Content Repurposing Pack — Setup Guide

Import steps • Configuration • curl tests • FAQ 3 n8n workflows for RSS digests, platform rewrites and publish queues

Spis treści

• Content Repurposing Pack — Setup Guide

Content Repurposing Pack — Setup Guide

Content Repurposing Pack - Setup Guide

Three importable n8n workflows that turn existing content into a scheduled digest and channel-specific drafts. Everything runs inside n8n. The external dependencies are an OpenAI-compatible `/v1/chat/completions` endpoint, an SMTP account, and (for W1) a feed that returns JSON.

Contents

`w1-rss-digest.json`	**Feed digest: summarize every new feed item and mail one digest**	**Schedule, every 6 hours**	**11**
`w2-content-converter.json`	Content converter: rewrite one text for X, LinkedIn or email	Webhook `POST /convert`	9
`w3-publish-queue.json`	Publish queue: four emails over 72 hours for one piece of content	Webhook `POST /queue`	9

All three use core nodes only (`n8n-nodes-base.*`): `scheduleTrigger`, `webhook`, `httpRequest`, `set`, `if`, `emailSend`, `splitOut`, `merge`, `wait`. No community nodes, no sub-workflows, no database of their own.

Prerequisites

- n8n 1.60 or newer, self-hosted (Docker) or n8n Cloud. The workflows are built for these node versions: `scheduleTrigger` 1.1, `webhook` 2, `httpRequest` 4.2, `set` 3.4, `if` 2.2, `emailSend` 2.1, `splitOut` 1, `merge` 3, `wait` 1.1. Older 1.x builds may show a version warning on import. [verify] against your installed version.
- An API key for any OpenAI-compatible `/v1/chat/completions` endpoint that accepts the `gpt-4o-mini` model name, or the ability to change the model name (see FAQ 1).
- An SMTP account you can send from.
- For W1: a feed endpoint that returns JSON with an `items` array. n8n's HTTP Request node does not parse XML, so a raw RSS/Atom URL needs a JSON bridge - see *Workflow 1 -> Feed source* below.
- For W2 and W3: a reachable base URL for webhooks. For local testing, n8n's own host is enough.

Import a workflow

Download the `.json` file you want to import.

Open n8n and open the workflow editor.

Open the `...` menu at the top right of the editor and choose Import from File. Some builds expose the same entry as Overview -> Add workflow -> Import from file. [verify] which label your build uses.

Pick the `.json` file. The workflow opens as a new, unsaved workflow.

Save it (`Ctrl/Cmd + S`).

Repeat for the other files. Each import creates a separate workflow.

Importing activates nothing. A webhook answers only after you activate its workflow, and the schedule trigger fires only while its workflow is active.

Set the API key

W1 and W2 call the model from an HTTP Request node. The key is read from the environment variable `OPENAI_API_KEY`:

```
Authorization: =Bearer {{$env.OPENAI_API_KEY}}
```

The variable must be present in the n8n process, not in your shell, and n8n must be restarted after you add it. Expressions are resolved per execution, so no workflow edit is needed afterwards.

Docker, inline:

```bash
docker run -d --name n8n \
-p 5678:5678 \
-e OPENAI_API_KEY="sk-your-key" \
-v n8n_data:/home/node/.n8n \
docker.n8n.io/n8nio/n8n
```

Docker Compose:

```yaml
services:
n8n:
image: docker.n8n.io/n8nio/n8n
ports:
- "5678:5678"
environment:
- OPENAI_API_KEY=sk-your-key
- N8N_BLOCK_ENV_ACCESS_IN_NODE=false
volumes:
- n8n_data:/home/node/.n8n
volumes:
n8n_data:
```

`N8N_BLOCK_ENV_ACCESS_IN_NODE` must be unset or `false`. When it is `true`,

`{{$env.OPENAI_API_KEY}}` resolves to an empty string and the model call returns `401`. [verify] the default on your distribution channel.

n8n Cloud: environment variables are available on paid plans only. [verify] If yours does not support them, use a credential instead of the header expression:

Create a Header Auth credential: name `Authorization`, value `Bearer sk-your-key`.

Open *Summarize Item (OpenAI)* in W1 and *AI Rewrite (OpenAI)* in W2.

Under Authentication choose Generic Credential Type -> Header Auth and select the credential.

Delete the `Authorization` entry from the node's header list so the header is not sent twice.

Configure SMTP

Go to Credentials -> Add credential and choose SMTP, the credential type the *Send Email* node uses.

Fill host, port, user and password. Use TLS for port 465 (implicit) or 587 (STARTTLS). Gmail requires 2-Step Verification and an app password, host `smtp.gmail.com`. [verify] your provider's current requirements.

Save and test the credential.

Open each email node, select your SMTP credential, and fill From Email. The pack leaves From Email empty; n8n then falls back to the SMTP user, which is usually what you want.

Fill To Email in W1, or send an `email` field in the webhook body for W2 and W3.

The JSON files ship with a placeholder credential reference (`credentials.smtp.id = "REPLACE_WITH_SMTP_CREDENTIAL_ID"`). Until you select your own credential the node shows a missing-credential warning; import is unaffected.

Email nodes per workflow:

W1	*Send Digest (Email SMTP)*
W2	*Send X Email*, *Send LinkedIn Email*, *Send Email-Format Email*
W3	*Send Queued Receipt (Email)*, *Send Slot 1 (Email)*, *Send Slot 2 (Email)*, *Send Slot 3 (Email)*

Workflow 1 - Feed digest

Chain: *Every 6 Hours* -> *Config (Feed URL)* -> *Fetch Feed* -> *Split Items* -> *Summarize Item (OpenAI)* -> *Extract Headline and Summary (Set)* -> *Skip Empty Summaries (IF)* -> *Collect Digests (Merge)* -> *Build Digest (Set)* -> *First Item Only (IF)* -> *Send Digest (Email SMTP)*.

Feed source

Fetch Feed performs a `GET` on `{{ $('Config (Feed URL)').first().json.feedUrl }}`. The response must be JSON with an `items` array of objects; *Split Items* splits on that field. The shipped default points at a public RSS-to-JSON bridge:

```

https://api.rss2json.com/v1/api.json?rss_url=https%3A%2F%2Fnews.ycombinator.com%2Frss

```

Two ways to point it at your own source:

- JSON bridge (any RSS/Atom feed). URL-encode your feed and append it as `rss_url`, for example `https://api.rss2json.com/v1/api.json?rss_url=https%3A%2F%2Fyour-domain.com%2Ffeed.xml`. The bridge

returns `{ status, feed, items: [ { title, link, pubDate, description, content, ... } ] }`, which is

the shape the workflow reads. Free use of public bridges is rate limited and may ask for an API key;

an empty `items` array or an error object is what you see when that happens. [verify] the current

limits of the bridge you use.

- Native JSON feed. If your CMS or newsletter tool already serves JSON with an `items` array, put that URL in the config node and delete nothing else. Item fields the workflow reads, all optional

except that one of them should exist: `title`, `link` (falls back to `url`), and

`description` / `content` / `summary` for the text.

Change the feed URL

Open the *Config (Feed URL)* node.

Edit the `feedUrl` value.

Save. No other node needs a change.

Change the schedule

Open the *Every 6 Hours* node.

Set Hours interval to the interval you want, for example `4` or `12`.

Save and activate the workflow.

The trigger runs in the instance timezone. Set `GENERIC_TIMEZONE` on the n8n process if that is not the

timezone you expect. [verify] your instance's setting.

What one run produces

- One model call per feed item, each summarizing that item into a headline plus two sentences.
- Items whose summary comes back empty are dropped by *Skip Empty Summaries*; the false output is

intentionally left unwired.

- *Build Digest (Set)* builds one text block for the whole run and marks the first item `yes`.
- *First Item Only* lets that one item through, so the run sends exactly one email. Without it every

item would send the same digest again.

- Email: subject `Content digest: N item(s)`, body listing `- headline`, the two-sentence summary and

the link for every kept item, then item count and generation timestamp.

Per-item cost is roughly the item text up to 6000 characters plus a short prompt. A feed of ten items

is ten model calls per run; at a 6-hour schedule that is four runs and up to 40 calls per day. Set the

interval to `12` or `24` hours if that is too much.

Workflow 2 - Content converter

Chain: *Convert Webhook* -> *Normalize Request (Set)* -> *AI Rewrite (OpenAI)* -> *Extract Rewrite (Set)* -> *Is X (IF)* -> *Send X Email*, else *Is LinkedIn (IF)* -> *Send LinkedIn Email*, else *Send Email-Format Email*.

Two IF nodes are used because an IF node has exactly two outputs: the first splits X from the rest, the second splits LinkedIn from everything else. An unknown `platform` value therefore lands on the email-format branch.

Request body

`content`	**yes**	**The long text to repurpose. Trimmed; length is recorded as `contentChars`.**
`platform`	yes	Case-insensitive. `x` and `linkedin` have their own branch; anything else, including `email`, uses the email-format branch.
`tone`	no	Free text passed to the model. Defaults to `neutral`.
`email`	no	Recipient. Defaults to `your@inbox.com`, so set it before real use.

Activate and test

Import, select your SMTP credential in all three email nodes, save.

Activate the workflow.

Send a request:

```bash
curl -X POST "https://n8n.example.com/webhook/convert" \
-H "Content-Type: application/json" \
-d '{
"content": "We rewrote our scheduler last quarter. The old one polled every queue every 200 ms and spent most of its budget on empty reads. The new one is event-driven and wakes only when a queue changes. Latency dropped from 800 ms to 40 ms, and the fleet costs 60 percent less.",
"platform": "x",
"tone": "direct, no hype",
"email": "you@example.com"
}'
```

The webhook answers immediately; the rewrite and the email continue in the execution log. Check Executions if no email arrives.

Same call for the other branches: send `"platform": "linkedin"` and `"platform": "email"`.

Platform rules the prompt enforces

x	First line of the reply	At most 280 characters including spaces and hashtags, ending with 3 hashtags
linkedin	First line of the reply	One-line hook, 4 short paragraphs, then a call to action
anything else (email format)	First line of the reply	3 paragraphs

The model is instructed to answer in a fixed layout: line 1 is `Subject:` plus the subject line, line 2 is empty, then the body. *Extract Rewrite (Set)* turns that layout into `subject`, `body` and `bodyChars`, so the email templates never depend on the raw API response.

Character counts are JavaScript `String.length`, so an emoji counts as two. Treat `bodyChars` as a sanity check, not as an exact X counter.

What each email contains

- X: platform, tone, body length against the 280 limit, the post text, and the source length.
- LinkedIn: platform, tone, body length, the post text, and a reminder that the first line is the hook.
- Email format: platform, tone, subject, body length, the body, and a reminder to copy the subject line into the email tool's subject field.

Customize the prompt

Open *AI Rewrite (OpenAI)*. The `jsonBody` field is one expression that builds the request:

```
={{ JSON.stringify({ model: 'gpt-4o-mini', temperature: 0.4, messages: [ ... ] }) }}
```

Edit the `system` message string to change style rules and the `user` message to change what is sent. Keep the `Subject:` /blank-line/body layout, or adjust the two expressions in *Extract Rewrite (Set)* to match whatever layout you switch to.

Workflow 3 - Publish queue

Chain: *Queue Webhook* -> *Normalize Queue Item (Set)* -> *Send Queued Receipt (Email)* -> *Wait 6 Hours* -> *Send Slot 1 (Email)* -> *Wait 18 Hours* -> *Send Slot 2 (Email)* -> *Wait 48 Hours* -> *Send Slot 3 (Email)*.

Request body

title	no	Used in every subject line. Defaults to `(no title)`.
content	yes	Repeated in the receipt and in all three slots.

`title`	**no**	**Used in every subject line. Defaults to `(no title)`.**
`platforms`	no	Comma-separated text, for example `x,linkedin`. Trimmed and joined back into `x, linkedin`.
`email`	no	Recipient. Defaults to `your@inbox.com`.

Timeline

Receipt	**immediately**	`Queued: <title> [q-<id>]`
Slot 1	+6 hours	`Publish now - slot 1: <title>`
Slot 2	+24 hours (6 + 18)	`Publish now - slot 2: <title>`
Slot 3	+72 hours (6 + 18 + 48)	`Publish now - slot 3: <title>`

The waits are relative to the moment the item was queued, not fixed clock times. `Normalize Queue Item

(Set) `stamps a` queueId` from the execution clock so the four emails of one item can be matched.

Activate and test

Import, select your SMTP credential in all four email nodes, save.

Activate the workflow.

Send a request:

```bash
curl -X POST "https://n8n.example.com/webhook/queue" \
-H "Content-Type: application/json" \
-d '{
"title": "Scheduler rewrite write-up",
"content": "We rewrote our scheduler last quarter. Latency dropped from 800 ms to 40 ms and the fleet costs 60 percent less. Full post: https://example.com/scheduler",
"platforms": "x,linkedin",
"email": "you@example.com"
}'
```

Only the receipt arrives at first. The other three arrive as the waits expire; each one is a separate execution segment, and the workflow must stay active for them to resume.

What each email contains

- Receipt: queue id, creation time, title, platform list, the full content between `--- content ---` markers, and the send schedule for the entry.
- Each slot: which slot of three it is, the queue id, platform list, send time, the full content between `--- copy ---` markers, and a pointer to the next email. Slot 3 says it is the last message.

The pack does not call any publishing API. These emails are the queue: a human posts the copy when the

mail arrives. Adding a platform API node later means replacing the *Send Slot* email nodes, not changing

the waits.

FAQ

1. Can I use a different model or provider?

Yes. Both AI nodes call `https://api.openai.com/v1/chat/completions` with `model: 'gpt-4o-mini'` inside

`jsonBody`. Point `url` at any OpenAI-compatible endpoint and change the `model` string. If the provider

needs a different auth header, edit `headerParameters` in the same node.

2. Do I need a bridge to read an RSS feed?

Yes, unless your source already serves JSON. n8n's HTTP Request node returns the XML body as a string

and none of the nodes used here parse XML; *Split Items* needs a real array. The default `feedUrl` uses a

public RSS-to-JSON bridge, and any equivalent bridge works as long as the items land under `items`.

3. Can I change how many items go into one digest?

The feed decides. This pack does not use the *Limit* node, to stay inside the nine core node types listed

above, so one run consumes every item the feed returns. Ask the feed for fewer items (most bridges accept

a count parameter) or lower the trigger frequency.

4. What happens if the model ignores the requested layout?

Extract falls back instead of failing: with no `Headline:` prefix the first line becomes the headline

and the whole text becomes the summary; with no `Subject:` prefix the first line becomes the subject

and the rest becomes the body. An empty reply produces an empty summary, which W1's filter then drops.

If a reply is wrapped in code fences, the fences will show up in the email body - tighten the system

prompt if your model does that.

5. Can an entry go to more than one address?

Yes. The `To Email` field accepts a comma-separated list, so `a@example.com,b@example.com` works in any

of the email nodes. Passing a comma-separated list in the webhook `email` field works for W2 and W3 as

well. The SMTP server must accept relaying for those addresses.

6. Can I run this on n8n Cloud?

Yes. Node versions and core-node behaviour are the same. Two caveats: environment variables need a paid

plan (use a Header Auth credential instead, see *Set the API key*), and webhook paths are registered per

workflow, so do not import two copies of the same file into one instance. [verify] plan limits.

7. How do I stop a queue entry that is already running?

Every wait is its own execution segment. Deactivate the workflow to stop all pending waits; delete the waiting executions in Executions to stop individual entries. Reactivating the workflow does not restore deleted executions.

Troubleshooting

The feed fetch fails

- `404` or `403`: the URL is wrong, the bridge rejects the request, or the feed requires an auth header or a browser user agent. Test the exact URL in a browser or with `curl -sS "https://api.rss2json.com/v1/api.json?rss_url=..."`.
- `Split Items` errors with something like "requires an array": the response is not JSON, or the JSON has no `items` array - a rate-limited or failing bridge typically returns `{ "status": "error", ... }`. Check the *Fetch Feed* node's output in the execution.
- An empty `items` array produces zero items out of *Split Items*, so the rest of the run never executes and no email is sent. A digest with a lower item count than the feed means the summary filter dropped the empty ones - check the IF node's false output in the execution.

Summaries come back empty

- `401` from the model endpoint: `OPENAI_API_KEY` is missing in the n8n process, or env access is blocked. Restart n8n after adding it.
- `429`: rate limit. Lower the trigger frequency, or ask the feed for fewer items.
- The model returns prose instead of the layout: tighten the `system` message in *Summarize Item (OpenAI)*; the extraction falls back to the first line as the headline.
- A digest with fewer items than the feed has means the filter dropped the rest. Open the IF node's false output in the execution to see which summaries were empty.

Wait executions and restarts

- A *Wait* node longer than 65 seconds does not hold a process open. n8n saves the execution and resumes it later from the database, so the instance must stay up and the workflow must stay active. [verify] the threshold on your version.
- Editing the workflow while entries are waiting is risky: the wait resumes on the saved execution, but node parameters are re-read. Save edits, then expect pending entries to continue on the new version. [verify] on your n8n version.
- After an instance restart, waiting executions resume on the next scheduler pass; expect a delay measured in minutes, not seconds. [verify]

- Execution pruning (`EXECUTIONS_DATA_PRUNE`, `EXECUTIONS_DATA_MAX_AGE`) can delete a waiting execution

before it resumes. Keep the retention window longer than your longest wait (48 hours here).

- Manual/test executions of W3 also queue the waits. If you only want to inspect the receipt, stop the execution after the first email, or test on a copy of the workflow with the waits removed.

No email arrives

- The node shows a missing credential: select your SMTP credential in every email node. There are three

in W2 and four in W3.

- `To Email` is still `your@inbox.com` (W1) or the webhook body has no `email` field (W2, W3). Both are placeholders, not a delivered address.

- The webhook returned `404`: the workflow is not active, or you called the production URL (`/webhook/...`) instead of the test URL (`/webhook-test/...`) while listening for a test event.

Webhook returns before the work is done

Both webhooks respond with `responseMode: onReceived`, so the HTTP call returns as soon as n8n receives

the request. Errors after that point appear in Executions, never in the curl response. If you need the result in the response, the *Respond to Webhook* node does that - it is outside this pack's node set.

Known limitations

W1 sends one digest per run, not one email per item. *Build Digest* builds the digest for all kept items, and *First Item Only* forwards a single item to the email node so exactly one message is sent. Without that second filter the email node would run once per item and send the same digest repeatedly.

The digest is text, not HTML. Every email node sends plain text. HTML needs an `html` field on the email nodes.

No retries and no error branch. A failed model call fails the whole run. n8n's Retry On Fail setting on the HTTP Request nodes is the smallest fix; an Error Trigger workflow is the full one.

No de-duplication in W1. The same feed item appears in the digest on every run while it stays in the feed. Persisting a "last seen" marker needs a Code node with workflow static data, which is outside the allowed node set.

W2 and W3 do not store their requests. The content lives in the execution record (and in the emails). Nothing is written to a database, so searchable history is the execution list.

W3's schedule is relative, not absolute. Waits start when the item is queued. If you need "Tuesdays at 09:00", replace the waits with a schedule trigger and a stored queue.

The pack assumes the model follows the `Subject:` / `Headline:` layout. Extraction degrades gracefully but not perfectly; a strict JSON reply would need a Code node to parse.