

Short answer. That folder tree is for a filesystem, not something you “make inside” n8n Cloud. It looks like a local or server repo meant to run n8n with Docker Compose and keep your agent files beside it. You cannot mount folders or run docker-compose inside n8n Cloud.

You have two clean paths. Pick one.

Option A. Keep using n8n Cloud

Store the files outside n8n and read them in your workflow.

Step 1. Put the files somewhere HTTP-accessible.

Use GitHub, S3, Google Drive, or similar. Keep the same folders: `templates_email`, `vendors_company`, `docs`. Give your collaborator write access.

Step 2. Move the agent config out of YAML.

Either convert `agent.yaml` to JSON and store it in n8n as Project Variables or in a Data Store.

Your collaborator can then read it at runtime.

(Path: Project settings. Variables. Add a JSON variable like `AGENT_CONFIG`. Or create a Data Store named `agent_config` and insert one record with the config.)

Step 3. Wire the workflow.

Use HTTP Request or the service’s node (S3, Google Drive, GitHub) to fetch templates. Parse your config variable. Feed OpenAI with your existing OpenAI credential.

Step 4. Permissions.

Invite your collaborator as Workspace Member, then make them Project Editor. Share the OpenAI credential with the project. They can build the agent without touching billing or workspace-wide settings.

This keeps you on Cloud and avoids server work. The tradeoff is your templates live in a cloud drive or repo, not a local folder.

Option B. Self-host n8n with that exact layout

If the developer expects that structure, run n8n via Docker and mount those folders.

Prereqs. Install Docker Desktop. Pick a machine that can stay on if others need access.

Step 1. Create a working folder on your machine, for example `N8N-LOCAL`. Inside it create:

- `agents/demo-agent`
- `templates_email`

- vendors_company
 - docs
 - n8n_data
- Put their agent.yaml into agents/demo-agent.

Step 2. Create docker-compose.yml in N8N-LOCAL with this content:

```
version: "3.8"
services:
  n8n:
    image: docker.n8n.io/n8nio/n8n:latest
    ports:
      - "5678:5678"
    environment:
      - TZ=America/Phoenix
      - N8N_HOST=localhost
      - N8N_PORT=5678
      - WEBHOOK_URL=http://localhost:5678/
      # Generate a long random string and keep it safe
      - N8N_ENCRYPTION_KEY=replace_with_a_long_random_string
    volumes:
      - ./n8n_data:/home/node/.n8n
      - ./agents:/workspace/agents
      - ./templates_email:/workspace/templates_email
      - ./vendors_company:/workspace/vendors_company
      - ./docs:/workspace/docs
    restart: unless-stopped
```

Step 3. Start it.

- Open a terminal in N8N-LOCAL.
- Run: `docker compose up -d`

Step 4. Open n8n.

Visit <http://localhost:5678>. Finish the setup. Create your OpenAI credential. Build the workflow. When a node needs a template, point to the mounted paths, for example
/workspace/templates_email/....

Step 5. Let your collaborator in.

Either give them access to this machine or put this on a cloud VM with a real domain and SSL. If you want the simplest collaboration, go back to Option A.

What I recommend

If the collaborator will “build the agent on the page” inside your n8n Cloud project, use Option A. Ask them to push those folders to a repo or bucket. Store the config in Variables or a Data Store. Share your OpenAI credential to the project. They will be productive fast, and you avoid server maintenance.

If they specifically asked for docker-compose with that folder map, use Option B. I can tailor the compose file to your domain and SSL, or sketch the first workflow that reads agent.yaml and a template file end to end.

You said:

i am looking for the most simple and effective route, where i will be able to easily find it and also how it will not break the bank. is github the best priced and user friendly option?

ChatGPT said:

Thought for 40s

Here's the straight answer.

GitHub is the cheapest and developer-friendly choice. Private repos are free, versioning is built in, and your dev will be faster there. For a non-coder, Google Drive is the most familiar and "I can find it later" friendly. Both are basically free at your scale. If you want the absolute simplest setup for you plus the least friction for your collaborator, pick GitHub. If you hate repos, use Drive.

Quick tradeoff:

- GitHub. Free, clean history, easy rollbacks, devs love it. Slight learning curve for uploads and file paths.
- Google Drive. Drag and drop, easy to find things, fine for small files. API auth is a bit more fiddly and version control is weak.

Below are minimal, step-by-step setups for both. Pick one and run it.

Option 1. GitHub (recommended)

Purpose. Store the exact folder tree from the screenshot, let n8n read files at runtime.

Step 1. Create a private repo

Name it something like n8n-agent-assets. Keep it private.

Step 2. Create folders to match your dev's structure

agents/demo-agent

templates_email

vendors_company

docs

Upload agent.yaml into agents/demo-agent.

Step 3. Invite your collaborator

Repo Settings. Collaborators. Add their GitHub handle. Give Write access.

Step 4. Make a read-only token for n8n

GitHub. Settings. Developer settings. Fine-grained token. Scope: read access to contents for this single repo. Copy the token.

Step 5. Add the token to n8n

n8n. Project. Credentials. New. GitHub. Use access token. Save. Share this credential with the project.

Step 6. Read files in your workflow

- Use a GitHub node. “Get file content”.
- Owner: your GitHub username.
- Repo: n8n-agent-assets.
- Path example: agents/demo-agent/agent.yaml or templates_email/welcome.txt.
- Pipe the output into a YAML node to convert agent.yaml to JSON if needed.
- Use the text from templates_email as your message body inputs to OpenAI.

Cost. Free unless you start storing big binaries with Git LFS.

Option 2. Google Drive

Purpose. Same folders, but in Drive.

Step 1. Make a top-level folder

Name it n8n-agent-assets. Inside it create agents, templates_email, vendors_company, docs, then agents/demo-agent. Upload agent.yaml.

Step 2. Share with collaborator

Give them Editor.

Step 3. Add Google Drive credentials in n8n

Create a Google Drive OAuth credential. Follow the sign-in popup. Share the credential with the project.

Step 4. Read files in your workflow

Use the Google Drive node to locate by path or file name, then “Download file”. If you fetch agent.yaml, run it through a YAML node to convert to JSON.

Cost. Free up to your personal storage. If you use Google Workspace, it is the cost of your seat.

What not to do

- Do not store your OpenAI key in GitHub or Drive. Keep it in n8n as an OpenAI credential and share that credential with the project.

- Do not try to recreate that folder tree inside n8n Cloud. n8n does not have a native file system.

If you want me to keep it dead simple and future-proof, I would set up the GitHub route and wire one example workflow that reads agent.yaml and one template file end to end. After that, you can just drag new files into the repo web UI and everything keeps working.

You said:

he replied when I asked about it here is what he said: That URL (*.n8n.cloud) means you're running on n8n's managed cloud, not a self-hosted instance. On n8n Cloud you cannot directly mount local folders like /agents/..., which is a core part of the demo design (watcher indexing, per-agent folders, audit logs). Allow me to do on my local host then I will show you the demo n8n.io - AI workflow automation tool n8n is a free and source-available workflow automation tool

ChatGPT said:

You're both right. He's correct that n8n Cloud cannot mount a local folder tree like /agents/.... That demo layout is for a filesystem or a self-hosted n8n with Docker. You want the simplest and cheapest path. Here's the clean call:

Pick this: n8n Cloud + GitHub repo

You keep using n8n Cloud. Store the agent folders in a **private GitHub repo**. Use a **GitHub webhook** to mimic the "watcher," and log activity inside n8n. It is cheap, easy to find later, and your collaborator will be productive fast.

What you get

- Per-agent folders by path in the repo.
- "Watcher" via GitHub → n8n webhook on each push.
- Audit logs inside n8n Data Stores or a Google Sheet.
- Your existing OpenAI credential stays in n8n. Do not put keys in GitHub.

Do this now, step by step

Step 1. Create the repo

- Name: `n8n-agent-assets` (private).
- Folders:
 - `agents/demo-agent`
 - `templates_email`
 - `vendors_company`
 - `docs`
- Put `agent.yaml` in `agents/demo-agent`.

Step 2. Add your collaborator

- Repo Settings. Collaborators. Invite them. Give Write access.

Step 3. Make a read-only GitHub token for n8n

- GitHub Settings. Developer settings. Fine-grained token. Scope: read repository contents for this single repo. Copy the token.

Step 4. Add GitHub credentials in n8n

- n8n. Project. Credentials. New. GitHub. Paste token. Save. Share with the project.

Step 5. Build the “watcher” in n8n

- Create a workflow with a **Webhook** trigger.
- Copy the Production URL.
- In your GitHub repo: Settings. Webhooks. Add webhook.
Payload URL = your n8n webhook URL.
Content type = `application/json`.
Events = “Just the push event.”
- Back in n8n, add a **Function** node to read the push payload and list changed file paths from `head_commit.added, modified, removed`.
- Add a **GitHub** node to “Get file content” for each changed path you care about. Example paths:
`agents/demo-agent/agent.yaml`
`templates_email/welcome.txt`
- If you need the YAML as JSON, add a **YAML** node to convert it.

Step 6. Add audit logging

- Create a **Data Store** in n8n named `agent_audit`. Fields: `timestamp`, `actor`, `event`, `file_path`, `commit_id`.
- After each push, upsert one row per file you indexed. That gives you an in-n8n audit trail at zero extra cost.

That is it. Your collaborator can now “build on the page” in your Cloud project, referencing repo files by path, and you get push-triggered indexing without running a server.

Alternative if you need his exact demo

Self-host n8n on a small VPS with Docker Compose and mount those folders. That matches his design perfectly. It adds setup and ongoing maintenance. Cost is still modest, but it is more

moving parts. If you go this route, I can give you a minimal compose file and the exact volume mounts.

Recommendation

Accept his local demo so you can see behavior. Then implement the GitHub + webhook pattern above. It is the least cognitive load for you, costs basically nothing beyond what you already have, and it plays nicely with n8n Cloud. When you want to add more agents, you just add another folder in the repo and push.

You said:

but in the long run, if i will be having multiple KBs for agents and need to create more folders, the github route will be better than the structure he is proposing right? Because a KB for multifamily housing will have some different things than a KB for restaurants

ChatGPT said:

Thought for 21s

Yes. For your long game with multiple KBs and lots of folders, GitHub beats a mounted local folder setup on n8n Cloud.

Why.

- You stay on n8n Cloud. No servers to maintain, no Docker, no surprise bills.
- Private repos are free. Collaboration, history, and rollback are built in.
- Webhooks give you a clean “watcher” each time content changes.
- Scaling to more agents or industries is just adding folders or branches. No ops work.

The developer’s layout with local mounts only shines if you self-host n8n. That adds admin work and fragility. For you, not worth it.

Here is a simple repo pattern that will age well and keeps multifamily vs restaurant clean without duplication.

```
n8n-agent-assets/
  kb_shared/                # content used by all agents
    policies/
    glossary/
    templates_email/
  kb_industries/
    multifamily/
      templates_email/
      vendors/
      docs/
    config.yaml              # optional defaults for this vertical
```

```

restaurant/
  templates_email/
  vendors/
  docs/
  config.yaml
agents/
  renewalert-mf-agent/
    agent.yaml # points at shared + multifamily paths
  renewalert-restaurant-agent/
    agent.yaml # points at shared + restaurant paths

```

How each agent chooses the right KB

- In `agent.yaml`, list the KB folders it should read, in order. Example: first `kb_shared`, then `kb_industries/multifamily`. Your workflow reads those paths, merges configs, and the last one wins on conflicts.
- Email templates go in each industry's `templates_email`. If a template name exists in both shared and industry, use the industry one.

Sample `agent.yaml` idea

```

name: renewalert-mf-agent
kb_paths:
  - kb_shared
  - kb_industries/multifamily
templates:
  welcome: templates_email/welcome.txt
  renewal_nudge: templates_email/renewal_nudge.txt
vendor_list: vendors/vendors.csv

```

How this plugs into n8n Cloud, step by step

1. Put that folder structure in a private GitHub repo. Invite your dev with Write access.
2. Create a fine-grained GitHub token with read access to that repo. Save it as a GitHub credential in n8n and share it with the project.
3. Make a “watcher” workflow: GitHub webhook → parse changed file paths → for any path under `kb_*` or `agents/*/agent.yaml`, fetch content with the GitHub node → update your Data Store or cache.
4. In your agent workflow, load `agent.yaml`, convert YAML to JSON, resolve the `kb_paths`, then fetch templates or vendor files by path before calling OpenAI.
5. Keep all secrets in n8n Credentials. Never in the repo.

Cost reality

- GitHub private repos are free at your scale. LFS only matters for big binaries.
- n8n Cloud cost does not change for this pattern.
- Self-hosting to support mounted folders is cheap on paper, but your time and downtime risk are not.

Bottom line

You will be spinning up more agents and more KBs across industries. GitHub plus webhooks plus n8n Cloud gives you speed, control, and near-zero ops. The “local mounts” structure will box you into self-hosting and constant tinkering. Stick with GitHub.

You said:

i still need to clarify, even with webhooks and a KB on github, we will need the agent to know which KB to pull from on github routed through n8n right? I know for the MVP right now its not a big issue, we could combine KB data in one folder for now to show custom features by each industry in one agent. But down the road, is it an easy fix to switch? If for example, we had Bryten Real Estate Management as an account and they had specific SOPs and vendors they wanted in their KB, i would need to build that KB on Github and then in the agent's instructions make sure Bryten's agents knew to pull from their specific file on Github, via n8n. Does a webhook get us there or is that a problem? Again, thats a problem down the road, i just dont want to build something now that has to be changed massively later, rather build the foundation correctly knowing this is down the road.

ChatGPT said:

Thought for 32s

Short and clear: webhooks don't decide which KB to use. They just tell n8n “files changed.” Routing the agent to the right KB is your job in the workflow. If you design that routing once, switching later is painless.

Here is the foundation that scales to multifamily, restaurants, and Bryten-specific SOPs without rework.

The model to use

- **Mono-repo on GitHub** with a predictable folder scheme.
- **Account registry in n8n** that maps an account to its KB spec.
- **Resolver step in the workflow** that turns “which account is this?” into a list of folders to read, in priority order.
- **Webhook** from GitHub to n8n only for cache refresh or re-indexing, not for KB selection.

Folder scheme that won't paint you into a corner

```
n8n-agent-assets/  
  kb/shared/           # used by everyone  
  kb/industries/  
    multifamily/
```

```

    restaurant/
kb/accounts/
  bryten-real-estate/          # client-specific overrides
    templates_email/
    vendors/
    docs/
    config.yaml
agents/
  renewalert-generic/agent.yaml

```

Precedence rule. Account overrides industry. Industry overrides shared. If a file exists in multiple places, the most specific wins.

What the agent reads

Put a selector in `agent.yaml` that declares the search order with variables. Example:

```

name: renewalert-generic
kb_selector:
  repo: n8n-agent-assets
  branch: main
  paths:
    - kb/shared
    - kb/industries/{{industry}}
    - kb/accounts/{{account_slug}}
templates:
  renewal_nudge: templates_email/renewal_nudge.txt

```

Where `{{industry}}` and `{{account_slug}}` come from

Create a tiny **Account Registry** in n8n as a Data Store table:

- `account_slug` (bryten-real-estate)
- `industry` (multifamily)
- `kb_repo` (n8n-agent-assets)
- `kb_branch` (main)
- optional `extra_paths` (array)

At runtime you identify the account (domain, API key, or project id), look it up in the registry, and fill the placeholders. If no match, default to industry or shared.

Workflow shape in n8n

1. **Trigger** (Webhook or HTTP Request). Carry an account identifier in a header or payload.

2. **Lookup account** in the Data Store. Get `industry`, `account_slug`, `repo`, `branch`.
3. **Load agent.yaml** from GitHub. Convert YAML to JSON.
4. **Resolve paths**. Replace variables in `kb_selector.paths`. Build the final ordered list.
5. **Fetch files** from GitHub using the GitHub node. For each needed file, scan the list in order and take the first hit.
6. **Compose prompts** and call OpenAI using your existing credential.
7. **Log** to a Data Store for audit.

What the GitHub webhook does

- Fires on push. n8n receives changed paths.
- If changed paths are under any KB folder that your registry references, mark that account's KB "stale" in a small cache table or re-index immediately.
- None of this affects routing. It just keeps your cache fresh and cheap.

Why this makes switching trivial later

Today, you could point every account to `kb/shared` and one industry folder. Later, when Bryten needs its own SOPs:

- Create `kb/accounts/bryten-real-estate/...` in GitHub.
- Add one row to the Account Registry with `account_slug = bryten-real-estate`, `industry = multifamily`.
- No workflow changes. The resolver will automatically include Bryten's folder and its files will override shared or industry versions.

Access control and cost

- **Cost**. GitHub private repos are free at your size. n8n Cloud cost unchanged. No servers to maintain.
- **Security**. Use a fine-grained GitHub token with read-only scope to that one repo. Keep OpenAI keys in n8n Credentials, not in GitHub.
- **Future split**. If a client someday needs their own repo, your registry already supports `kb_repo` and `kb_branch`. Swap values in the registry and you are done.

Bottom line

A GitHub KB with a resolver in n8n is the right foundation. Webhooks help you sync changes. The registry plus selector tells the agent exactly which KB to read. Start with one folder if you want. Because the resolver exists from day one, moving Bryten or any future client to their own KB later is a data change, not a rebuild.

If you want, I can lay out the exact n8n nodes for the resolver and hand you a tiny sample Data Store schema so your dev can wire it in one pass.