

Webhooks

Tell other systems when something happens in Resourcecloud – a task is finished, a comment is added, an incident comes in – so n8n, Make, Zapier or your own service can react.

Resourcecloud posts each event as signed JSON to a URL you choose.

Overview

- A **webhook** is a URL plus the events it wants. It is for the **whole instance** (RC Admins) or for **one team's work** (the team's lead or deputy, and RC Admins).
 - Every event is posted **signed** with the webhook's secret, so the receiver knows it is genuine.
 - A delivery that fails is **retried**; every attempt is in a **log** you can read and resend from.
 - Nothing waits for the receiver: events are queued when they happen and sent by the scheduler within about a minute.
-

1. Add a webhook

- 1 Open **Settings → External Services → Webhooks** (RC Admins) or the **Webhooks** card in your team's **Team Settings** (team lead, deputy). Click **Add webhook**.
- 2 Give it a **name**, the **URL** that receives the events (https, a public address) and the **events** you want – or **All events**.
- 3 Choose **Events from**: the whole instance, or one team. This cannot be changed later.
- 4 Save. The **secret** is shown **once**: copy it into the receiver now. If you lose it, use **New secret**.
- 5 Click **Test**: a **ping** event goes through the whole path and the page tells you what the receiver answered.

The URL must be reachable from the Resourcecloud server. Addresses in private ranges (10.x, 192.168.x, localhost, cloud metadata) are refused so a webhook cannot be used to probe your own network. A self-hosted instance that really must post into its intranet sets `webhook_allow_private` in its configuration, or the environment variable `RC_WEBHOOK_ALLOW_PRIVATE=1` .

2. The events

Event	When	Team
<code>task.created</code>	a task is created (any way: form, quick insert, e-mail, API)	the task's team
<code>task.updated</code>	a task is changed in the task editor, moved on the Kanban board or changed through the API	the task's team
<code>task.done</code>	a task is finished (Done button, Done lane, API)	the task's team
<code>comment.created</code>	a comment is added to a task or a project	the task's team
<code>incident.created</code>	an incident is created (comes <i>in addition to</i> <code>task.created</code>)	the incident's team
<code>mention.created</code>	somebody is mentioned with <code>@username</code> in a comment (one event per person mentioned; only if that person may see the task)	the task's team; a project comment: the project's teams
<code>incident.escalated</code>	an incident ran past its reaction time or its escalation time (comes whoever else is notified)	the incident's team
<code>project.status_changed</code>	a project is requested for approval, approved, rejected, kicked off, cancelled or finished	the project's own team and the teams that have tasks in it
<code>vacation.approved</code>	leave is approved	instance-wide webhooks only

A webhook for a team receives only the events of that team's work. A project has no team of its own, so its events go to the webhooks of its project team and of the teams that have tasks in it; leave has no team at all, so only instance-wide webhooks get it.

Every event has the same envelope; `data` depends on the event:

```
{
  "id": "5d1f0c2e8a7b4e3f9c6a1b2d3e4f5a6b",
  "event": "task.done",
  "created": "2026-10-11T09:25:13+00:00",
  "instance": "https://your-instance.example.com",
  "data": {
    "id": 4711, "title": "Write the release notes", "type": "Development", "team_id":
    "parent_task_id": null, "customer_id": null, "assigned_worker": "anna", "priority"
    "due": "2026-10-20", "deadline": null, "done": true, "done_date": "2026-10-11 11:2
    "url": "https://your-instance.example.com/edit_workflow?workid=4711",
    "actor": "anna"
  }
}
```

Event	data
<code>task.*</code> , <code>incident.created</code>	as above; <code>task.updated</code> adds <code>changed</code> (the fields, when known: always from the API and the Kanban board, <code>["stage"]</code> for a move), <code>actor</code> is who did it (absent for the system)
<code>comment.created</code>	<code>id</code> , <code>task_id</code> , <code>project_id</code> , <code>author</code> , <code>internal</code> (false = visible to the customer) , <code>text</code> (plain text, at most 2000 characters) , <code>minutes</code> , <code>created</code> , <code>task_url</code>
<code>project.status_changed</code>	<code>id</code> , <code>name</code> , <code>code</code> , <code>status</code> , <code>state</code> , <code>done</code> , <code>project_manager</code> , <code>url</code> , <code>actor</code> ; status is <code>planning</code> , <code>approved</code> , <code>rejected</code> , <code>in_progress</code> , <code>delayed</code> , <code>waiting</code> , <code>waiting_for_approval</code> , <code>canceled</code> or <code>done</code>
<code>incident.escalated</code>	as a task, plus <code>violation</code> : <code>reaction</code> or <code>escalation</code>
<code>mention.created</code>	<code>comment_id</code> , <code>mentioned</code> (username) , <code>by</code> , <code>text</code> (at most 2000 characters) , <code>task_id</code> , <code>task_title</code> , <code>project_id</code> , <code>team_id</code> , <code>url</code>
<code>vacation.approved</code>	<code>id</code> , <code>worker</code> , <code>substitute</code> , <code>begin_date</code> , <code>end_date</code> , <code>days</code> , <code>type</code> , <code>approved_by</code>

ping	message, subscription_id - only from the Test button
------	--

The description of a task and the history of its comments are **not** sent. A receiver that needs more reads it from the REST API with the id.

Private tasks never produce events - neither for the task nor for its comments. An integration is not a party of a private task.

3. Check the signature

Every request carries:

Header	Meaning
X-RC-Event	the event name
X-RC-Delivery	the event id - the same on every retry ; remember it if your reaction must not run twice
X-RC-Attempt	1 for the first try, 2 for the first retry ...
X-RC-Signature	t=<unix time>,v1=<hex> where v1 is the HMAC-SHA256, with the webhook's secret as key, of t + "." + the raw request body

A receiver recomputes the HMAC from the **raw body** (before any JSON parsing), compares it in constant time and rejects a t older than five minutes.

PHP

```
$body = file_get_contents('php://input');
sscanf($_SERVER['HTTP_X_RC_SIGNATURE'], 't=%d,v1=%s', $t, $sig);
$ok = abs(time() - $t) <= 300
    && hash_equals(hash_hmac('sha256', $t . '.' . $body, $secret), $sig);
```

Node.js

```
const crypto = require('crypto');
const [, t, sig] = /^t=(\d+),v1=([0-9a-f]{64})$/ .exec(req.headers['x-rc-signature']);
const expected = crypto.createHmac('sha256', secret).update(t + '.' + rawBody).digest('hex');
const ok = Math.abs(Date.now() / 1000 - t) <= 300
    && crypto.timingSafeEqual(Buffer.from(expected), Buffer.from(sig));
```

Python

```
import hmac, hashlib, re, time
t, sig = re.match(r't=(\d+),v1=([0-9a-f]{64})$', headers['X-RC-Signature']).groups()
expected = hmac.new(secret.encode(), (t + '.').encode() + raw_body, hashlib.sha256).hexdigest()
ok = abs(time.time() - int(t)) <= 300 and hmac.compare_digest(expected, sig)
```

After **New secret** the old secret stops signing at once - change the receiver first or accept a short gap.

4. Delivery, retries, log

- The receiver must answer **2xx** within **10 seconds**. Anything else (an error status, a timeout, no connection) is a failed attempt. Redirects are not followed.
- Retries come after 1 minute, 5 minutes, 30 minutes, 2 hours and 12 hours - six attempts in all. Then the delivery is **Given up**; it stays in the log and **Send again** starts a fresh run.
- Answer **410 Gone** to say "stop": the webhook switches itself off. It also switches off after 25 failed attempts in a row; the page shows why. Switch it on again after fixing the receiver.
- Open **Log** on the Webhooks page for the last 30 days: time, event, state, attempts and the receiver's answer. Older records are removed.
- Answer fast and work later: acknowledge first, then process. A receiver that takes longer than 10 seconds is retried and gets the same event twice.

5. n8n

The file `docs/webhooks/n8n-task-done.json` is a ready flow: **Webhook** → **Verify signature** → **Task finished?** → **What was finished** (title, who, team, link) → a placeholder node for your own step.

- 1 In n8n choose **Import from file** and pick the JSON.
- 2 Open **Verify signature** and replace `PASTE-YOUR-WEBHOOK-SECRET-HERE` with the secret. n8n must allow the crypto module: start it with `NODE_FUNCTION_ALLOW_BUILTIN=crypto`.
- 3 Open the **Webhook** node and copy its **Production URL** into a new webhook in Resourcecloud, event `task.done`. Publish (activate) the flow.
- 4 Click **Test** in Resourcecloud, then finish a task. The flow runs; the `ping` is filtered out by **Task finished?**.

Replace the last node with what you need: a Slack or Teams message, a row in a sheet, a ticket in another tool. Change the condition in **Task finished?** for other events. The flow answers immediately (*Respond: immediately*); a signature that does not match shows up as a failed execution in n8n.

6. Make and Zapier

Make: add a **Webhooks** → **Custom webhook** module, copy its address into a Resourcecloud webhook, click **Test** so Make learns the structure, then build the scenario on it. Make cannot check the signature by itself; keep the address secret, or add a *Tools* step that computes the HMAC (see above).

Zapier: create a Zap with the trigger **Webhooks by Zapier** → **Catch Hook**, copy the Custom Webhook URL into a Resourcecloud webhook and click **Test**. The fields of `data` then show up as `data title`, `data assigned_worker` ... in the next steps. Zapier too does not check signatures on its own; a **Code by Zapier** step can.

7. Slack and Teams

A webhook can post a ready-made message into a chat channel instead of sending JSON. Choose **Send to: Slack channel** or **Microsoft Teams channel** when you add it. The webhook is then for **one team's channel** (choose the team under **Events from**) - or for the whole instance, if one channel should hear everything.

- 1 **Slack:** create an *Incoming Webhook* for the channel (Slack app → Incoming Webhooks → Add New Webhook to Workspace) and copy its URL (`https://hooks.slack.com/services/...`).
- 2 **Teams:** in the channel choose **Workflows** and the template **Post to a channel when a webhook request is received** (*When a Teams webhook request is received*), and copy the URL it shows. Microsoft is retiring the older "Incoming Webhook" connector; use Workflows.
- 3 In Resourcecloud add a webhook, pick the type, paste the URL, choose the team and save. A channel starts with the four events it is for: **new incidents**, **escalations** (an incident ran past its reaction or escalation time), **mentions** (`@name` in a comment) and **project approvals** (a project waits for approval, is approved or rejected, starts, is cancelled or finished). Change them if you like.
- 4 Click **Test**: the channel gets a test message.

The message says what happened in one line, with the team, assignee and due date as facts and a link **Open** to the task or project. It is written in the instance's default language. The address is the only secret there is - anyone who has it can post to the channel - so a Slack or Teams webhook is **not signed** and has no **New secret**. Retries, the log and the rules of section 4 are the same.

Task changes, new comments and "task finished" can be switched on per channel in the same list. A mention in a private task never reaches a channel.

8. REST API and MCP

Everything on the page is in the API (openapi.yaml, tag *Webhooks*): `GET /webhooks/events` , `GET|POST /webhooks` , `GET|PUT|DELETE /webhooks/{id}` , `POST /webhooks/{id}/test` , `POST /webhooks/{id}/rotate-secret` , `GET /webhooks/{id}/deliveries` , `POST`

/webhooks/{id}/deliveries/{delivery_id}/redeliver . The same rules apply: RC Admin, or lead/deputy for the team's webhooks. The secret is in the answer of the POST that creates it, and of rotate-secret, and nowhere else (a Slack or Teams webhook, "format": "slack" or "teams", has none).

```
curl -u USER:TOKEN -X POST https://YOUR-INSTANCE/api/v1/webhooks \
-H 'Content-Type: application/json' \
-d '{"name": "n8n", "url": "https://n8n.example.com/webhook/resourcecloud", "events"
```

The hosted MCP endpoint has the read tools `list_webhook_events`, `list_webhooks` and `list_webhook_deliveries` . Creating a webhook (and so a secret) is left to the page and the REST API.

Rules and limits

- Events are recorded where people and the API do the work: the task forms and editor, quick insert, the Kanban board, the Done button, the REST API, e-mail to task, comments, the project approval screens and leave approval. Work that no person does - the autoclose job, a Jira import, scheduled-task generation - does **not** fire events, and neither do the smaller one-field edits of a task (renaming, changing the type from a list) that have not been wired up yet.
- Webhooks are at-least-once: a receiver may get an event twice (a retry after a timeout). Use `X-RC-Delivery` to ignore a repeat.
- The order of events is the order they happened in, but a retry can arrive after a later event.
- Delivery records are kept 30 days.
- Deleting a webhook deletes its log and drops what was still queued.