



UNIVERSIDAD  
RICARDO PALMA  
RECTORADO



INSTITUTO DE  
DATOS E INTELIGENCIA  
ARTIFICIAL

*“Bienvenidos a la IA Generativa: El Futuro Está Aquí”*



**Miguel Cotrina**

Arquitecto de Datos & Educador en IA

### 🎓 Perfil Académico

- - Ingeniero de Software –  
Universidad Tecnológica del Perú
- - Magíster en Ciencia de Datos –  
Universidad Ricardo Palma

### 💼 Perfil Profesional

- - Arquitecto de Datos & IA – Indra, Región Perú
- - Instructor de Big Data, Cloud e IA Generativa



Conócame en  
LinkedIn

# Objetivo

Desarrollar una aplicación con inteligencia artificial generativa, usando las principales tecnologías y enfoques (langchain, rag y memoria) desplegarla como servicio y exponerla mediante un FrontEnd en vercel

# **Crea tu propia APP con inteligencia artificial generativa**

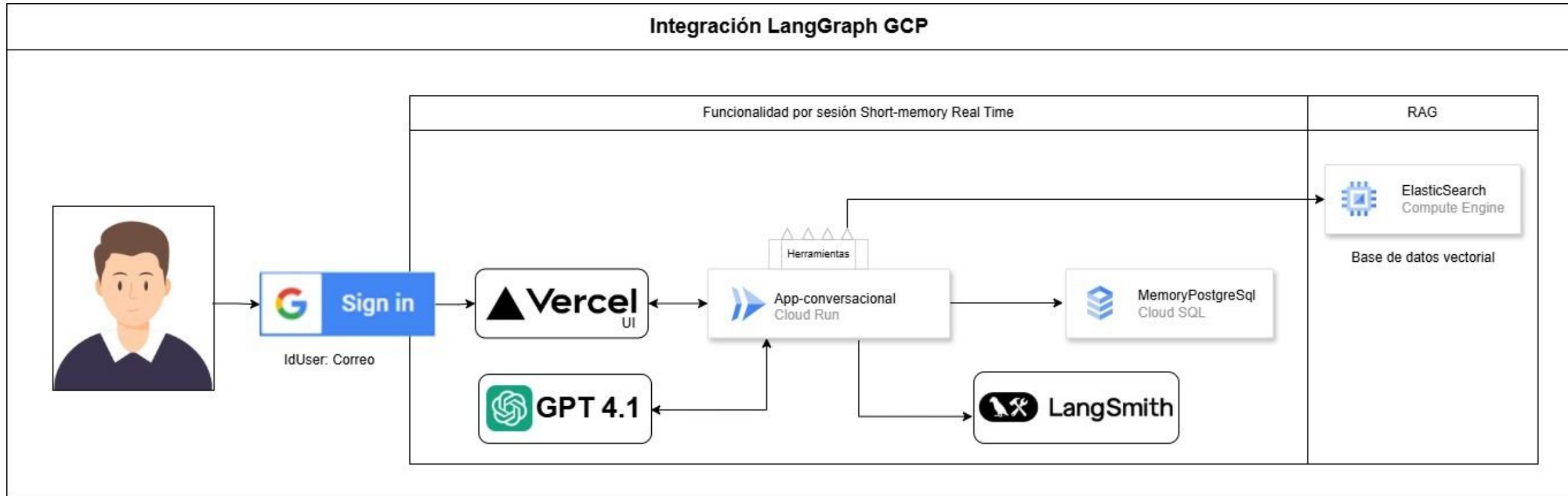
# El desafío

**Objetivo:** Responder preguntas de clientes con datos de la empresa, en segundos, sin perder contexto las 24 horas del día.

**Problema actual:** Información desactualizada, chatbots que “alucinan”, soluciones difíciles de escalar.

**Promesa del taller:** Veremos un camino *end-to-end* para pasar “de la idea al enlace de producción”.

# Mapa del stack



Modelo sugerido: "gpt-4.1-2025-04-14"

# Que es Langchain(Framework)

- **Biblioteca gratuita** que acerca la inteligencia artificial a tus proyectos sin complicaciones.
- **Funciona como un set de bloques:** combinas piezas ya preparadas (preguntar, buscar, recordar) y obtienes un asistente listo para usar.
- **Menos código, más resultado:** en pocas líneas creas un chatbot que consulta tus datos y responde con coherencia.
- Pensado para equipos que quieren **implementar IA rápido, de forma segura y fácil de mantener.**

# Que es RAG

- **Busca, complementa y responde:** antes de contestar, el asistente **busca** la información correcta, la **añade** al mensaje y luego **genera** la respuesta.
- Evita que la IA “se invente” datos: siempre apoya sus palabras en documentos reales de tu empresa.
- Actualizar conocimientos es fácil: subir nuevos archivos y el asistente los usará sin volver a entrenarlo.
- Ideal para preguntas sobre manuales, catálogos o políticas internas, porque combina la rapidez del modelo con la **precisión de tus propios datos**.

# Desplegar Elasticsearch en GCP - Paso 1

- 1) Crea una máquina virtual(Se recomienda la más pequeña para comenzar)
- 2) Entrar en el ssh y ejecutar los comandos:
  - a) `sudo apt-get install wget`
  - b) `wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo gpg --dearmor -o /usr/share/keyrings/elasticsearch-keyring.gpg`
  - c) `sudo apt-get install apt-transport-https`
  - d) `echo "deb [signed-by=/usr/share/keyrings/elasticsearch-keyring.gpg] https://artifacts.elastic.co/packages/8.x/apt stable main" | sudo tee /etc/apt/sources.list.d/elasticsearch-8.x.list`
  - e) `sudo apt-get update && sudo apt-get install elasticsearch`
  - f) `sudo /bin/systemctl daemon-reload`
  - g) `sudo /bin/systemctl enable elasticsearch.service`

Al finalizar te devolverá las credenciales para acceder a elasticsearch con el usuario por defecto elastic



# Desplegar Elasticsearch en GCP - Paso 2

1. Ingresa a el archivo elasticsearch.yml con el comando
  - a. `sudo nano /etc/elasticsearch/elasticsearch.yml`
2. Edita el archivo(Descomenta)
  - a. `network.host: 0.0.0.0`
  - b. `http.port: 9200`
3. Edita el archivo(coloca en false el valor de enabled)
  - a. `xpack.security.http.ssl:`  
  
`enabled: false`  
  
`keystore.path: certs/http.p12`

# Desplegar Elasticsearch en GCP - Paso 3

1. Inicializa el servicio
  - a. `sudo systemctl start elasticsearch.service`
2. (opcional) Detener el servicio
  - a. `sudo systemctl stop elasticsearch.service`
3. Apertura de puertos en firewall
  - a. Se recomienda aperturar el puerto 9200 en el firewall de GCP

Nota: Para poder ingresar a tu elasticsearch puedes usar elasticvue, se despliega como una extensión de Chrome

Referencias: <https://www.elastic.co/guide/en/elasticsearch/reference/current/deb.html>

# Desplegar Cloud SQL

- 1) Entra a la consola de GCP
- 2) Crea una instancia
- 3) Selecciona PostgreSQL
- 4) Se recomienda seleccionar la configuración más pequeña
- 5) Llenar los datos solicitados y dar crear instancia
- 6) Cuando finalice la creación ingresar desde el nombre de la instancia a su configuración
- 7) Ingresar a la sección connections y a la pestaña networking
- 8) Click en ADD A NETWORK e ingresar las ip desde donde se van a conectar, en caso estar realizando pruebas pueden colocar 0.0.0.0/0 para validar. pero no se recomienda

# Desplegar app en Google Cloud

1) Generar la imagen

```
gcloud builds submit --tag gcr.io/cobalto-data/talleria:latest
```

2) Desplegar en Cloud run

```
gcloud run deploy apicloudia--image gcr.io/cobalto-data/talleria:latest --platform  
managed --region us-west4 --allow-unauthenticated
```

3) Obtener la URL de despliegue

a) Las variables de entrada son agent y msg

<https://apicloudia-610199020496.us-west4.run.app/agent?msg=hola&idagente=abc1>

# Repositorio Front

1) Clonar repositorio original

```
git clone https://github.com/macespinoza/agentui-withlogin
```

```
cd agentui-withlogin
```

```
git remote remove origin
```

2) Agregar nuevo repositorio

```
git remote add origin <turepo>
```

```
git push -u origin main
```

# Realizar cambio en Front

1) Realizamos el cambio en la línea 6 por nuestra api desplegada

ruta: agentui-withlogin\src\app\api\agent\route.ts

```
1 // src/app/api/agent/route.ts
2 import type { NextRequest } from 'next/server';
3
4 export async function GET(request: NextRequest) {
5     // recrea la URL de tu API remota usando los mismos parámetros
6     const url = `apicloudia-610199020496.us-west4.run.app/agent?` +
7         new URL(request.url).searchParams.toString();
8
9     // forward
10    const apiRes = await fetch(url);
11    const text = await apiRes.text();
12
13    return new Response(text, {
14        status: apiRes.status,
15        headers: { 'Content-Type': 'text/plain' },
16    });
17 }
```

# Actualizar cambios en tu repositorio

1) Actualizamos los cambios en nuestro repositorio

`git add .`

`git commit -m "update"`

`git push`

# Creación de credenciales de Google Cloud

1) ingresa a la plataforma de google

**url :** <https://console.cloud.google.com/apis/credentials>

2) Seleccionar el tipo de credencial

**Crear credencial:** OAUTH

3) Rellena los datos de la aplicación

**Tipo de aplicación:** Web Application

**Nombre:** Definir uno

**Authorized redirect URIs**(Cambiar luego el dominio de despliegue creado en vercel):

<https://envdata-macespinozas-projects.vercel.app/api/auth/callback/google>



# Guardamos las variables

Variables Globales para el despliegue

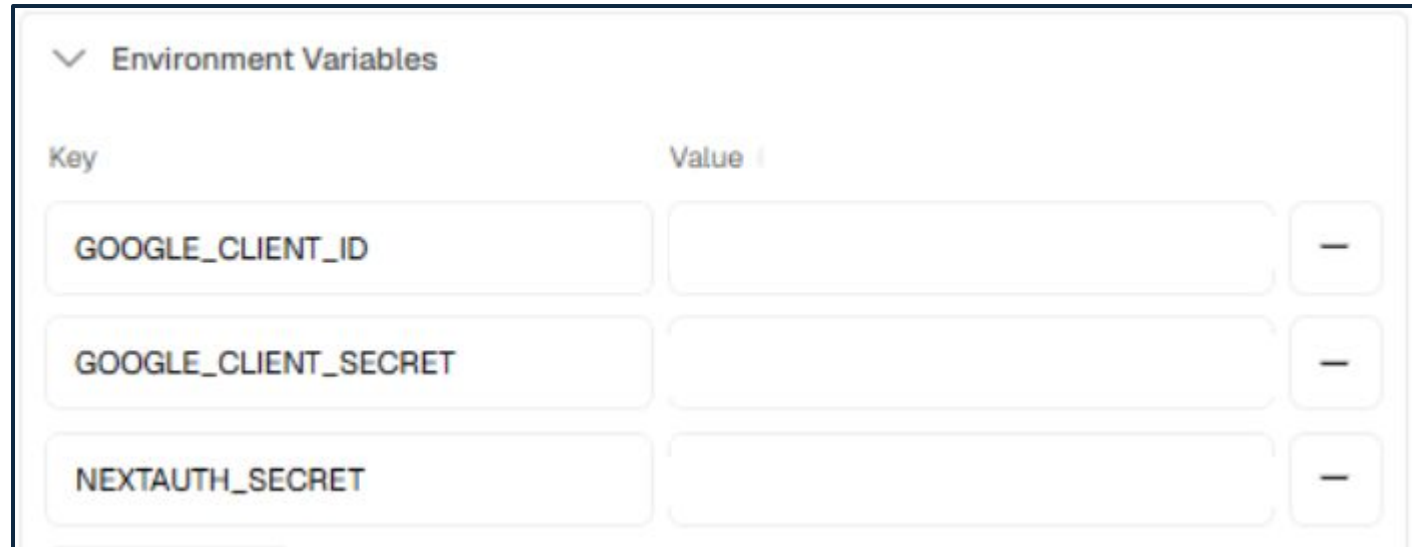
GOOGLE\_CLIENT\_ID : Client ID(Obtenido de GCP)

GOOGLE\_CLIENT\_SECRET: Client secret(Obtenido de GCP)

NEXTAUTH\_SECRET: Url para generar código  
base64(<https://auth-secret-gen.vercel.app/>)

# Creamos una cuenta en vercel

1. Creamos cuenta en Vercel
2. Creamos proyecto en vercel
3. Seleccionamos repositorio
4. Agregamos “Enviroment Variables” y en el valor colocamos los obtenidos en el slide anterior y luego desplegamos

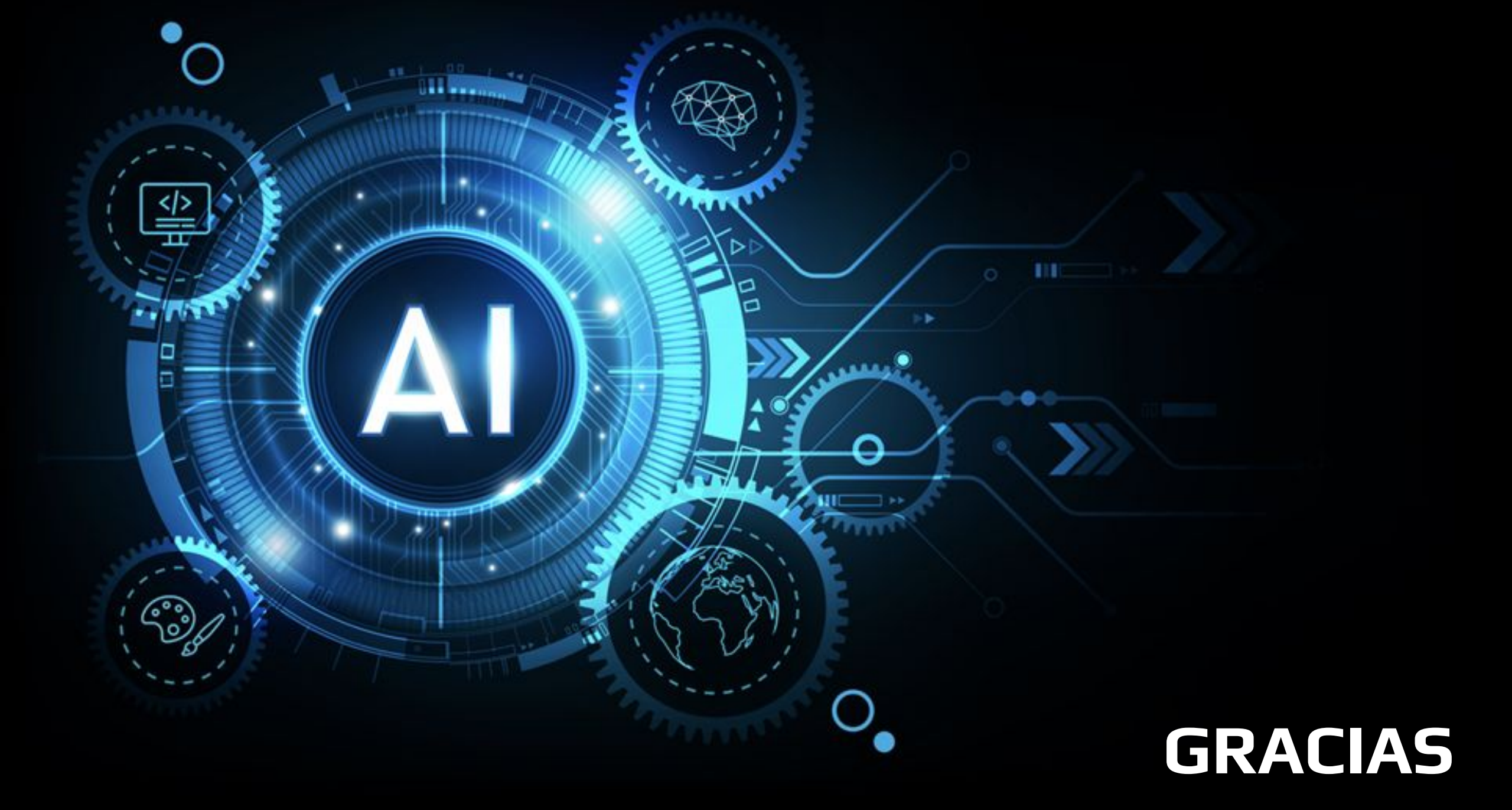


The screenshot shows the 'Environment Variables' section in the Vercel dashboard. It features a table with two columns: 'Key' and 'Value'. There are three rows of variables listed: 'GOOGLE\_CLIENT\_ID', 'GOOGLE\_CLIENT\_SECRET', and 'NEXTAUTH\_SECRET'. Each row has an empty input field for the value and a minus sign icon to the right, indicating that the values are not yet set.

Key	Value	
GOOGLE_CLIENT_ID		—
GOOGLE_CLIENT_SECRET		—
NEXTAUTH_SECRET		—

# Documentación/ Bibliografía

- Repositorio de proyecto
- <https://github.com/macespinoza/gcp-ai-agent-starter-kit>
-



**GRACIAS**