GHL

- Integration vCX with GHL

Integration vCX with GHL

? Recommended Architecture

1. **Push vCX Conversations into GHL**

Use the https://highlevel.stoplight.io/docs/integrations/0443d7d1a4bd0-overview to:

- Create or update a **Contact** using `clientId` as the unique identifier
- Create a **Conversation** under that contact
- Add each message as a **Message** object inside the conversation

> All of this is supported via `POST /conversations/{id}/messages` and related endpoints .

2. **Pull GHL Replies into vCX**

Set up a **webhook subscription** in GHL to listen for:

- `message.incoming`
- `conversation.updated`

These webhooks will fire whenever a GHL user (or bot) replies. You can map the GHL `conversationId` to your `conversationId` using metadata or a lookup table.

> OAuth 2.0 is now required for all new integrations, so you'll need to register your app in GHL's Developer Portal .

? Key GHL API Docs You'll Need

Table
Copy

Task	Endpoint	Notes
Create/Update Contact	`POST /contacts`	Use `clientId` as external ID
Create Conversation	`POST /conversations`	Link to contact
Send Message	`POST /conversations/{id}/messages`	Includes text, type, timestamp
Listen for Replies	Webhook: `message.incoming`	Use to sync back to vCX

All endpoints are documented at:

📖 https://highlevel.stoplight.io/docs/integrations

vCX ? GoHighLevel – Two-Way Chat Sync (Node.js)

0. Prerequisites

- Node ≥ 18
- A GHL developer account (https://developers.gohighlevel.com)
- Your app registered in the portal with scopes: `contacts.write conversations.write conversations.read locations.read`
- Redirect URI set to `https://yourdomain.com/auth/callback`
- Environment variables:

```
GHL_CLIENT_ID=xxxxxxxx
GHL_CLIENT_SECRET=xxxxxxxx
GHL_REDIRECT_URI=https://yourdomain.com/auth/callback
```

1. Install dependencies

`package.json (excerpt)`

```
{
  "type": "module",
  "dependencies": {
    "axios": "^1.6.0",
    "dotenv": "^16.3.1",
    "express": "^4.18.2"
  }
}
```

```
npm install
```

2. Minimal Express server skeleton

server.js (top)

```
import 'dotenv/config';
import express from 'express';
import axios from 'axios';
import crypto from 'crypto';
const app = express();
app.use(express.json());

const PORT = process.env.PORT || 3000;

/* --- In-memory maps for demo purposes --- */
const tokenStore = new Map();          // locationId -> {access_token, refresh_token, expires_
const conversationMap = new Map();     // vcxConversationId -> ghlConversationId

app.listen(PORT, () => console.log(`Listening on :${PORT}`));
```

3. OAuth 2.0 – Authorization URL

GET /install

```
app.get('/install', (req, res) => {
  const state = crypto.randomUUID();
  const url = `https://marketplace.gohighlevel.com/oauth/chooselocation?response_type=code&cli
  res.redirect(url);
});
```

4. OAuth 2.0 – Exchange code for tokens

GET /auth/callback

```
app.get('/auth/callback', async (req, res) => {
  const { code, locationId } = req.query;
  const { data } = await axios.post('https://services.leadconnectorhq.com/oauth/token', {
    client_id: process.env.GHL_CLIENT_ID,
    client_secret: process.env.GHL_CLIENT_SECRET,
    grant_type: 'authorization_code',
    code,
    redirect_uri: process.env.GHL_REDIRECT_URI
  });
  tokenStore.set(locationId, {
    access_token: data.access_token,
    refresh_token: data.refresh_token,
```

```
    expires_at: Date.now() + data.expires_in * 1000
  });
  res.send(`Sub-account ${locationId} connected.`);
});
```

5. Helper – get valid access token (auto-refresh)

```
async function getToken(locationId) {
  let t = tokenStore.get(locationId);
  if (!t) throw new Error('Location not authorized');

  if (Date.now() > t.expires_at - 60_000) {
    const { data } = await axios.post('https://services.leadconnectorhq.com/oauth/token', {
      client_id: process.env.GHL_CLIENT_ID,
      client_secret: process.env.GHL_CLIENT_SECRET,
      grant_type: 'refresh_token',
      refresh_token: t.refresh_token
    });
    t = {
      access_token: data.access_token,
      refresh_token: data.refresh_token,
      expires_at: Date.now() + data.expires_in * 1000
    };
    tokenStore.set(locationId, t);
  }
  return t.access_token;
}
```

6. Upsert Contact (by vCX clientId)

```
POST /contact
```

```
app.post('/contact', async (req, res) => {
  const { locationId, clientId, email, phone, name } = req.body;
  const token = await getToken(locationId);

  // Search by externalId first
  const search = await axios.get(`https://rest.gohighlevel.com/v1/contacts/?query=${clientId}&
    headers: { Authorization: `Bearer ${token}` }
  });
  let contactId = search.data.contacts?.[0]?.id;

  if (!contactId) {
    const { data } = await axios.post('https://rest.gohighlevel.com/v1/contacts/', {
      locationId,
      name,
      email,
```

```
      phone,
      source: 'vCX',
      tags: ['vCX'],
      customFields: [{ id: 'clientId', value: clientId }]
    }, { headers: { Authorization: `Bearer ${token}` } });
    contactId = data.contact.id;
  }
  res.json({ contactId });
});
```

7. Create Conversation & Push Messages

```
POST /push-message
```

```
app.post('/push-message', async (req, res) => {
  const { locationId, clientId, vcxConversationId, fromUser, body, timestamp } = req.body;
  const token = await getToken(locationId);

  // 1. Ensure contact
  const { contactId } = (await axios.post(`http://localhost:${PORT}/contact`, {
    locationId, clientId, email: `${clientId}@example.com`, name: clientId
  })).data;

  // 2. Ensure conversation
  let ghlConvId = conversationMap.get(vcxConversationId);
  if (!ghlConvId) {
    const { data } = await axios.post('https://rest.gohighlevel.com/v1/conversations/', {
      locationId,
      contactId,
      type: 'chat'
    }, { headers: { Authorization: `Bearer ${token}` } });
    ghlConvId = data.conversation.id;
    conversationMap.set(vcxConversationId, ghlConvId);
  }

  // 3. Push message
  await axios.post(`https://rest.gohighlevel.com/v1/conversations/${ghlConvId}/messages`, {
    type: fromUser ? 'Inbound' : 'Outbound',
    message: body,
    dateAdded: new Date(timestamp).toISOString()
  }, { headers: { Authorization: `Bearer ${token}` } });

  res.sendStatus(200);
});
```

8. Receive GHL Replies via Webhook

```
POST /webhook
```

```
app.post('/webhook', (req, res) => {
  const { type, locationId, conversationId, message } = req.body;

  if (type !== 'message.incoming') return res.sendStatus(200);

  // Reverse lookup
  let vcxConvId;
  for (const [vId, gId] of conversationMap.entries()) {
    if (gId === conversationId) vcxConvId = vId;
  }
  if (!vcxConvId) return res.sendStatus(200);

  // TODO: forward to vCX backend
  console.log('Forward to vCX:', { vcxConversationId: vcxConvId, fromUser: false, body: messag

  res.sendStatus(200);
});
```

Register this URL in GHL → Settings → API → Webhooks.

9. Quick test with cURL

```
# 1. Start your server
node server.js

# 2. Install the app (open in browser)
open http://localhost:3000/install

# 3. Push a message
curl -X POST http://localhost:3000/push-message \
  -H "Content-Type: application/json" \
  -d '{"locationId":"LOC_ID","clientId":"c_abc123","vcxConversationId":"conv_456","fromUser":t
```

10. Production checklist

- Use persistent storage (Redis/Postgres) instead of in-memory maps.
- Verify webhook signatures (GHL sends headers `X-Signature`).
- Rate-limit token refresh.
- Handle pagination when searching contacts.
- Wrap axios calls in retries with exponential backoff.

Last updated 2024-07-20