Arquitectura de Datos e Integración Escalable para el Ecosistema GoHighLevel: Diseño en Supabase y Sincronización Event-Driven para Agentes de IA

1. Introducción y Contexto Arquitectónico

La evolución de las plataformas de gestión de relaciones con clientes (CRM) hacia ecosistemas abiertos y programables ha alcanzado un hito significativo con la transición de GoHighLevel (GHL) hacia su API V2.0 y la introducción de herramientas avanzadas para la creación de Agentes de Inteligencia Artificial. Este informe técnico aborda, con un nivel de detalle exhaustivo, el diseño, modelado e implementación de una infraestructura de datos externa basada en Supabase (PostgreSQL), destinada a servir como el "cerebro" persistente para aplicaciones avanzadas y agentes autónomos que operan sobre el ecosistema de GHL.
El contexto actual, impulsado por el concurso de Agentes de IA de GHL, exige una arquitectura que trascienda la simple integración punto a punto. Los desarrolladores y arquitectos de soluciones se enfrentan al reto de construir sistemas que no solo lean y escriban datos, sino que comprendan el contexto semántico de las interacciones empresariales. Para lograr esto, es imperativo establecer una "fuente de verdad" sincronizada en tiempo real que mitigue las limitaciones inherentes a las APIs REST tradicionales —como los límites de tasa (rate limits) y la latencia de red— y habilite capacidades analíticas y vectoriales nativas.
La arquitectura propuesta en este documento se fundamenta en un paradigma orientado a eventos (Event-Driven Architecture), utilizando la robustez de los *Private Integration Tokens* para la seguridad, la flexibilidad de los esquemas JSONB de PostgreSQL para manejar la variabilidad de los datos de GHL, y la inmediatez de los Webhooks para mantener la consistencia de los datos. A lo largo de este análisis, desglosaremos cada entidad maestra de GHL, desde la estructura básica de una ubicación de negocio hasta la complejidad de los hilos de conversación multicanal, traduciendo las especificaciones JSON de la API en modelos relacionales optimizados para Supabase.

1.1 El Cambio de Paradigma: De Polling a Event-Driven

Históricamente, las integraciones con CRMs dependían de ciclos de *polling* (consultas periódicas) para detectar cambios. En el entorno de GHL API V2, este enfoque es insostenible e ineficiente debido a los límites de tasa estrictos : 100 peticiones por ráfaga de 10 segundos y un tope diario de 200,000 peticiones. Para un Agente de IA que necesita contexto inmediato sobre el estado de un lead o el historial de una conversación, esperar al siguiente ciclo de polling es inaceptable.
La solución que exploraremos invierte el flujo de control: GHL notifica a nuestra infraestructura (Supabase) sobre cada cambio de estado relevante a través de Webhooks. Esto permite que

nuestra base de datos en Supabase mantenga un "espejo" casi en tiempo real de los datos de GHL. Así, las operaciones de lectura intensiva —como las requeridas para construir el contexto de un LLM (Large Language Model)— se ejecutan localmente contra PostgreSQL, con latencias de milisegundos y sin consumir cuota de API, reservando las llamadas a la API de GHL exclusivamente para las mutaciones (escrituras) o la sincronización inicial.

2. Estrategia de Autenticación y Seguridad

La seguridad es el cimiento sobre el cual se construye cualquier integración de datos sensibles. GHL ofrece dos mecanismos principales en su API V2: OAuth 2.0 y *Private Integrations Tokens*. Para el propósito de construir un backend persistente y controlado para Agentes de IA, el análisis dicta una preferencia clara hacia los tokens de integración privada, aunque es crucial entender ambos mecanismos para escenarios de aplicaciones distribuidas en el Marketplace.

2.1 Private Integration Tokens: La Llave Maestra para Integraciones Internas

Los *Private Integrations Tokens* representan una evolución significativa respecto a las API Keys de la versión 1.0. Mientras que las API Keys otorgaban acceso indiscriminado, los tokens privados funcionan como *Access Tokens* de OAuth2 estáticos pero con alcances (scopes) definidos y granulares.

2.1.1 Anatomía y Ciclo de Vida del Token

Técnicamente, un Private Integration Token es una cadena de caracteres que encapsula los permisos concedidos a una integración específica dentro de una sub-cuenta (location) o agencia. A diferencia del flujo estándar de OAuth, estos tokens no expiran en el corto plazo (24 horas) y no requieren la gestión compleja de *refresh tokens*.

- **Seguridad Basada en Scopes:** Al crear el token, el administrador debe seleccionar explícitamente qué recursos puede acceder la integración. Por ejemplo, un Agente de IA diseñado para agendar citas solo requeriría scopes como calendars.write, calendars.readonly, contacts.readonly, minimizando la superficie de ataque en caso de compromiso del token.
- **Rotación de Credenciales:** La seguridad operativa exige la rotación periódica de credenciales. GHL implementa un mecanismo sofisticado de rotación que genera un nuevo token mientras mantiene el antiguo válido por una ventana de 7 días. Esto es crítico para arquitecturas de alta disponibilidad, permitiendo actualizar las variables de entorno en las *Edge Functions* de Supabase sin interrupción del servicio.
- **Inmutabilidad y Almacenamiento:** El token se muestra una única vez en la interfaz de usuario (UI). Esto obliga a implementar prácticas de *Secret Management* desde el día cero. En el contexto de Supabase, estos tokens deben almacenarse exclusivamente en el *Vault* cifrado o como secretos de entorno en el servicio de Edge Functions, nunca hardcodeados en el código fuente.

2.1.2 Implementación en las Cabeceras HTTP

Para interactuar con la API V2, cada petición HTTP debe construirse meticulosamente. La

documentación especifica que el token debe enviarse en el encabezado Authorization con el prefijo Bearer. Adicionalmente, y de importancia crítica, es la inclusión del encabezado Version. La omisión de este último puede resultar en respuestas con estructuras de datos legadas (V1), rompiendo los esquemas de deserialización JSON esperados.

Estructura de Headers Obligatoria:

```
Authorization: Bearer <PRIVATE_INTEGRATION_TOKEN>
Version: 2021-07-28
Accept: application/json
Content-Type: application/json
```

2.2 Autenticación Externa (External Auth) para Aplicaciones Marketplace

Si el objetivo es distribuir el Agente de IA como una aplicación en el Marketplace de GHL, la estrategia cambia hacia OAuth 2.0 y *External Authentication*. Este mecanismo permite validar la identidad del usuario de GHL contra nuestro sistema (Supabase) antes de completar la instalación.

El flujo de *External Auth* inyecta parámetros críticos durante la instalación: companyId, locationId, y userId. Estos identificadores son la base para implementar la *Multi-tenancy* (múltiples inquilinos) en la base de datos.

- **Company ID:** Identifica a la Agencia. Fundamental para políticas de segregación de datos a nivel macro.
- **Location ID:** Identifica la sub-cuenta específica. Este será el tenant_id principal en la mayoría de las tablas de Supabase.

2.3 Diseño de Seguridad en Supabase (Row Level Security)

Dado que estamos replicando datos sensibles de múltiples ubicaciones, confiar únicamente en la lógica de la aplicación es insuficiente. Supabase (PostgreSQL) ofrece *Row Level Security* (RLS), que permite definir políticas de acceso a nivel de fila.

Estrategia RLS Propuesta:

1. **Contexto de Ejecución:** Las *Edge Functions* que procesan webhooks operarán con la service_role_key (superusuario) para escribir datos sin restricciones.
2. **Acceso de Cliente (Agente IA):** Si el agente accede vía cliente REST/GraphQL, se debe generar un JWT que contenga el location_id en sus *claims*.
3. **Política SQL:**

```
CREATE POLICY "Aislar datos por Location" ON public.ghl_contacts
USING (location_id = auth.jwt() ->> 'location_id');
```

Esta política asegura que, incluso si hay un error en la lógica de consulta del agente, este nunca podrá acceder a contactos de una ubicación diferente a la que está autorizado.

3. Modelado de Datos Exhaustivo: Tablas Maestras y Esquemas CRUD

El diseño del esquema de base de datos es la tarea central de esta arquitectura. No basta con copiar los campos del JSON; debemos interpretar los tipos de datos, normalizar relaciones donde sea beneficioso para la integridad referencial y utilizar tipos semi-estructurados (JSONB) donde la flexibilidad de GHL lo demande. A continuación, se presenta un análisis detallado entidad por entidad, basado en los esquemas de respuesta JSON documentados.

3.1 Entidad Raíz: Business (Locations)

La entidad Location (referida a veces como Business en la API V1, pero Location en V2) es el contenedor padre de todos los datos operativos.
Análisis del Esquema JSON : La respuesta de la API para GET /businesses/{id} o /locations/{id} devuelve un objeto rico en detalles de configuración.

- **Identificadores:** id (string alfanumérico).
- **Datos Demográficos:** name, phone, email, website, address, city, state, postalCode, country.
- **Configuración Técnica:** timezone (crítico para la programación de citas), settings (objeto anidado con configuraciones diversas).

Diseño de Tabla en Supabase (ghl_locations): Se recomienda almacenar los campos de dirección y contacto como columnas explícitas para facilitar consultas geoespaciales o de filtrado, mientras que las configuraciones variables se mantienen en JSONB.

Columna	Tipo PostgreSQL	Descripción y Racional
id	text (PK)	ID original de GHL. No usar UUID generado; usar el ID de GHL para integridad.
name	text	Nombre comercial de la ubicación. Indexado para búsquedas.
email	text	Correo de contacto principal.
phone	text	Teléfono principal. Normalizar formato si es posible (E.164).
address_data	jsonb	Objeto conteniendo calle, ciudad, estado, CP, país. Permite consultas directas: address_data->>'city'.
timezone	text	Zona horaria (e.g., "America/New_York"). Vital para convertir timestamps UTC.
settings	jsonb	Almacena configuraciones crudas devueltas por la API para uso futuro.
api_key	text (Encrypted)	(Opcional) Si se almacenan keys legadas, usar columnas cifradas de Supabase Vault.
created_at	timestamptz	Timestamp de creación en nuestra DB.
last_sync	timestamptz	Marca de tiempo de la última sincronización exitosa.

3.2 Entidad Core: Contacts (Contactos)

El contacto es la entidad más dinámica y compleja. La API V2 permite una personalización extrema a través de customFields, lo que presenta un desafío de modelado.

Análisis del Esquema JSON :

- **Campos Estándar:** id, firstName, lastName, email, phone, dateOfBirth, tags (array de strings), dnd (Do Not Disturb, booleano), type (lead/customer).
- **Campos Complejos:**
  - customFields: Un array de objetos, donde cada objeto tiene id, key y value. El value puede ser string, array (para checkboxes) u objeto (para archivos adjuntos).
  - tags: Un array simple de strings ["tag1", "tag2"].
  - followers: Array de IDs de usuarios asignados.

Estrategia de Modelado Híbrido: Intentar normalizar customFields en una tabla EAV (Entity-Attribute-Value) tradicional (Tabla Attributes, Tabla Values) suele resultar en consultas SQL extremadamente lentas y complejas. La potencia de PostgreSQL en Supabase permite una estrategia híbrida: columnas para campos fijos y JSONB para campos dinámicos.

Diseño de Tabla en Supabase (ghl_contacts):

```
CREATE TABLE public.ghl_contacts (
    id text PRIMARY KEY, -- GHL Contact ID
    location_id text NOT NULL REFERENCES public.ghl_locations(id) ON
DELETE CASCADE,
    first_name text,
    last_name text,
    email text,
    phone text, -- Recomendar índice UNIQUE combinado con location_id
    date_of_birth date,
    tags text, -- Array nativo de Postgres. Permite operadores: WHERE
tags @> '{vip}'
    type text, -- 'lead', 'customer', etc.
    dnd boolean DEFAULT false,
    assigned_to text, -- ID del usuario principal responsable
    custom_fields jsonb DEFAULT ''::jsonb, -- Almacena el array
original de customFields
    custom_attributes jsonb DEFAULT '{}'::jsonb, -- Versión aplanada {
"key": "value" } para consultas fáciles
    source text, -- Fuente de atribución (e.g., "Facebook Lead Form")
    date_added timestamptz,
    date_updated timestamptz,
    raw_data jsonb, -- Payload completo para depuración
    search_vector tsvector GENERATED ALWAYS AS (
        setweight(to_tsvector('spanish', coalesce(first_name, '')),
'A') ||
        setweight(to_tsvector('spanish', coalesce(last_name, '')),
'A') ||
        setweight(to_tsvector('spanish', coalesce(email, '')), 'B')
    ) STORED
);
```

```
-- Índices Estratégicos
CREATE INDEX idx_contacts_location ON
public.ghl_contacts(location_id);
CREATE INDEX idx_contacts_email ON public.ghl_contacts(email);
CREATE INDEX idx_contacts_tags ON public.ghl_contacts USING GIN(tags);
CREATE INDEX idx_contacts_custom_attr ON public.ghl_contacts USING
GIN(custom_attributes);
CREATE INDEX idx_contacts_search ON public.ghl_contacts USING
GIN(search_vector);
```

Nota sobre custom_attributes: Se recomienda una transformación en la *Edge Function* de ingestión que convierta el array de customFields [{key: "color", value: "red"}] a un objeto plano {"color": "red"}. Esto simplifica drásticamente las consultas SQL dentro de Supabase (e.g., custom_attributes->>'color' = 'red').

3.3 Entidad Transaccional: Opportunities (Oportunidades)

Las oportunidades representan el estado del negocio en el embudo de ventas. Para análisis financiero y de rendimiento, la precisión de estos datos es vital.

Análisis del Esquema JSON:

- Dependencias fuertes: pipelineId y pipelineStageId.
- Estado: status (open, won, lost, abandoned).
- Valor: monetaryValue (número).

Diseño de Tabla en Supabase (ghl_opportunities): Es crucial modelar también los Pipelines y Stages para poder realizar joins y obtener los nombres legibles de las etapas, no solo sus IDs opacos.

```
CREATE TABLE public.ghl_opportunities (
    id text PRIMARY KEY,
    location_id text NOT NULL REFERENCES public.ghl_locations(id),
    contact_id text NOT NULL REFERENCES public.ghl_contacts(id),
    name text,
    status text CHECK (status IN ('open', 'won', 'lost', 'abandoned',
'all')),
    pipeline_id text NOT NULL,
    pipeline_stage_id text NOT NULL,
    monetary_value numeric(15, 2), -- Alta precisión para moneda
    assigned_to text,
    source text,
    loss_reason text, -- Capturar razón de pérdida si existe
    created_at timestamptz,
    updated_at timestamptz,
    CONSTRAINT fk_pipeline_stage FOREIGN KEY (pipeline_id,
pipeline_stage_id)
        REFERENCES public.ghl_pipeline_stages(pipeline_id, id) --
Requiere tabla de stages
);
```

3.4 Entidad de Automatización: Workflows (Flujos de Trabajo)

Los workflows son el motor lógico de GHL. Un Agente de IA necesita saber en qué flujos está activo un contacto para no interferir o duplicar acciones.

Diseño de Tabla en Supabase (ghl_workflows):

- id: ID del Workflow.
- name: Nombre descriptivo.
- status: Estado de publicación (draft, published).
- trigger_types: Array de tipos de disparadores (útil para análisis).

Adicionalmente, se recomienda una tabla ghl_contact_workflow_history para registrar las ejecuciones. Aunque GHL no envía webhooks para *cada paso* de un workflow, sí podemos inferir actividad o registrar cuando un workflow añade una etiqueta específica.

3.5 Entidad de Comunicación: Conversations & Messages (Contexto Crítico para IA)

Para un Agente de IA, el historial de conversación es el activo más valioso. GHL estructura esto en "Conversations" (contenedores) y "Messages" (items individuales).

Análisis del Esquema JSON:

- **Conversations:** id, contactId, unreadCount, lastMessageBody, lastMessageType (SMS, Email, WhatsApp, Call).
- **Messages:** id, conversationId, body (contenido), messageType, direction (inbound/outbound), status (delivered, failed, read), attachments (lista de URLs).

Diseño de Tablas en Supabase:

```
CREATE TABLE public.ghl_conversations (
    id text PRIMARY KEY,
    location_id text REFERENCES public.ghl_locations(id),
    contact_id text REFERENCES public.ghl_contacts(id),
    type text, -- Canal principal
    unread_count integer DEFAULT 0,
    last_message_body text, -- Cache para visualización rápida
    last_activity_at timestamptz,
    assigned_to text, -- Agente humano responsable
    is_archived boolean DEFAULT false
);

CREATE TABLE public.ghl_messages (
    id text PRIMARY KEY,
    conversation_id text NOT NULL REFERENCES
public.ghl_conversations(id) ON DELETE CASCADE,
    location_id text REFERENCES public.ghl_locations(id),
    contact_id text REFERENCES public.ghl_contacts(id),
    body text,
    message_type text, -- 'SMS', 'EMAIL', 'WHATSAPP', 'GMB', 'CALL'
    direction text CHECK (direction IN ('inbound', 'outbound')),
```

```
    status text,
    content_type text DEFAULT 'text/plain',
    attachments jsonb DEFAULT ''::jsonb, -- URLs de imágenes/docs
    meta_data jsonb, -- Para guardar headers de email, subject, call
duration
    created_at timestamptz NOT NULL
);

-- Índice compuesto para recuperar historial ordenado rápidamente
CREATE INDEX idx_messages_conversation_date ON
public.ghl_messages(conversation_id, created_at DESC);
```

3.6 Entidades de Facturación: Invoices & Products

Para agentes con capacidades transaccionales (ventas, soporte de facturación), es necesario replicar el esquema de Invoices.
Puntos Clave:

- **Polimorfismo:** altId y altType en el JSON de Invoice indican si pertenece a una location o agency. En nuestro caso, filtraremos mayormente por location.
- **Items:** Los items de la factura (items) contienen snapshots de los productos al momento de la venta.
- **Relación:** contactId conecta la factura con el cliente.

3.7 Entidades de Calendario: Appointments

La gestión de tiempo es fundamental. **Campos Clave:** startTime, endTime, status (confirmed, cancelled, noshow), calendarId. **Importancia de Timezone:** Siempre almacenar en UTC en la base de datos (timestamptz), pero guardar la startTime original y el timeZone del calendario para reconstruir la hora local del usuario correctamente.

4. Estrategia de Sincronización Optimizada con Webhooks

Una base de datos estática pierde valor rápidamente. La arquitectura propuesta utiliza una estrategia de sincronización reactiva "Event-First".

4.1 Mapeo de Eventos de Webhook a Acciones CRUD

La documentación de webhooks de GHL lista eventos granulares. Debemos mapear estos eventos a operaciones SQL específicas en nuestras tablas maestras.

Evento Webhook GHL	Entidad Afectada	Operación SQL	Notas Técnicas
ContactCreate	ghl_contacts	INSERT	Verificar si el ID ya existe (race conditions).
ContactUpdate	ghl_contacts	UPDATE	Actualizar last_activity y campos modificados.

Evento Webhook GHL	Entidad Afectada	Operación SQL	Notas Técnicas
ContactDelete	ghl_contacts	DELETE / Soft Delete	Se recomienda Soft Delete (is_deleted=true).
OpportunityCreate	ghl_opportunities	INSERT	Validar existencia de pipeline_id.
OpportunityUpdate	ghl_opportunities	UPDATE	Crítico: actualizar stage_id y status.
InboundMessage	ghl_messages	INSERT	Crear conversación si no existe.
OutboundMessage	ghl_messages	INSERT	Registrar respuesta del agente o automatización.
InvoiceCreate	ghl_invoices	INSERT	Iniciar seguimiento de pago.
AppointmentUpdate	ghl_appointments	UPSERT	Manejar reagendamientos y cancelaciones.

4.2 Análisis de Discrepancias en Payloads

Una observación crítica derivada del análisis de los snippets es que el payload del webhook **no siempre es idéntico** al objeto devuelto por la API REST.

- **Ejemplo ContactCreate:** El payload del webhook a menudo aplana ciertos campos o estructura los customFields de manera diferente a la API GET /contacts/{id}.
- **Solución (Patrón Fetch-on-Webhook):** Para garantizar la integridad absoluta de los datos, una práctica recomendada en sistemas críticos es utilizar el webhook meramente como una "señal de invalidación".
    1. Recibir Webhook ContactUpdate con ID 123.
    2. Edge Function recibe la señal.
    3. Edge Function llama inmediatamente a la API GET /contacts/123 usando el Private Token.
    4. El JSON completo y canónico de la API se usa para el UPSERT en Supabase. *Este enfoque consume 1 crédito de API por cambio, pero garantiza que la estructura de datos sea 100% consistente con la especificación oficial, evitando errores por payloads de webhook incompletos o indocumentados.*

4.3 Diseño de Ingestión con Supabase Edge Functions

Las *Edge Functions* (Deno) actúan como el middleware de integración, validando y transformando los datos antes de tocar la base de datos.

Flujo de Proceso Detallado:

1. **Endpoint Único o Dedicado:** Se puede configurar un endpoint https://<ref>.supabase.co/functions/v1/ghl-hooks que despache internamente según el type del evento.
2. **Validación de Seguridad:** Aunque GHL no firma los requests con un secreto compartido estándar en todos los casos documentados, se debe incluir un token secreto en la URL del webhook configurada en GHL (?secret=MI_TOKEN_SECRETO) y validarlo en la

función.

3. **Idempotencia:** GHL podría enviar el mismo evento dos veces. La Edge Function debe calcular un hash del payload o usar el messageId del evento (si existe) para evitar procesar duplicados, o confiar en la operación ON CONFLICT DO UPDATE de SQL para manejarlo a nivel de base de datos.
4. **Transformación de Datos:**
    - Parseo de fechas ISO 8601.
    - Aplanamiento de customFields para la columna custom_attributes JSONB.
    - Conversión de precios (GHL a veces usa centavos, a veces unidades flotantes; estandarizar a decimal).

Ejemplo de Código para Edge Function (TypeScript/Deno):

```
import { serve } from "https://deno.land/std@0.168.0/http/server.ts"
import { createClient } from "https://esm.sh/@supabase/supabase-js@2"

serve(async (req) => {
  // 1. Verificación de Secreto
  const url = new URL(req.url)
  if (url.searchParams.get('secret')!==
Deno.env.get('WEBHOOK_SECRET')) {
    return new Response("Unauthorized", { status: 401 })
  }

  const payload = await req.json()
  const supabase = createClient(
    Deno.env.get('SUPABASE_URL')?? '',
    Deno.env.get('SUPABASE_SERVICE_ROLE_KEY')?? ''
  )

  // 2. Despacho por Tipo de Evento
  try {
    if (payload.type === 'ContactCreate' |

| payload.type === 'ContactUpdate') {
      // Mapeo de campos del payload a columnas de DB
      // Nota: GHL envía 'id' en el root o en 'contact_id'
dependiendo del evento
      const contactData = {
        id: payload.id |

| payload.contact_id,
        location_id: payload.locationId,
        email: payload.email,
        first_name: payload.firstName,
        last_name: payload.lastName,
        tags: payload.tags, // Asumiendo array de strings
        custom_fields: payload.customFields,
        // Lógica de transformación para custom_attributes
        updated_at: new Date().toISOString()
```

```
        }

        // 3. Operación UPSERT
        const { error } = await supabase
          .from('ghl_contacts')
          .upsert(contactData, { onConflict: 'id' })

        if (error) throw error
      }

      //... Manejo de otros eventos (Opportunity, Message, etc.)

      return new Response(JSON.stringify({ success: true }), {
        headers: { "Content-Type": "application/json" }
      })

    } catch (err) {
      console.error("Error processing webhook:", err)
      return new Response(JSON.stringify({ error: err.message }), {
status: 500 })
    }
})
```

5. Optimización para Agentes de IA: RAG y Contexto Vectorial

El objetivo final de esta arquitectura es potenciar Agentes de IA. Un simple CRUD no es suficiente; necesitamos preparar los datos para *Retrieval-Augmented Generation* (RAG).

5.1 Extensión pgvector en Supabase

Supabase soporta pgvector nativamente. Esto permite almacenar *embeddings* (representaciones vectoriales semánticas) de los datos de GHL junto con los datos relacionales.

Diseño de Tabla de Embeddings (ghl_embeddings): En lugar de vectorizar cada columna, creamos "documentos de contexto" sintéticos.

```
CREATE EXTENSION IF NOT EXISTS vector;

CREATE TABLE public.ghl_knowledge_vectors (
    id uuid DEFAULT gen_random_uuid() PRIMARY KEY,
    location_id text NOT NULL,
    entity_type text, -- 'contact', 'conversation', 'product'
    entity_id text NOT NULL, -- FK polimórfica
    content_chunk text, -- Texto legible: "El contacto Juan Perez
compró Producto X..."
    embedding vector(1536), -- Dimensión para OpenAI
```

```
text-embedding-3-small
    created_at timestamptz DEFAULT now()
);

CREATE INDEX on public.ghl_knowledge_vectors using ivfflat (embedding
vector_cosine_ops);
```

5.2 Estrategia de Vectorización Reactiva

Aprovechando los triggers de base de datos de Supabase, podemos automatizar la generación de embeddings.

1. **Trigger:** Cuando se inserta un nuevo mensaje en ghl_messages o se actualiza una nota en un contacto.
2. **Edge Function:** Un Database Webhook invoca una función que:
    - Toma el texto nuevo.
    - Llama a la API de Embeddings (OpenAI/Cohere).
    - Inserta el vector resultante en ghl_knowledge_vectors.
3. **Resultado:** El Agente de IA puede realizar búsquedas semánticas ("Buscar clientes que hayan preguntado por descuentos en el último mes") consultando directamente la tabla de vectores, cruzando resultados con los datos estructurados de ghl_contacts.

6. Guía de Implementación: SDKs y Herramientas

Aunque la integración directa vía REST es posible, el uso de SDKs tipados acelera el desarrollo y reduce errores.

6.1 SDK Oficial vs. Comunitario

El análisis de recursos revela la existencia de:

- **SDK Oficial (@gohighlevel/api-client):** Proporcionado por GHL. Ofrece soporte oficial, manejo de autenticación y tipos básicos. Es la opción recomendada para producción.
- **SDKs Comunitarios:** A menudo ofrecen características experimentales o abstracciones de más alto nivel, pero con riesgo de desactualización.

Recomendación: Utilizar el SDK oficial para las operaciones de escritura dentro de las Edge Functions de Supabase para garantizar compatibilidad con los cambios de la API.

6.2 Manejo de Errores y Rate Limiting

El código de integración debe ser robusto frente a fallos.

- **Headers de Rate Limit:** GHL devuelve headers X-RateLimit-Daily-Remaining y X-RateLimit-Interval-Milliseconds. La lógica de ingestión debe leer estos headers y, si el remanente es bajo, pausar las operaciones no críticas o encolarlas en una tabla job_queue en Supabase para procesamiento diferido.
- **Backoff Exponencial:** En caso de errores 429 (Too Many Requests), implementar reintentos con espera exponencial.

7. Conclusiones y Recomendaciones Finales

La arquitectura presentada transforma a GoHighLevel de un simple SaaS a una plataforma de datos programable. Al centralizar los datos en Supabase, diseñando esquemas que respetan la complejidad de los objetos JSON de GHL y utilizando una sincronización reactiva vía Webhooks, se superan las barreras tradicionales de integración.

Puntos Clave para el Éxito:

1. **Priorizar Private Integration Tokens** para la seguridad del backend.
2. **Implementar RLS en Supabase** desde el inicio para garantizar la seguridad multi-tenant.
3. **Adoptar un enfoque híbrido de modelado** (Columnas SQL + JSONB) para manejar la flexibilidad de GHL sin sacrificar rendimiento.
4. **Integrar Vectorización (RAG)** directamente en el flujo de ingestión de datos para habilitar capacidades de IA verdaderamente contextuales.

Esta infraestructura no solo cumple con los requisitos del concurso de Agentes de IA, sino que establece una base empresarial para escalar operaciones de automatización complejas, análisis de datos y experiencias de cliente hiper-personalizadas.

Fuentes citadas

1. HighLevel API, https://help.gohighlevel.com/support/solutions/articles/48001060529-highlevel-api 2. Private Integrations | HighLevel API - GoHighLevel Marketplace, https://marketplace.gohighlevel.com/docs/Authorization/PrivateIntegrationsToken 3. Update Opportunity | HighLevel API - GoHighLevel Marketplace, https://marketplace.gohighlevel.com/docs/ghl/opportunities/update-opportunity/index.html 4. Get Business | HighLevel API - GoHighLevel Marketplace, https://marketplace.gohighlevel.com/docs/ghl/businesses/get-business/index.html 5. Create Contact | HighLevel API - GoHighLevel Marketplace, https://marketplace.gohighlevel.com/docs/ghl/contacts/create-contact/index.html 6. Get Workflow | HighLevel API - GoHighLevel Marketplace, https://marketplace.gohighlevel.com/docs/ghl/workflows/get-workflow/index.html 7. Create Invoice | HighLevel API - GoHighLevel Marketplace, https://marketplace.gohighlevel.com/docs/ghl/invoices/create-invoice/index.html 8. Search Conversations | HighLevel API - GoHighLevel Marketplace, https://marketplace.gohighlevel.com/docs/ghl/conversations/search-conversation/index.html 9. Create Business | HighLevel API, https://marketplace.gohighlevel.com/docs/ghl/businesses/create-business/index.html 10. Get Contacts | HighLevel API - GoHighLevel Marketplace, https://marketplace.gohighlevel.com/docs/ghl/contacts/get-contacts/index.html 11. Get Opportunity | HighLevel API - GoHighLevel Marketplace, https://marketplace.gohighlevel.com/docs/ghl/opportunities/get-opportunity/index.html 12. Send a new message | HighLevel API - GoHighLevel Marketplace, https://marketplace.gohighlevel.com/docs/ghl/conversations/send-a-new-message/index.html 13. Get transcription by Message ID | HighLevel API - GoHighLevel Marketplace, https://marketplace.gohighlevel.com/docs/ghl/conversations/get-message-transcription/index.html 14. Update invoice | HighLevel API - GoHighLevel Marketplace,

https://marketplace.gohighlevel.com/docs/ghl/invoices/update-invoice/index.html 15. Webhook | HighLevel API, https://marketplace.gohighlevel.com/docs/category/webhook/index.html 16. Webhook Integration Guide | HighLevel API - GoHighLevel Marketplace, https://marketplace.gohighlevel.com/docs/webhook/WebhookIntegrationGuide/index.html 17. LC Email | HighLevel API - GoHighLevel Marketplace, https://marketplace.gohighlevel.com/docs/webhook/LCEmailStats/index.html 18. gohighlevel/api-client - NPM, https://www.npmjs.com/package/@gohighlevel/api-client 19. CallBackCode/ghlSDK: A public Node.js compatible SDK for working with HighLevel's (GHL's) Version 2 API. - GitHub, https://github.com/CallBackCode/ghlSDK