MANUAL DEL USUARIO · CENTRO DE AYUDA DE SIAC

API de IA de la plataforma

El motor de IA de SIAC, para tu propio código. Un endpoint compatible con OpenAI para que tus aplicaciones usen los modelos, las cuotas y la trazabilidad de SIAC. Si tu SDK habla con OpenAI, ya sabe hablar con SIAC: solo cambias la base URL y la clave.

Dirigido a	Plataforma	Tiempo	Pantallas capturadas
Desarrolladores	**SIAC v1.0.147**	**20 minutos**	**30 de septiembre de 2026**

Contenido

1 · Antes de empezar 3
2 · Cómo funciona 4
3 · Qué es esta API 5
4 · Crea tu clave de API 6
5 · Dentro de un servidor de Dev Studio 9
6 · Autenticación 11
7 · Descubrir modelos 12
8 · Chat completions 14
9 · Modelo Automático 17
10 · Errores y límites 2
11 · Huella de consumo 2
12 · Otros endpoints 22
13 · De consumo a costo 23
14 · Preguntas frecuentes 25

CAPÍTULO 1

Antes de empezar

PARA QUIÉN ES
Desarrolladores que integran IA en sus propias aplicaciones, scripts o servidores de Dev Studio, con los modelos y la tarifa de su organización.

TIEMPO ESTIMADO
20 minutos

PANTALLAS DE
SIAC v1.0.147 · capturadas el 30 de septiembre de 2026

Lo que necesita

- Una cuenta de SIAC activa, con al menos un modelo de IA aprobado para tu usuario.
- Una clave de API (la creas tú en **Ajustes**), o un servidor de Dev Studio con los modelos de la plataforma habilitados.
- Cualquier cliente que hable el protocolo de OpenAI (los SDK oficiales de Python o Node, curl, frameworks de agentes).

Tabla 1. Permisos

Rol	Qué puede hacer aquí
Usuario de la organización	Crear y revocar sus propias claves de API y llamar a la API con los modelos, límites y horario que su política le permite.
Administrador de la organización	Además, habilitar los modelos de la plataforma dentro de los servidores de Dev Studio de la organización y ver el consumo de cada servidor.

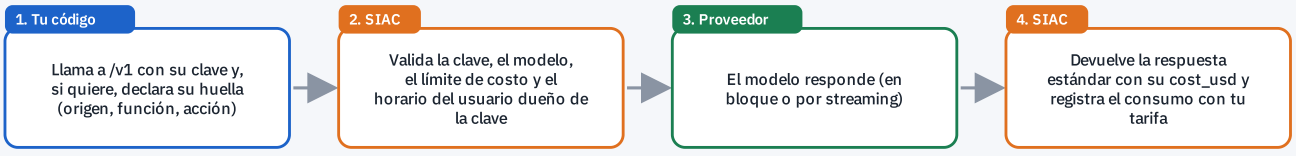
La misma API que ya usan opencode, los complementos de Office y los agentes, ahora para tu propio código. Esta es una **referencia técnica**: la versión no técnica del consumo está en el manual Huella de IA.

CAPÍTULO 2

Cómo funciona

De tu código a una respuesta con costo y trazabilidad

Tu aplicación llama a `https://ia.siac.com.co/v1` con una clave de API, exactamente como llamaría a OpenAI. SIAC valida la clave, aplica las reglas de tu organización (modelos aprobados, límites de costo, horario), envía la petición al proveedor del modelo y te devuelve la respuesta estándar **más su costo**. Nada se consume por fuera: cada llamada queda registrada con su modelo, sus tokens y su costo efectivo.



Qué pasa en cada llamada

CAPÍTULO 3

Qué es esta API

El mismo motor de IA de SIAC, para tu propio código

La plataforma expone un endpoint compatible con la API de OpenAI. La base URL **siempre termina en** `/v1`:

```
https://ia.siac.com.co/v1
```

Eso significa que puedes usar los SDK oficiales de OpenAI (Python, Node, etc.), o cualquier herramienta que hable ese protocolo, sin adaptadores. Cada petición se registra, se cobra con la **tarifa de tu organización** y respeta las **cuotas y modelos aprobados** para tu usuario.

CAPÍTULO 4

Crea tu clave de API

Fuera de un servidor de Dev Studio, tu clave sale de Ajustes

DÓNDE ESTÁ
Ajustes

DIRECCIÓN
`/settings`

QUIÉN LA USA
Cualquier usuario con acceso

CUÁNDO USARLA
Dónde salen las credenciales cuando tu código corre **fuera** de un servidor de Dev Studio: tu portátil (laptop), otro proveedor, un script suelto, opencode.

En Ajustes, la sección de claves te recuerda la dirección de la API 1: el **Endpoint compatible con OpenAI**, `https://ia.siac.com.co/v1`. Debajo, la tabla 2 lista tus claves con su prefijo, cuándo se crearon, su último uso y si están activas o revocadas; Revocar 3 desactiva una clave de inmediato. Arriba, en el encabezado de Ajustes, está Nueva clave: crea una clave con el nombre que le pongas.

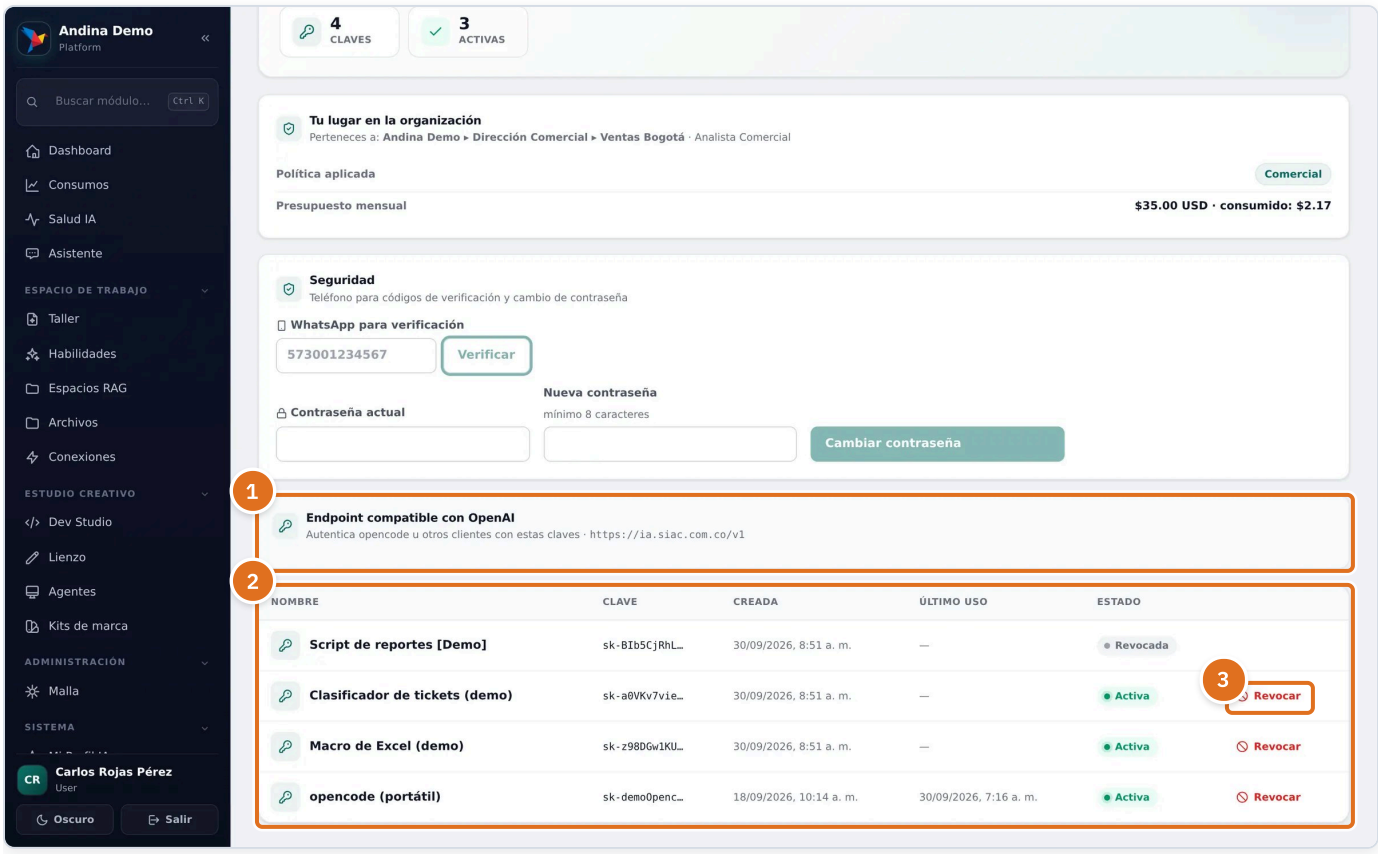


Figura 1. Tus claves de API en Ajustes: una en uso, una recién creada y una revocada.

1. **Endpoint compatible con OpenAI.** Dirección a la que se conectan opencode u otros clientes compatibles con OpenAI usando tus claves de API.
2. **Tus claves.** Tus claves de API con su prefijo, fecha de creación, último uso y si están activas o revocadas.
3. **Revocar.** Desactiva la clave tras confirmar; cualquier integración que la use deja de funcionar.

PASO A PASO

1. Abre **Ajustes** y baja hasta las claves de API.
2. Pulsa **Nueva clave**, arriba a la derecha.
3. Ponle un nombre que diga **para qué es** 1 (por ejemplo, «Clasificador de facturas») y pulsa **Crear clave** 2.

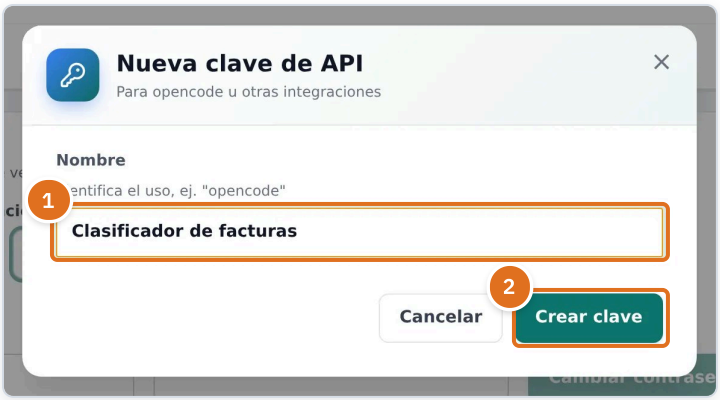


Figura 2. La ventana «Nueva clave», con el nombre escrito.

1. **Nombre.** Un nombre que identifique el uso de la clave (por ejemplo «opencode» o «Script de reportes»).
2. **Crear clave.** Crea la clave con el nombre escrito y la muestra una sola vez para que la copies.

4. **Copia la clave en ese momento.** La clave completa solo se muestra una vez; después solo verás su prefijo (`sk-…`).
5. Guárdala donde vive tu backend (una variable de entorno o un gestor de secretos), nunca en el código.

IMPORTANTE

Una clave **hereda** el límite de consumo, el horario y los modelos permitidos del usuario que la creó. Si la clave se filtra, revócala en **Ajustes**: pierde acceso de inmediato.

CONSEJO

Una clave por integración: si mañana revocas la del clasificador, el resto sigue funcionando, y la huella de consumo te dice qué gastó cada una. Más detalle en el manual de Ajustes.

Tabla 2. Qué hace cada elemento de «Crea tu clave de API» · los usan: Cualquier usuario con acceso

Elemento	Qué hace
Endpoint compatible con OpenAI	Dirección a la que se conectan opencode u otros clientes compatibles con OpenAI usando tus claves de API.
Tus claves	Tus claves de API con su prefijo, fecha de creación, último uso y si están activas o revocadas.
Revocar	Desactiva la clave tras confirmar; cualquier integración que la use deja de funcionar.
Nombre	Un nombre que identifique el uso de la clave (por ejemplo «opencode» o «Script de reportes»).
Crear clave	Crea la clave con el nombre escrito y la muestra una sola vez para que la copies.

CAPÍTULO 5

Dentro de un servidor de Dev Studio

No creas claves a mano: el servidor ya las trae en su entorno

DÓNDE ESTÁ
Dev Studio › Ficha del servidor › Ajustes

DIRECCIÓN
`/devstudio`

QUIÉN LA USA
Administradores de la organización (dueños del servidor).

CUÁNDO USARLA
Cuando tu código corre en un servidor de Dev Studio: el Constructor, tus Espacios o un proceso propio.

En la ficha del servidor, pestaña **Ajustes**, la sección **Modelos de IA de la plataforma** 1 le da al servidor su propia clave. Al pulsar **Habilitar** se genera un token y queda inyectado en `~/.siac/ai.env`, con estas variables:

```
SIAC_IA_BASE_URL=https://ia.siac.com.co/v1
SIAC_IA_API_KEY=sk-...
# Alias para herramientas que esperan los nombres de OpenAI:
OPENAI_BASE_URL=https://ia.siac.com.co/v1
OPENAI_API_KEY=sk-...
```

En una terminal del IDE las variables ya están disponibles; para un proceso propio, carga el archivo con `source ~/.siac/ai.env`. En tus scripts basta con leer las variables de entorno: cualquier SDK compatible con OpenAI funciona sin tocar código. Desde la misma sección puedes **Rotar token** (el anterior deja de servir) o **Deshabilitar** los modelos.

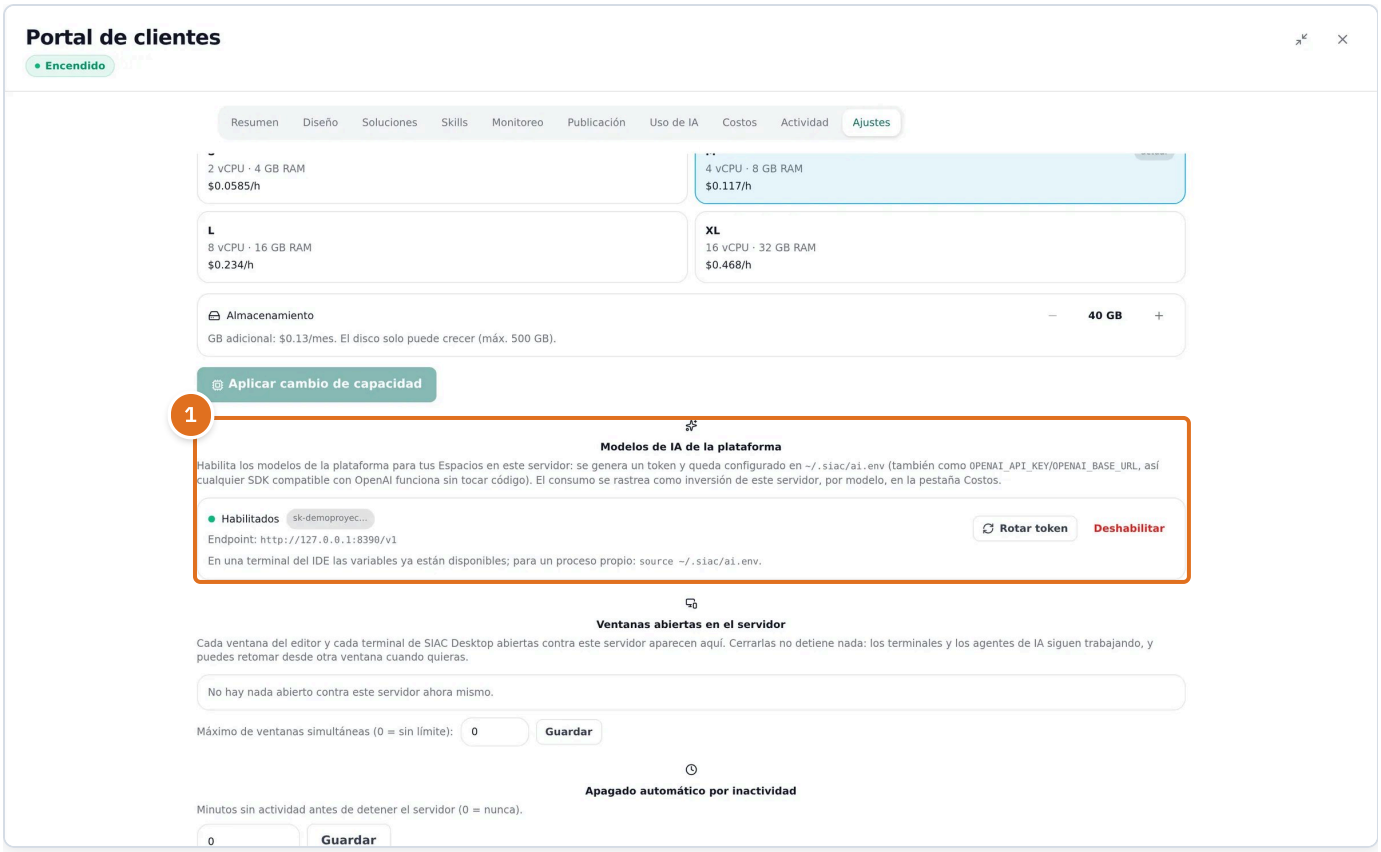


Figura 3. Los modelos de la plataforma habilitados en un servidor de Dev Studio, con el prefijo de su token y su endpoint (en la pantalla de ejemplo, el de un entorno de pruebas; en el tuyo verás https://ia.siac.com.co/v1).

1. **Modelos de IA de la plataforma.** Habilita los modelos de SIAC para las aplicaciones de este servidor: deja una llave lista en el servidor y el consumo se suma a su inversión.

REGLA DEL SISTEMA

El consumo que sale con esa clave se atribuye solo al servidor: aparece como inversión de ese servidor, por modelo, en sus pestañas **Uso de IA** y **Costos**.

Tabla 3. Qué hace cada elemento de «Dentro de un servidor de Dev Studio» · los usan: Cualquier usuario con acceso

Elemento	Qué hace
Modelos de IA de la plataforma	Habilita los modelos de SIAC para las aplicaciones de este servidor: deja una llave lista en el servidor y el consumo se suma a su inversión.

CAPÍTULO 6

Autenticación

Bearer token, y la clave nunca en el frontend

Todas las peticiones llevan la clave en la cabecera estándar:

```
Authorization: Bearer sk-...
```

Regla de oro: la clave vive solo en el backend. Nunca la incrustes en JavaScript del navegador, en una app móvil ni en código que se distribuya: cualquiera podría extraerla y consumir con tu cuota. Si tu interfaz necesita IA, haz que el frontend llame a un endpoint propio de tu backend, y que sea el backend quien hable con `/v1`.

IMPORTANTE

Una clave hereda el límite de consumo, el horario y los modelos permitidos del usuario que la creó. Si la clave se filtra, revócala en **Ajustes**: pierde acceso de inmediato.

CAPÍTULO 7

Descubrir modelos

GET /v1/models — el catálogo manda

Devuelve los modelos **aprobados para tu usuario**, en el formato estándar de OpenAI, con campos extra propios de SIAC:

```
curl -s https://ia.siac.com.co/v1/models \
  -H "Authorization: Bearer $SIAC_IA_API_KEY"
```


```
{
  "object": "list",
  "data": [
    {
      "id": "glm-5.2",
      "object": "model",
      "display_name": "GLM-5.2",
      "context_limit": 202752,
      "input_per_1m_usd": 0.83,
      "output_per_1m_usd": 3.34
    }
  ]
}
```


Los campos extra

- `display_name` — nombre legible del modelo, para mostrar en tu interfaz.
- `context_limit` — tamaño máximo de contexto, en tokens.
- `input_per_1m_usd` — precio efectivo por millón de tokens de entrada, ya con la tarifa de tu organización.
- `output_per_1m_usd` — precio efectivo por millón de tokens de salida.
- `cache_read_per_1m_usd` y `cache_write_per_1m_usd` — precio efectivo del caché de contexto (cuando el proveedor lo confirma, la lectura desde caché sale más barata; si no, vale lo mismo que la entrada).

Solo ves el precio **efectivo** para tu organización: la API nunca expone el precio base ni el margen.

REGLA DEL SISTEMA

Elige el modelo consultando el catálogo, no de memoria. Los modelos disponibles y sus precios cambian con el tiempo; si tu código fija un id que ya no está aprobado, recibirás un 403. Lo robusto es listar `/v1/models` al arrancar (o cachearlo unos minutos) y elegir según `context_limit` y precio.

CONSEJO

Cuando un proveedor retira un modelo, SIAC redirige las peticiones a su reemplazo el día del retiro: tu código no se rompe, pero conviene migrar el id con tiempo.

CAPÍTULO 8

Chat completions

POST /v1/chat/completions — igual que OpenAI, con costo incluido

curl

```
curl -s https://ia.siac.com.co/v1/chat/completions \
  -H "Authorization: Bearer $SIAC_IA_API_KEY" \
  -H "Content-Type: application/json" \
  -H "X-SIAC-Origin: api" \
  -H "X-SIAC-Feature: clasificador-tickets" \
  -d '{
    "model": "glm-5.2",
    "messages": [
      {"role": "system", "content": "Clasificas tickets de soporte en: facturacion, tecnico,
ventas."},
      {"role": "user", "content": "No me llega la factura de julio"}
    ]
  }'
```

La respuesta es la estándar de OpenAI, más un campo aditivo `cost_usd` con el **costo efectivo** de ese request (tokens de entrada y salida, con tu tarifa):

```
{
  "id": "chatcmpl-...",
  "choices": [{ "message": { "role": "assistant", "content": "facturacion" }, ... }],
  "usage": { "prompt_tokens": 41, "completion_tokens": 3, "total_tokens": 44 },
  "cost_usd": 0.000044
}
```

Python (SDK de OpenAI)

```
import os
from openai import OpenAI

client = OpenAI(
    base_url=os.environ["SIAC_IA_BASE_URL"],  # termina en /v1
    api_key=os.environ["SIAC_IA_API_KEY"],
    default_headers={"X-SIAC-Origin": "worker"},
)

resp = client.chat.completions.create(
    model="glm-5.2",
    messages=[{"role": "user", "content": "Resume este pedido: ..."}],
    extra_headers={"X-SIAC-Feature": "clasificador-tickets", "X-SIAC-Action": "resumen"},
)
print(resp.choices[0].message.content)
```

Node (SDK de OpenAI)

```
import OpenAI from "openai";

const client = new OpenAI({
  baseURL: process.env.SIAC_IA_BASE_URL, // termina en /v1
  apiKey: process.env.SIAC_IA_API_KEY,
  defaultHeaders: { "X-SIAC-Origin": "web-chat" },
});

const resp = await client.chat.completions.create(
  { model: "glm-5.2", messages: [{ role: "user", content: "Hola" }] },
  { headers: { "X-SIAC-Feature": "asesor-repuestos" } }
);
console.log(resp.choices[0].message.content);
```

Streaming (SSE)

Pasa `"stream": true` y la respuesta llega como **Server-Sent Events**: una secuencia de eventos `data:`, cada uno un fragmento (`delta`) del mensaje, y un `data: [DONE]` final. Cada evento termina en una **línea en blanco** (`\n\n`): si escribes tu propio parser o un proxy intermedio, respeta ese separador o los fragmentos llegarán vacíos o pegados.

```
data: {"id":"chatcmpl-...","choices":[{"delta":{"content":"Hola"}}]}

data: {"id":"chatcmpl-...","choices":[{"delta":{"content":", ¿en qué"}}]}

data: [DONE]
```

Con los SDK no tienes que parsear nada:

```
# Python
stream = client.chat.completions.create(
    model="glm-5.2",
    messages=[{"role": "user", "content": "Escribe un haiku"}],
    stream=True,
)
for chunk in stream:
    delta = chunk.choices[0].delta.content
    if delta:
        print(delta, end="", flush=True)
```

Herramientas (tools)

El campo `tools` usa el formato estándar de OpenAI (**function calling**): el modelo responde con `tool_calls`, tu código ejecuta la función y devuelve el resultado en un mensaje con `role: "tool"`.

```
{
  "model": "glm-5.2",
  "messages": [{"role": "user", "content": "¿Cuánto vale el filtro F-204?"}],
  "tools": [{
    "type": "function",
    "function": {
      "name": "buscar_repuesto",
      "description": "Busca un repuesto por referencia y devuelve precio y stock",
      "parameters": {
        "type": "object",
        "properties": { "referencia": { "type": "string" } },
        "required": ["referencia"]
      }
    }
  }]
}
```

Todo lo que ya sabes del ecosistema OpenAI —SDKs, frameworks de agentes, function calling— funciona igual. SIAC solo añade **control, precio efectivo y trazabilidad**.

CAPÍTULO 9

Modelo Automático

siac-auto — el router elige por ti

Si tu plan lo incluye, verás en el catálogo el modelo `siac-auto` (lo habilita tu organización). Al usarlo, la plataforma **clasifica la tarea** y elige el modelo más adecuado por costo y capacidad: una clasificación rápida no necesita el mismo modelo que un análisis largo. En `/v1/models` aparece primero, con precio 0: el precio real es el del modelo que atienda cada request.

```
resp = client.chat.completions.create(
    model="siac-auto",
    messages=[{"role": "user", "content": "..."}],
)
```

Para que sepas qué decidió el router, la respuesta incluye dos cabeceras:

- `X-SIAC-Model` — el modelo real que atendió el request.
- `X-SIAC-Task` — el tipo de tarea que detectó el router.

El costo (`cost_usd`) **siempre corresponde al modelo real elegido**, y así queda registrado en tu consumo. Si el modelo elegido agota su cuota con el proveedor, el router reintenta con el siguiente candidato sin que tu código lo note.

CAPÍTULO 10

Errores y límites

Qué significan y cómo reaccionar

- `400` — **Modelo no disponible**: el id no existe en el catálogo o está deshabilitado. *Qué hacer:* consulta `/v1/models` y usa uno de la lista.
- `401` — **Clave inválida, revocada o rotada.** *Qué hacer:* no reintentes; consigue una clave válida (**Ajustes**, o el `ai.env` actualizado del servidor).
- `403` — **El modelo pedido no está permitido para tu usuario** («No tienes permiso para usar el modelo…»), o estás **fuera del horario o de los días** que permite tu política («El uso de IA está restringido al horario…»). *Qué hacer:* con el modelo, consulta `/v1/models` y usa uno del catálogo, o pide que te habiliten el modelo; con el horario, espera a que se abra la ventana.
- `429` — **Cuota del proveedor del modelo agotada por ahora.** *Qué hacer:* reintenta con backoff exponencial con tope (ej. 1 s, 2 s, 4 s… hasta 30 s) o cambia a otro modelo del catálogo.
- `429` con el mensaje «Has alcanzado el límite mensual…» — **tu tope de costo** (el de tu usuario, tu área, tu organización o la propia clave). *Qué hacer:* no reintentes: no cambiará hasta que suba el límite o empiece el mes.

Además, tu organización puede configurar **topes de costo** (límite mensual por usuario, por área o por clave) y **horarios permitidos**. Fuera de esos límites la API rechaza el request con un mensaje claro; tu código debe tratarlos como **errores no reintentables** hasta que cambie la condición. Lee el texto del error antes de decidir si reintentas: el mismo 429 puede ser el proveedor (pasa en segundos) o tu presupuesto (no pasa solo).

CAPÍTULO 11

Huella de consumo

Declara quién gasta, en qué y para qué — la parte más importante

Cada llamada puede (y debe) declarar **tres niveles de contexto** con cabeceras de petición. Con eso, el dueño del negocio ve en qué se van los tokens de su servidor, sin leer código:

- `X-SIAC-Origin` — **canal o superficie** por donde entra el trabajo. Ejemplos: `web-chat`, `api`, `worker`, `whatsapp`, `panel-admin`.
- `X-SIAC-Feature` — **función de negocio** que consume. Ejemplos: `asesor-repuestos`, `clasificador-tickets`.
- `X-SIAC-Action` — **etapa** dentro de la función (opcional). Ejemplos: `triage`, `respuesta`, `resumen`, `vision`.

El **servidor** y el **modelo** no se declaran: se atribuyen solos a partir de la clave de API y del request. Tú solo aportas los tres niveles de arriba.

Formato de los slugs

- **kebab-case**: minúsculas, sin acentos ni espacios, palabras separadas por guiones (también se admiten `.` y `_`). SIAC normaliza lo obvio: pasa a minúsculas, quita las tildes y cambia los espacios por guiones.
- **Máximo 40 caracteres.**
- Un slug inválido **se ignora** y ese consumo aparece como `sin-etiqueta` en el dashboard (o con la etiqueta que se deriva de la clave, como `ide` para el Constructor). **Nunca produce un error**: etiquetar mal jamás rompe tu aplicación.

Alternativa por body

Si no puedes tocar las cabeceras (por ejemplo, un framework que no las expone), puedes mandar lo mismo dentro del body:

```
{
  "model": "glm-5.2",
  "messages": [...],
  "metadata": { "origin": "worker", "feature": "clasificador-tickets", "action": "triage" }
}
```

La plataforma consume ese campo; **jamás llega al proveedor del modelo**. Si mandas las dos cosas, mandan las cabeceras.

Patrón recomendado

Un único módulo de IA por proyecto: el **origen fijo** en `default_headers` (se declara una vez) y la **función por llamada** en `extra_headers`:

```
# Python — ia.py, el único lugar del proyecto que habla con la API
import os
from openai import OpenAI

client = OpenAI(
    base_url=os.environ["SIAC_IA_BASE_URL"],
    api_key=os.environ["SIAC_IA_API_KEY"],
    default_headers={"X-SIAC-Origin": "worker"},  # fijo para todo el proyecto
)

def completar(feature: str, messages: list, action: str | None = None, **kw):
    headers = {"X-SIAC-Feature": feature}
    if action:
        headers["X-SIAC-Action"] = action
    return client.chat.completions.create(
        model=kw.pop("model", "glm-5.2"),
        messages=messages,
        extra_headers=headers,
        **kw,
    )

# En el resto del código:
completar("clasificador-tickets", msgs, action="triage")
```

```
// Node — ia.js, equivalente
import OpenAI from "openai";

const client = new OpenAI({
  baseURL: process.env.SIAC_IA_BASE_URL,
  apiKey: process.env.SIAC_IA_API_KEY,
  defaultHeaders: { "X-SIAC-Origin": "web-chat" }, // fijo para todo el proyecto
});

export function completar(feature, messages, { action, ...opts } = {}) {
  const headers = { "X-SIAC-Feature": feature };
  if (action) headers["X-SIAC-Action"] = action;
  return client.chat.completions.create(
    { model: "glm-5.2", messages, ...opts },
    { headers }
  );
}
```

Cómo diseñar buenos slugs

- **El origen es la superficie, no la función.** Si el mismo asesor atiende por la web y por WhatsApp, la función sigue siendo `asesor-repuestos`; lo que cambia es el origen (`web-chat` vs. `whatsapp`).

- **La función es una palabra de negocio**, que el dueño reconocería sin leer código (`asesor-repuestos`, no `handler-v2`), y **estable en el tiempo**: si la renombras cada semana, las series históricas se parten. Para mostrar un nombre bonito no cambies el slug: el dueño le pone nombre legible en la pestaña **Uso de IA**.
- **Nunca pongas ids de usuario o de conversación, timestamps ni datos personales en un slug.** Rompen la agregación (cada valor único crea una serie nueva) y disparan el **freno de cardinalidad**: por servidor caben 32 orígenes, 400 funciones y 128 acciones distintas; el excedente cae en el cajón `otros` y pierdes justo el detalle que buscabas.

CAPÍTULO 12

Otros endpoints

Espacios RAG y piezas de marca

- `POST /v1/knowledge/search` — busca en un **Espacio RAG** de tu organización (los documentos que subes en **Conocimiento › Espacios RAG**, antes llamado Proyectos) y devuelve los pasajes más relevantes con su fuente. Recibe `rag_space_id`, `query` y `k` (cuántos pasajes; 5 por defecto). Útil para dar contexto real a tus prompts (RAG) sin montar tu propio índice.
- `POST /v1/brand/piece` — genera piezas con tu **kit de marca** (colores, logo y tipografía de tu organización) listas para usar en documentos o presentaciones.

Ambos usan la **misma autenticación** y la **misma huella de consumo** que `/v1/chat/completions`.

CAPÍTULO 13

De consumo a costo

Cada request queda registrado, y lo ves en el panel

DÓNDE ESTÁ
Dev Studio › Ficha del servidor › Uso de IA

DIRECCIÓN
`/devstudio`

QUIÉN LA USA
Administradores de la organización.

CUÁNDO USARLA
Cuando quieras saber cuánto gasta tu aplicación y en qué: por día, por origen, por función y por modelo.

Cada llamada a la API queda registrada con su modelo, tokens y **costo efectivo** (calculado con la tarifa de tu organización — el mismo valor que llega en `cost_usd`). No hay consumo invisible.

Si tu código corre en un servidor de Dev Studio, abre la ficha del servidor. La pestaña **Uso de IA** 1 muestra el consumo tipificado por la huella: las cifras del periodo 2, las **barras por día** 3 y el desglose **origen → función → acción → modelo** 4, con el costo de cada cosa. La pestaña **Costos** 5 da los totales del servidor, IA incluida.

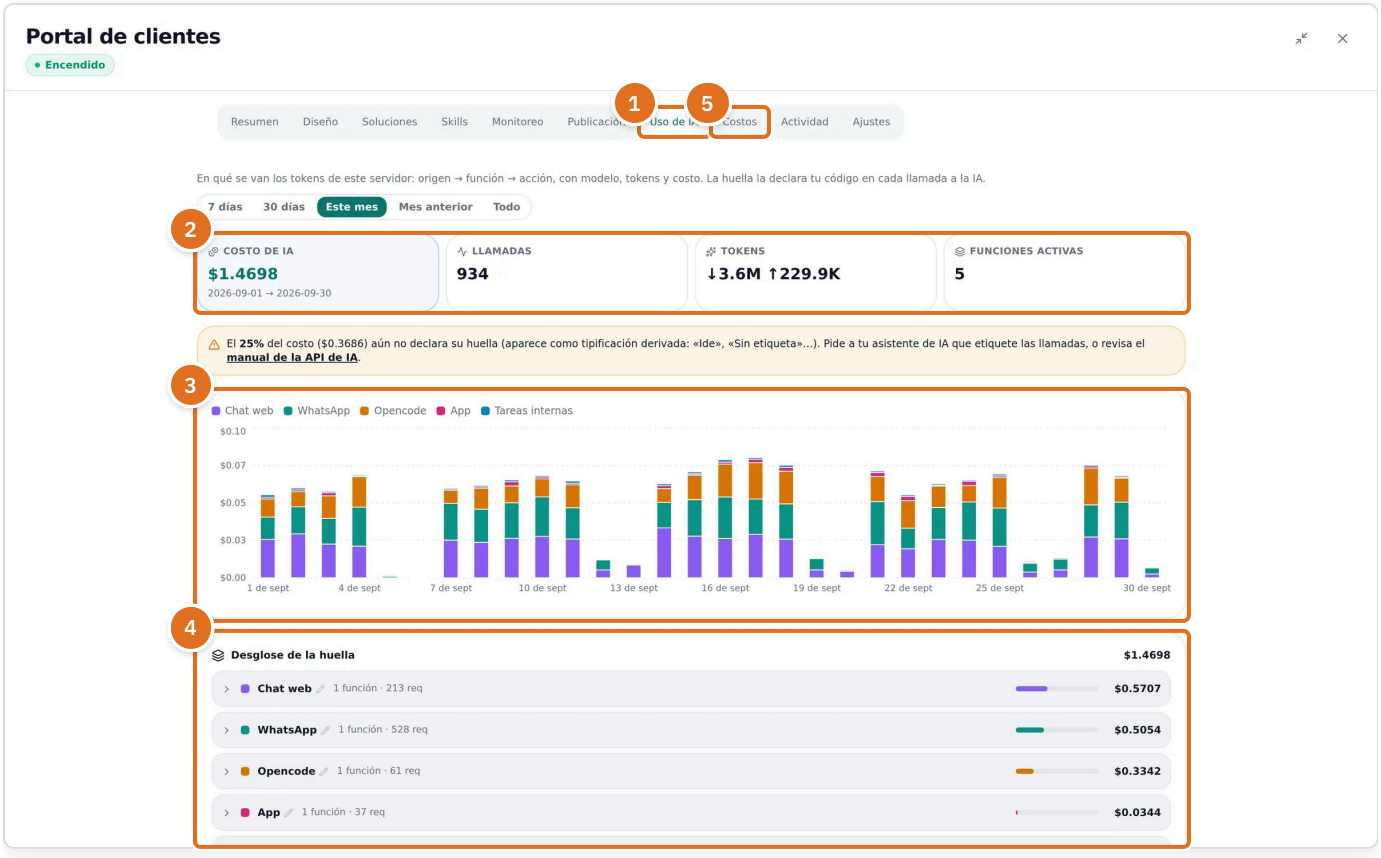


Figura 4. La pestaña «Uso de IA» de un servidor cuya aplicación declara su huella.

1. **Uso de IA.** La huella de IA de este servidor: por qué canal entró cada llamada, qué función del negocio la pidió y en qué etapa, con modelo, tokens y costo.
2. **Indicadores del periodo.** El costo total de IA del rango, las llamadas, los tokens de entrada y salida y cuántas funciones consumieron.
3. **Consumo por día.** Una barra por día, apilada por origen (un color por canal): muestra qué día se disparó el consumo y por qué canal. Pasa el cursor para ver el detalle.
4. **Desglose de la huella.** La tabla principal: cada origen se abre en sus funciones, cada función en sus acciones y sus modelos, siempre con costo y tokens.
5. **Costos.** Lo que ha costado este servidor: infraestructura e IA, en el tiempo.

CONSEJO

Si una parte importante del costo aún no declara su huella (en la pantalla, el 26 %), la pestaña te avisa y enlaza a este manual: es la señal para etiquetar las llamadas que faltan.

Tabla 4. Qué hace cada elemento de «De consumo a costo» · los usan: Cualquier usuario con acceso

Elemento	Qué hace
Uso de IA	La huella de IA de este servidor: por qué canal entró cada llamada, qué función del negocio la pidió y en qué etapa, con modelo, tokens y costo.
Indicadores del periodo	El costo total de IA del rango, las llamadas, los tokens de entrada y salida y cuántas funciones consumieron.
Consumo por día	Una barra por día, apilada por origen (un color por canal): muestra qué día se disparó el consumo y por qué canal. Pasa el cursor para ver el detalle.
Desglose de la huella	La tabla principal: cada origen se abre en sus funciones, cada función en sus acciones y sus modelos, siempre con costo y tokens.
Costos	Lo que ha costado este servidor: infraestructura e IA, en el tiempo.

Dónde verlo:

- **Ficha del servidor → pestaña «Uso de IA»**: el consumo tipificado por la huella (origen → función → acción → modelo).
- **Ficha del servidor → pestaña «Costos»**: los totales del servidor, IA incluida.
- **Inversión en servidores** (en la lista de Dev Studio): la vista consolidada de toda tu organización, servidor por servidor.

Los tres **cuadran entre sí**: la huella no cambia el cobro, solo lo explica. Si tu app etiqueta bien, ese reporte se explica solo — la versión no técnica está en el manual Huella de IA. Fuera de Dev Studio, tu consumo por clave aparece en **Consumos** y en **Registros**.

CAPÍTULO 14

Preguntas frecuentes

¿Tengo que cambiar mi código si ya uso OpenAI?

No: cambias la base URL (`https://ia.siac.com.co/v1`) y la clave. Los SDK, el streaming y las tools funcionan igual.

¿Puedo llamar a la API desde el navegador o una app móvil?

No. La clave vive solo en tu backend; el frontend llama a un endpoint propio de tu backend, y es el backend quien habla con `/v1`.

¿Por qué recibo 403 con un modelo que ayer funcionaba?

Porque ya no está aprobado para tu usuario, o estás fuera del horario que permite tu política. Lee el mensaje del error y consulta `/v1/models`: el catálogo manda.

¿Qué pasa si no declaro la huella?

Nada se rompe: el consumo se cobra igual y aparece como `sin-etiqueta` (o con lo que se deriva de la clave). Solo pierdes la explicación de en qué se gastó.

Se me filtró una clave, ¿qué hago?

Revócala en **Ajustes** con **Revocar**: pierde acceso de inmediato. Crea otra y cámbiala en tu backend.