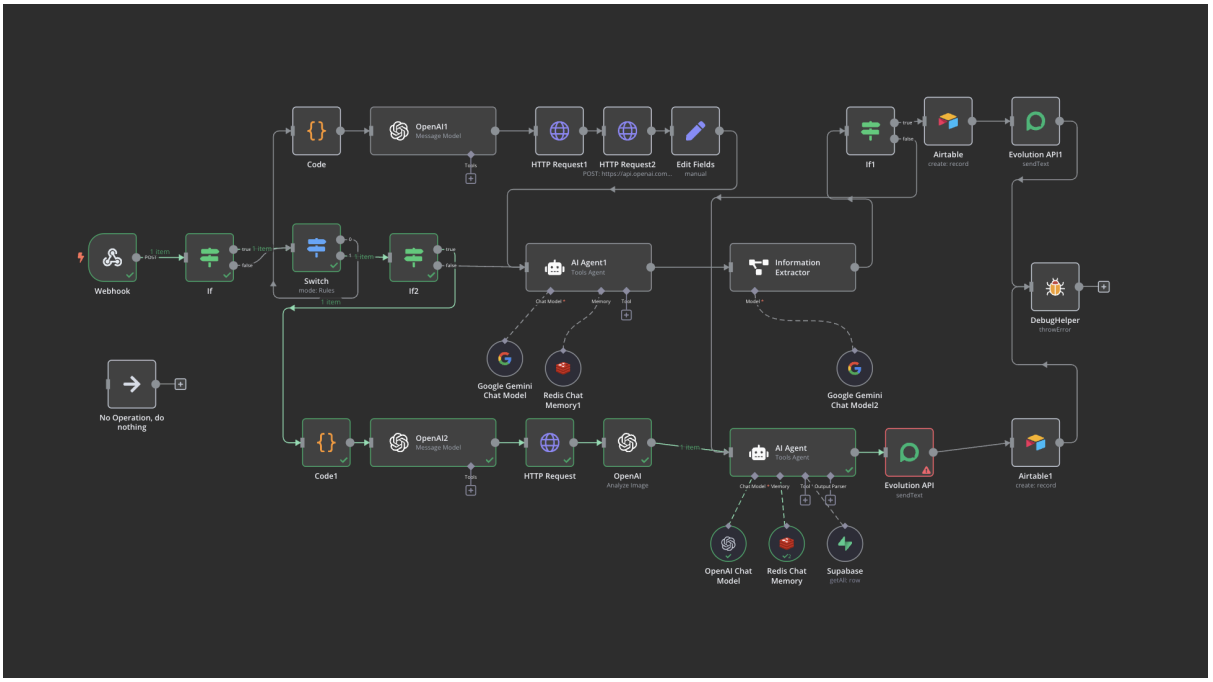
Resumen Ejecutivo y Objetivos del Proyecto

- **Nombre del Proyecto:** Sofia atención al cliente bot.
- **Propósito:** Automatizar la interacción con clientes potenciales a través de WhatsApp, captar información de contacto, proporcionar atención al cliente y optimizar el embudo de ventas.
- **Público Objetivo:** Clientes potenciales que interactúan con campañas publicitarias en Meta.
- **Canal de Comunicación:** WhatsApp.
- **Funciones Principales:**
    - Responder preguntas frecuentes.
    - Agendar citas.
    - Facilitar el crecimiento del CRM.



(Flujo del bot n8n para atención al cliente-2024)

Arquitectura Tecnológica y Componentes

Este fragmento describe las herramientas y tecnologías que se utilizarán para construir y operar el chatbot.

- **Infraestructura:** Google Cloud (instancia e2-standard-2, Debian 11, 100 GB SSD, reglas de firewall HTTP/HTTPS).
    - *Detalle:* Se utilizará una máquina virtual en Google Cloud con las siguientes especificaciones:
        - Tipo de máquina: e2-standard-2 (2 vCPUs, 8 GB de RAM).
        - Sistema Operativo: Debian 11.
        - Disco: 100 GB SSD.
        - Red: Se habilitarán las reglas de firewall para permitir el tráfico HTTP y HTTPS, lo cual es esencial para el acceso web seguro.

- **Plataforma de Despliegue:** EasyPanel.
    - *Detalle:* EasyPanel simplificará la gestión y el despliegue de la aplicación en la instancia de Google Cloud.

Guía de Instalación de Easy Panel y n8n en Google Cloud

Esta guía detalla el proceso para instalar y configurar **Easypanel** y **n8n** en una instancia de Google Cloud con Ubuntu. También incluye la resolución de errores comunes.

---

1. Configuración inicial de la instancia

1.1. Crear una instancia en Google Cloud

- Ve a la consola de Google Cloud.
- Crea una instancia de VM con las siguientes especificaciones mínimas:
    - **Sistema operativo:** Ubuntu 20.04
    - **Memoria:** 2 GB (recomendado: 4 GB)
    - **CPU:** 1 vCPU
- Habilita el acceso al puerto 80, 443 y cualquier otro necesario en las **reglas de firewall**.

---

2. Actualizar el sistema y preparar herramientas básicas

Ejecuta los siguientes comandos en tu terminal para actualizar el sistema:

bash
Copiar código

```
sudo apt update && sudo apt upgrade -y
sudo apt install -y curl wget gnupg unzip nano apt-transport-https
ca-certificates software-properties-common
```

Errores comunes y solución

- Si ves errores sobre paquetes faltantes, asegúrate de que el sistema tenga acceso a internet y repite el comando anterior.

---

3. Instalar Docker

3.1. Agregar el repositorio de Docker

bash
Copiar código

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg
--dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
echo "deb [arch=amd64
signed-by=/usr/share/keyrings/docker-archive-keyring.gpg]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
| sudo tee /etc/apt/sources.list.d/docker.list
```

3.2. Instalar Docker

bash
Copiar código

```
sudo apt update
sudo apt install -y docker-ce docker-ce-cli containerd.io
```

3.3. Verificar instalación

bash
Copiar código

```
sudo systemctl start docker
sudo systemctl enable docker
docker --version
```

---

4. Instalar Docker Compose

4.1. Descargar Docker Compose

bash
Copiar código
```
sudo curl -L
"https://github.com/docker/compose/releases/latest/download/docker-c
ompose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

4.2. Dar permisos de ejecución

bash
Copiar código
```
sudo chmod +x /usr/local/bin/docker-compose
```

4.3. Verificar la instalación

bash
Copiar código
```
docker-compose --version
```

---

5. Instalar Easypanel

5.1. Descargar e instalar Easypanel

bash
Copiar código
```
sudo -i
curl -fsSL https://get.easypanel.io | bash
```

Errores comunes y solución

1. **Error: "you must be root to execute this script"**

Solución: Ejecuta el comando como `root` usando:
bash
Copiar código
```
sudo -i
curl -fsSL https://get.easypanel.io | bash
```

    -
2. **Error: "something is already running on port 80"**

Verifica qué proceso está usando el puerto 80:
bash
Copiar código
```
sudo lsof -i :80
```

-

Si `google_osconfig_agent` o `google_guest_agent` están ocupando el puerto, deténlos:
bash
Copiar código
```
sudo systemctl stop google-osconfig-agent
sudo systemctl stop google-guest-agent
```

5.2. Acceder a Easypanel

Abre tu navegador y accede a:

plaintext
Copiar código
```
http://<IP-de-tu-servidor>
```

Si el puerto fue cambiado durante la instalación (por ejemplo, `8080`):

plaintext
Copiar código
```
http://<IP-de-tu-servidor>:8080
```

---

6. Configurar Easypanel y agregar n8n

6.1. Crear un nuevo proyecto

1. Abre Easypanel y haz clic en **Create Project** o **+ New**.
2. Asigna un nombre al proyecto, como `n8n-chatbots`.

6.2. Agregar un servicio para n8n

1. Dentro del proyecto, haz clic en **+ Service**.
2. Configura el servicio con los siguientes parámetros:
   - **Name (Nombre):** `n8n`
   - **Docker Image:** `n8nio/n8n`

- **Ports (Puertos):** `5678:5678`
- **Environment Variables:**
  - `N8N_BASIC_AUTH_ACTIVE=true`
  - `N8N_BASIC_AUTH_USER=admin`
  - `N8N_BASIC_AUTH_PASSWORD=tu_contraseña_segura`
  - (Opcional) `N8N_USER_FOLDER=/home/node/.n8n`
- **Volumes:**
  - **Ruta local:** `/home/node/.n8n`
  - **Ruta del contenedor:** `/home/node/.n8n`

3. Haz clic en **Deploy** o **Save** para iniciar el servicio.

6.3. Acceder a n8n

1. Una vez desplegado, abre tu navegador y accede a:

plaintext
Copiar código
`http://<IP-de-tu-servidor>:5678`

2. Ingresa las credenciales configuradas en las variables de entorno (`admin` y tu contraseña).

---

7. Configurar el Firewall en Google Cloud

Si no puedes acceder a Easypanel o n8n, abre los puertos necesarios en el firewall:

7.1. Crear regla de firewall

1. Ve a **VPC Network > Firewall** en la consola de Google Cloud.
2. Crea una nueva regla con los siguientes valores:
   - **Nombre:** `allow-easypanel-n8n`
   - **Red:** `default`
   - **Rango de IP de origen:** `0.0.0.0/0`
   - **Protocolo y puertos:**
     - TCP: `80, 443, 5678`

---

8. Solución de problemas comunes

8.1. Puerto ocupado

- Usa `sudo lsof -i :80` para identificar el proceso.
- Detén los servicios conflictivos con `sudo systemctl stop <servicio>`.

8.2. Acceso denegado

- Verifica las reglas de firewall en Google Cloud.
- Confirma que el puerto está abierto con:
  bash
  Copiar código
  `sudo lsof -i :<puerto>`

- **Orquestación y Automatización:** n8n.
  - *Detalle:* n8n se utilizará para automatizar flujos de trabajo, incluyendo la lógica del chatbot, la integración con otras APIs y la gestión de datos.

1. Nodo Webhook (Evento de entrada desde WhatsApp)

Lógica:

- Se activa cuando se recibe un mensaje entrante desde un número de WhatsApp conectado.

Datos Recibidos:

- Información del contacto: nombre, número de teléfono, identificador único.
- El mensaje recibido: contenido, tipo de contenido, metadatos como fecha/hora.
- Información sobre la conversación: estado, etiquetas, prioridad, asignación a un agente.

Procesamiento:

- Verificar que el mensaje contiene la información mínima (contenido y número del remitente).
- Identificar el tipo de contenido del mensaje (texto, archivo, sticker, etc.).
- Almacenar información relevante para su posterior procesamiento.

---

2. Nodo If (Evaluación de si el mensaje es entrante)

Condición Evaluada:

Verifica si el tipo de mensaje es **entrante** (incoming):
plaintext

Copiar código
```
{{ $json.body.message_type }} == "incoming"
```

-

Acciones:

- **True**: Si el mensaje es entrante, continúa con el procesamiento del mensaje.
- **False**: Si el mensaje no es entrante (por ejemplo, es un mensaje saliente), el flujo no avanza por este camino.

---

3. Nodo Switch (Decisión de tipo de archivo adjunto)

Lógica:

- Este nodo evalúa el tipo de archivo adjunto (por ejemplo, si es un audio o un texto) y determina el flujo adecuado para el mensaje.

Reglas de Enrutamiento:

- **Audio**: Si el archivo es un **audio**, el flujo va por una ruta de audio.
- **Texto**: Si el archivo es **texto**, el flujo va por una ruta de texto.

Formato de Regla:

Audio:
plaintext
Copiar código
```
{{ $('Webhook').item.json.body.attachments[0].file_type }} ==
"audio"
```

-

Texto:
plaintext
Copiar código
```
{{ $json.body.content_type }} == "text"
```

-

---

4. Nodo If1 (Evaluación de datos para agendar cita)

Condición Evaluada:

Verifica si existe la clave `datos_agendar_cita` en los datos extraídos.
plaintext
Copiar código

```
{{ $json.output.datos_agendar_cita }} exists
```

-

Acciones:

- **True**: Si los datos de agendar cita están presentes, el flujo continúa para agendar la cita en Airtable.
- **False**: Si los datos no están presentes (es una consulta normal), el flujo sigue por el camino False y pasa al siguiente nodo (Evolution API).

---

5. Nodo Airtable (Agendar Cita)

Lógica:

- Si los datos para agendar una cita están presentes, se almacenan en Airtable.

Datos Enviados a Airtable:

json

Copiar código

```
{
  "fields": {
    "Nombre": "Juan Pérez",
    "Fecha": "2024-12-25",
    "Hora": "10:00",
    "Estado": "Agendada"
  }
}
```

---

6. Nodo Evolution API (Consulta Normal)

Lógica:

- Si el mensaje es una consulta normal, se envía a la **Evolution API** para procesar la consulta.

Datos Enviados a Evolution API:

json

Copiar código

```
{

  "query": "Tengo una duda sobre mi factura"

}
```

Acciones:

- La **Evolution API** procesa el mensaje y genera una respuesta basada en el contenido del mensaje.
- La respuesta de la API se envía de vuelta al cliente.

---

7. Nodo Code (Generación de URL del Archivo)

Lógica:

- Este nodo genera una URL del archivo (por ejemplo, para imágenes o audios) a partir de la información recibida en el webhook.

Código de Ejemplo:

javascript

Copiar código

```
const generateImageUrl = (data) => {

    const data_url = data.attachments[0].data_url;

    const base_url =
"https://chatbots-chatwoot.ed3w2m.easypanel.host/rails/active_storag
e/blobs/redirect/";

    const final_url = base_url + data_url.split("redirect/").pop();
```

```
    return final_url;
};

const jsonData = {
    "webhookUrl":
"https://chatbots-n8n.ed3w2m.easypanel.host/webhook-test/test",
    "executionMode": "test",
    "attachments": [
        {
            "file_name": "image_1.oga",
            "data_url": " "
        }
    ]
};

const imageUrl = generateImageUrl(jsonData);

return [
  {
    json: {
      imageUrl: imageUrl
    }
  }
];
```

---

8. Nodo Information Extractor (Extracción de Datos Específicos)

Lógica:

- Este nodo extrae la información relevante del mensaje, como la fecha, hora y nombre si se trata de una cita, o el contenido completo del mensaje si es una consulta normal.

Datos Extraídos:

json

Copiar código

```
{
  "datos_agendar_cita": {
    "fecha": "2024-12-25",
    "hora": "10:00",
    "nombre": "Juan Pérez"
  },
  "datos_consulta_normal": {
    "mensaje": "Tengo una duda sobre mi factura"
  }
}
```

---

9. Nodo If2 (Verificación de Datos de Agendar Cita)

Condición Evaluada:

Verifica si los datos de la cita están presentes.
plaintext
Copiar código

```
{{ $json.output.datos_agendar_cita }} exists
```

Acciones:

- **True**: Si los datos de la cita están presentes, el flujo sigue para procesar el agendamiento de la cita en Airtable.
- **False**: Si no hay datos de cita, el flujo se dirige a procesar la consulta normal a través de la Evolution API.

---

10. Nodo Final (Respuesta al Cliente)

Lógica:

- Si el mensaje fue una consulta, la **Evolution API** procesa la consulta y devuelve una respuesta.
- Si fue una cita, se almacena la cita en Airtable y se confirma el agendamiento con una respuesta al cliente.

Resumen General del Flujo:

1. **Webhook** recibe el mensaje entrante desde WhatsApp.
2. **Nodo If** valida si el mensaje es entrante.
3. **Switch** decide qué tipo de contenido está adjunto (audio, texto, etc.).
4. **Nodo If1** verifica si el mensaje es para agendar una cita.
5. Si es **agendar cita**, los datos se almacenan en **Airtable**.
6. Si es **consulta normal**, el mensaje se envía a **Evolution API** para obtener una respuesta.
7. El flujo regresa al cliente con la respuesta adecuada.

(Imagen de referencia página.1 flujo n8n 2024)

- **Plataforma de Chat (Interfaz de Usuario):** Chatwoot.
    - *Detalle:* Chatwoot actuará como la interfaz principal para la interacción con los usuarios, gestionando las conversaciones y proporcionando una interfaz para los agentes (si se requiere interacción humana).

- **Base de Datos Principal (Datos Transaccionales):** PostgreSQL.
    - *Detalle:* PostgreSQL almacenará los datos principales de la aplicación, como la información de las conversaciones, configuraciones, etc.

- **Base de Datos para Recuperación Aumentada Generativa (RAG):** Supabase.

- *Detalle:* Supabase se utilizará para implementar capacidades de RAG, lo que permitirá al chatbot acceder y utilizar información externa para mejorar sus respuestas.

Pasos para crear la tabla y función en Supabase:

1. **Accede al SQL Editor de Supabase**:
   - Inicia sesión en tu cuenta de **Supabase**.
   - En el proyecto que estás utilizando, ve a la sección de **SQL Editor**.
2. **Habilitar la extensión `pgvector`**:
   - En el editor, copia y pega el siguiente del link:

Framework:https://supabase.com/docs/guides/ai/langchain?database-method=sql

Instrucciones detalladas de las credenciales:
https://www.youtube.com/watch?v=zoWgOUQQLuk&t=30s

3. **Ejecutar los scripts**:
   - Después de pegar cada uno de los fragmentos de código en el editor de SQL, haz clic en **Run** para ejecutar los scripts.
   - Supabase procesará los comandos y debería indicarte si los ejecutó correctamente.

- **CRM (Gestión de Relaciones con el Cliente):** Airtable.
  - *Detalle:* Airtable se utilizará para gestionar la información de los clientes, incluyendo datos de contacto, historial de interacciones, etc.

Instrucciones para las credenciales: https://airtable.com/create/tokens

- **Almacenamiento Adicional (Datos Complementarios):** Hojas de Cálculo de Google.
  - *Detalle:* Se utilizarán Hojas de Cálculo de Google para almacenar información adicional que pueda ser necesaria para el chatbot.

Instrucciones para las credenciales: https://cloud.google.com/apis?hl=es-419

- **API de WhatsApp (Conexión con WhatsApp):** Evolution API.
  - *Detalle:* Evolution API permitirá la integración con WhatsApp, gestionando el envío y la recepción de mensajes.

Instrucciones para las credenciales:
https://www.youtube.com/watch?v=y9imNPSMghQ&t=1s

- **Procesamiento de Lenguaje Natural (NLP):** GPT-4.0 Mini.
    - *Detalle:* GPT-4.0 Mini se utilizará para comprender y generar lenguaje natural, permitiendo al chatbot mantener conversaciones más fluidas y naturales.

Instrucciones para las credenciales:

- **Caché (Almacenamiento Temporal):** Redis.
    - *Detalle:* Redis se utilizará para almacenar datos en caché, lo que mejorará el rendimiento del chatbot al reducir la necesidad de acceder constantemente a la base de datos principal.

- **Contenedores:** Docker.
    - *Detalle:* Docker se utilizará para contenerizar las diferentes partes de la aplicación, lo que facilitará el despliegue y la gestión.

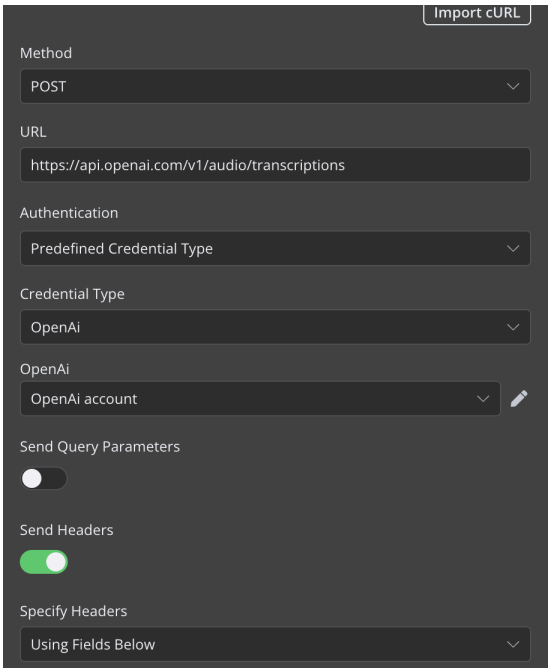
- **Lenguaje de Programación:** JavaScript.
    - *Detalle:* JavaScript será el lenguaje principal de desarrollo, utilizado para la lógica del chatbot, las integraciones con APIs y la interfaz de usuario.

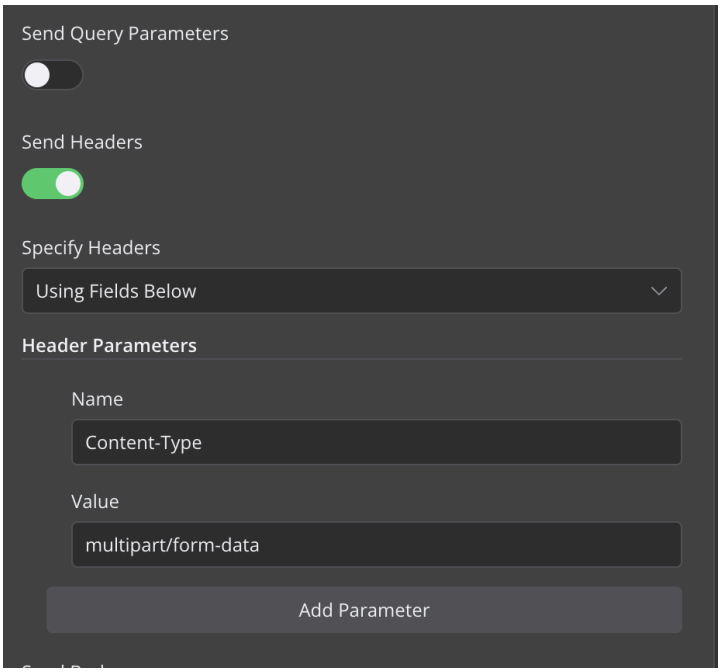
- **APIs Adicionales:**
    - API de Google Gemini: Podría utilizarse para complementar las capacidades de NLP.
    - API de Google Calendar: Para la funcionalidad de agendar citas.
    - API de Whisper (OpenAI): Podría utilizarse para transcripción de audio (si se implementa esta funcionalidad).

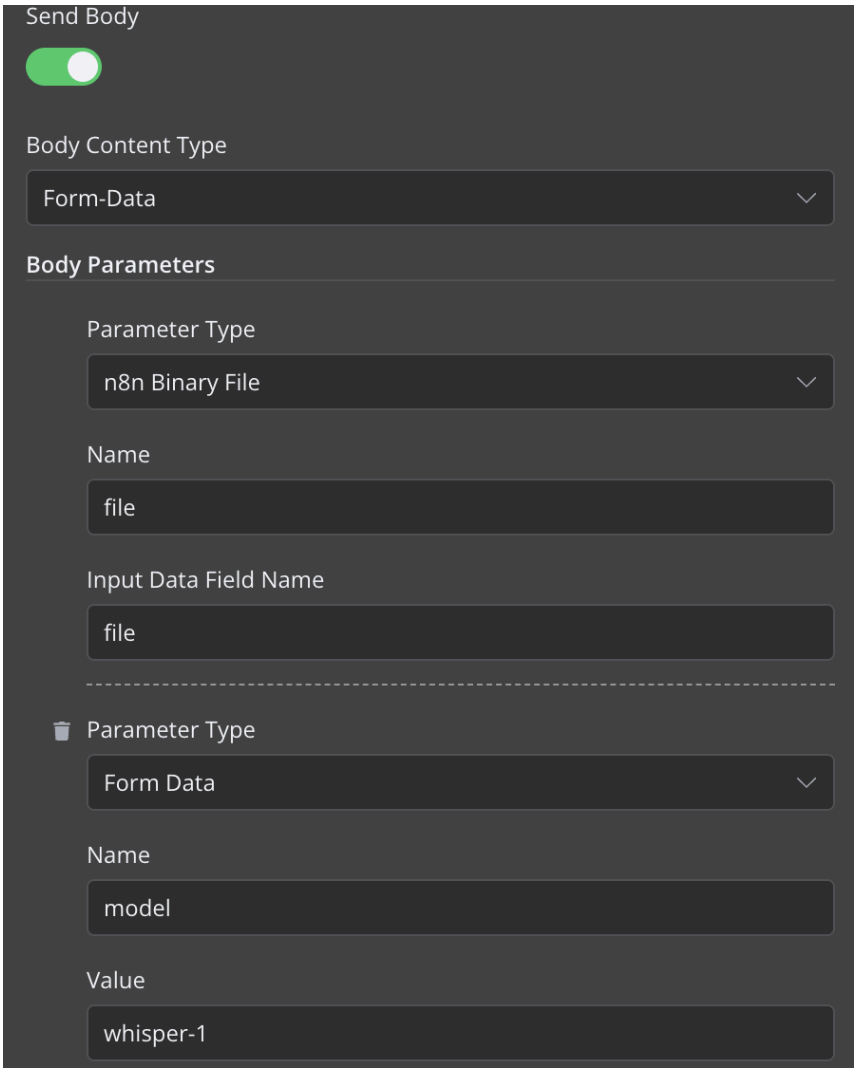
Nodo de whisper en http:

Curl estándar:

```
curl -X POST https://api.openai.com/v1/audio/transcriptions \
  -H "Authorization: Bearer tu_clave_api_de_openai" \
  -H "Content-Type: multipart/form-data" \
  -F "file=@ruta/a/tu/archivo/audio.mp3" \
  -F "model=whisper-1"
```


Import cURL
Method
POST
URL
https://api.openai.com/v1/audio/transcriptions
Authentication
Predefined Credential Type
Credential Type
OpenAi
OpenAi
OpenAi account
Send Query Parameters
Send Headers
Specify Headers
Using Fields Below


Send Query Parameters
Send Headers
Specify Headers
Using Fields Below
Header Parameters
Name
Content-Type
Value
multipart/form-data
Add Parameter


Send Body
Body Content Type
Form-Data
Body Parameters
Parameter Type
n8n Binary File
Name
file
Input Data Field Name
file
Parameter Type
Form Data
Name
model
Value
whisper-1

- **Archivos de Configuración:** Se utilizarán variables de entorno, principalmente para la configuración de Evolution API.

```
SERVER_TYPE=http
SERVER_PORT=8080
SERVER_URL=https://$(PRIMARY_DOMAIN)
SENTRY_DSN=
CORS_ORIGIN=*
CORS_METHODS=GET,POST,PUT,DELETE
CORS_CREDENTIALS=true
LOG_LEVEL=ERROR,WARN,DEBUG,INFO,LOG,VERBOSE,DARK,WEBHOOKS,WEBSOCKET
LOG_COLOR=true
LOG_BAILEYS=error
DEL_INSTANCE=false
DATABASE_PROVIDER=postgresql
DATABASE_CONNECTION_URI=postgres://postgres:1df116fd6f20a2587b2f@chatbots_postgresevo2:5432/chatbots
DATABASE_CONNECTION_CLIENT_NAME=evolution_v2
CACHE_REDIS_ENABLED=true
CACHE_REDIS_URI=redis://default:0102d09022d2b96b792b@chatbots_redisevo2:6379
CACHE_REDIS_PREFIX_KEY=evolution_v2
CACHE_REDIS_SAVE_INSTANCES=true
CACHE_LOCAL_ENABLED=true
CACHE_IGNORE_JIDS=[]
AUTHENTICATION_API_KEY=x7LsUTGvA1CXcNxZMPfzG2RwtyyMb2zz
AUTHENTICATION_EXPOSE_IN_FETCH_INSTANCES=true
LANGUAGE=en
DATABASE_SAVE_DATA_INSTANCE=true
DATABASE_SAVE_DATA_NEW_MESSAGE=true
DATABASE_SAVE_MESSAGE_UPDATE=true
DATABASE_SAVE_DATA_CONTACTS=true
DATABASE_SAVE_DATA_CHATS=true

DATABASE_SAVE_DATA_LABELS=true
DATABASE_SAVE_DATA_HISTORIC=true
RABBITMQ_ENABLED=false
RABBITMQ_URI=amqp://localhost
RABBITMQ_EXCHANGE_NAME=evolution
RABBITMQ_GLOBAL_ENABLED=false
WEBSOCKET_ENABLED=false
WA_BUSINESS_TOKEN_WEBHOOK=evolution
WA_BUSINESS_URL=https://graph.facebook.com
WA_BUSINESS_VERSION=v20.0
WA_BUSINESS_LANGUAGE=en_US
WEBHOOK_GLOBAL_ENABLED=true
WEBHOOK_GLOBAL_URL=''
WEBHOOK_GLOBAL_WEBHOOK_BY_EVENTS=true
WEBHOOK_EVENTS_*=true/false (Varias variables para eventos de webhook)
CONFIG_SESSION_PHONE_CLIENT=Evolution API
CONFIG_SESSION_PHONE_NAME=Chrome
CONFIG_SESSION_PHONE_VERSION=2.3000.1015901307
QRCODE_LIMIT=30
QRCODE_COLOR='#175197'
TYPEBOT_ENABLED=false
TYPEBOT_API_VERSION=latest
CHATWOOT_ENABLED=true
CHATWOOT_MESSAGE_READ=true
CHATWOOT_MESSAGE_DELETE=true
CHATWOOT_BOT_CONTACT=true
CHATWOOT_IMPORT_DATABASE_CONNECTION_URI=postgresql://user:passwprd@host:5432/chatwoot?sslmode=disable
CHATWOOT_IMPORT_PLACEHOLDER_MEDIA_MESSAGE=true
OPENAI_ENABLED=true
DIFY_ENABLED=true
S3_ENABLED=false
```

Storage -> Add volume Mount :

evolution_instances

/evolution/instances

Despliegue, Mantenimiento y Escalabilidad

- **Despliegue:** Se realizará en una instancia e2-standard-2 de Google Cloud, utilizando EasyPanel para la gestión del servidor.
    - *Detalle:* El proceso de despliegue inicial consistirá en configurar el entorno en la instancia de Google Cloud, instalar las dependencias (Docker, Node.js, etc.) y desplegar la aplicación.

- **Proceso de Despliegue (a futuro - se recomienda CI/CD):** Se recomienda implementar un proceso de Integración Continua/Entrega Continua (CI/CD) para automatizar el despliegue. Un flujo típico de CI/CD podría ser:
    - **Desarrollo:** Se realizan cambios en el código y se suben a un repositorio (ej. GitHub).
    - **Integración Continua (CI):** Un servidor de CI (ej. GitHub Actions, GitLab CI) detecta los cambios en el código.
    - **Pruebas:** Se ejecutan las pruebas automatizadas (unitarias, de integración, E2E).
    - **Construcción:** Se construye una nueva versión de la aplicación (ej. una imagen de Docker).
    - **Despliegue Continuo (CD):** La nueva versión se despliega automáticamente en el servidor de producción.

- **Monitoreo y Alertas:** Se utilizará EasyPanel para el monitoreo básico del servidor. Se recomienda complementar con herramientas más avanzadas como:
    - Prometheus + Grafana: Para monitoreo de métricas y visualización.
    - Datadog o New Relic: Plataformas de observabilidad más completas.
    - UptimeRobot o Cloud Monitoring (de Google Cloud): Para monitoreo de la disponibilidad del servidor.
    - *Detalle:* Estas herramientas permitirán monitorear el rendimiento del servidor (uso de CPU, memoria, disco), la disponibilidad de la aplicación, el tiempo de respuesta del chatbot y otros indicadores clave. Las alertas notificarán cuando se detecten problemas.

- **Escalabilidad:** Inicialmente, no se implementará escalabilidad automática. Se recomienda considerar estrategias de escalabilidad horizontal a futuro, como:
    - Balanceo de Carga: Distribuir el tráfico entre múltiples instancias del chatbot.
    - Réplicas de la Aplicación: Ejecutar múltiples copias del chatbot en diferentes servidores.
    - Base de Datos Escalable: Utilizar una base de datos que pueda escalar horizontalmente (ej. una base de datos NoSQL o una base de datos relacional con sharding).

- **Respaldo de Datos:** Se utilizará Duplicati para realizar copias de seguridad de los datos.
    - *Detalle:* Es importante definir una política de respaldo que especifique la frecuencia de las copias de seguridad y el lugar donde se almacenarán (ej. un almacenamiento en la nube).

- **Plan de Recuperación ante Desastres (DRP):** Se proporcionó una estructura general anteriormente. Los puntos clave son:
    - **Evaluación de Riesgos:** Identificar los posibles desastres que podrían afectar al sistema.
    - **Backups:** Realizar copias de seguridad regulares de los datos y la configuración.
    - **Procedimientos de Recuperación:** Definir los pasos a seguir para restaurar el sistema en caso de un desastre.
    - **RTO (Recovery Time Objective):** Tiempo máximo aceptable de inactividad.
    - **RPO (Recovery Point Objective):** Pérdida máxima aceptable de datos.
    - **Pruebas:** Probar periódicamente el plan de recuperación para asegurar su efectividad.
    - **Comunicación:** Establecer un plan de comunicación para informar a los usuarios en caso de un desastre.
    - **Automatización:** Automatizar los procesos de respaldo y recuperación siempre que sea posible.

- **Registro de Actualizaciones:** Se utilizará Notion para llevar un registro de las actualizaciones y cambios realizados en el proyecto.