

Webhooks

Sage anderen Systemen Bescheid, wenn in Resourcecloud etwas passiert – eine Aufgabe ist erledigt, ein Kommentar kommt dazu, ein Incident geht ein –, damit n8n, Make, Zapier oder dein eigener Dienst reagieren kann. Resourcecloud sendet jedes Ereignis als signiertes JSON an eine URL deiner Wahl.

Überblick

- Ein **Webhook** ist eine URL plus die Ereignisse, die sie haben will. Er gilt für die **ganze Instanz** (RC-Admins) oder für die **Arbeit eines Teams** (Teamleitung, Vertretung und RC-Admins).
 - Jedes Ereignis wird mit dem Geheimnis des Webhooks **signiert** gesendet; der Empfänger erkennt, dass es echt ist.
 - Eine fehlgeschlagene Zustellung wird **wiederholt**; jeder Versuch steht in einem **Protokoll**, aus dem du erneut senden kannst.
 - Nichts wartet auf den Empfänger: Ereignisse werden im Moment ihres Entstehens vorgemerkt und vom Scheduler binnen etwa einer Minute gesendet.
-

1. Webhook anlegen

- 1 Öffne **Einstellungen → Externe Dienste → Webhooks** (RC-Admins) oder die Karte **Webhooks** in den **Team-Einstellungen** deines Teams (Teamleitung, Vertretung). Klicke auf **Webhook hinzufügen**.
- 2 Vergib einen **Namen**, die **URL**, die die Ereignisse empfängt (https, eine öffentliche Adresse), und die gewünschten **Ereignisse** – oder **Alle Ereignisse**.
- 3 Wähle **Ereignisse aus**: die ganze Instanz oder ein Team. Das lässt sich später nicht ändern.
- 4 Speichere. Das **Geheimnis** wird **einmal** angezeigt: Trage es jetzt beim Empfänger ein. Hast du es verloren, nimm **Neues Geheimnis**.

- 5 Klicke auf **Testen**: Ein `ping`-Ereignis läuft den ganzen Weg, und die Seite zeigt, was der Empfänger geantwortet hat.

Die URL muss vom Resourcecloud-Server aus erreichbar sein. Adressen in privaten Bereichen (10.x, 192.168.x, localhost, Cloud-Metadaten) werden abgelehnt, damit ein Webhook nicht dazu dient, dein Netz abzutasten. Eine selbst betriebene Instanz, die wirklich ins Intranet senden muss, setzt `webhook_allow_private` in ihrer Konfiguration oder die Umgebungsvariable `RC_WEBHOOK_ALLOW_PRIVATE=1` .

2. Die Ereignisse

Ereignis	Wann	Team
<code>task.created</code>	eine Aufgabe wird angelegt (egal wie: Formular, Schnellaufgabe, E-Mail, API)	Team der Aufgabe
<code>task.updated</code>	eine Aufgabe wird im Editor geändert, auf dem Kanban-Board bewegt oder über die API geändert	Team der Aufgabe
<code>task.done</code>	eine Aufgabe wird erledigt (Erledigt-Knopf, Erledigt-Spalte, API)	Team der Aufgabe
<code>comment.created</code>	zu einer Aufgabe oder einem Projekt kommt ein Kommentar	Team der Aufgabe
<code>incident.created</code>	ein Incident wird angelegt (kommt <i>zusätzlich zu</i> <code>task.created</code>)	Team des Incidents
<code>mention.created</code>	jemand wird mit <code>@benutzername</code> in einem Kommentar erwähnt (ein Ereignis je erwähnter Person; nur wenn sie die Aufgabe sehen darf)	Team der Aufgabe; bei einem Projektkommentar: die Teams des Projekts
<code>incident.escalated</code>	ein Incident hat Reaktions- oder Eskalationszeit überschritten (kommt, wer sonst auch benachrichtigt wird)	Team des Incidents

<code>project.status_changed</code>	ein Projekt wird zur Genehmigung vorgelegt, genehmigt, abgelehnt, gestartet, abgebrochen oder abgeschlossen	Projektteam und die Teams, die Aufgaben darin haben
<code>vacation.approved</code>	Urlaub wird genehmigt	nur instanzweite Webhooks

Ein Team-Webhook erhält nur die Ereignisse der Arbeit dieses Teams. Ein Projekt hat kein eigenes Team; seine Ereignisse gehen an die Webhooks seines Projektteams und der Teams, die Aufgaben darin haben. Urlaub hat gar kein Team, deshalb bekommen ihn nur instanzweite Webhooks.

Jedes Ereignis hat denselben Rahmen; `data` hängt vom Ereignis ab:

```
{
  "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"
  }
}
```

Ereignis	data
<code>task.*</code> , <code>incident.created</code>	wie oben; <code>task.updated</code> ergänzt <code>changed</code> (die Felder, wenn bekannt: immer bei der API und dem Kanban-Board, <code>["stage"]</code> bei einer Verschiebung), <code>actor</code> ist, wer es getan hat (fehlt beim System)
<code>comment.created</code>	<code>id</code> , <code>task_id</code> , <code>project_id</code> , <code>author</code> , <code>internal</code> (false = für den Kunden sichtbar) , <code>text</code> (reiner Text, höchstens 2000 Zeichen) , <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 ist <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> oder <code>done</code>

<code>incident.escalated</code>	wie eine Aufgabe, dazu <code>violation</code> : <code>reaction</code> oder <code>escalation</code>
<code>mention.created</code>	<code>comment_id</code> , <code>mentioned</code> (Benutzername) , <code>by</code> , <code>text</code> (höchstens 2000 Zeichen) , <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>
<code>ping</code>	<code>message</code> , <code>subscription_id</code> - nur vom Testen-Knopf

Die Beschreibung einer Aufgabe und die Kommentarhistorie werden **nicht** gesendet. Wer mehr braucht, liest es mit der Id aus der REST-API.

Private Aufgaben erzeugen nie Ereignisse - weder für die Aufgabe noch für ihre Kommentare. Eine Integration ist keine Beteiligte einer privaten Aufgabe.

3. Signatur prüfen

Jede Anfrage trägt:

Header	Bedeutung
<code>X-RC-Event</code>	der Ereignisname
<code>X-RC-Delivery</code>	die Ereignis-Id - bei jeder Wiederholung dieselbe ; merke sie dir, wenn deine Reaktion nicht doppelt laufen darf
<code>X-RC-Attempt</code>	1 beim ersten Versuch, 2 bei der ersten Wiederholung ...
<code>X-RC-Signature</code>	<code>t=<Unix-Zeit></code> , <code>v1=<hex></code> , wobei <code>v1</code> der HMAC-SHA256 mit dem Geheimnis des Webhooks als Schlüssel über <code>t</code> + <code>"."</code> + den rohen Anfragetext ist

Der Empfänger berechnet den HMAC aus dem **rohen Text** (vor jedem JSON-Parsen), vergleicht in konstanter Zeit und weist ein `t` ab, das älter als fünf Minuten ist.

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 + '.' + raw_body).encode(), hashlib.sha256).hexdigest()
ok = abs(time.time() - int(t)) <= 300 and hmac.compare_digest(expected, sig)
```

Nach **Neues Geheimnis** signiert das alte sofort nicht mehr - ändere zuerst den Empfänger oder nimm eine kurze Lücke in Kauf.

4. Zustellung, Wiederholung, Protokoll

- Der Empfänger muss binnen **10 Sekunden** mit **2xx** antworten. Alles andere (ein Fehlerstatus, eine Zeitüberschreitung, keine Verbindung) ist ein fehlgeschlagener Versuch. Weiterleitungen werden nicht verfolgt.
- Wiederholt wird nach 1 Minute, 5 Minuten, 30 Minuten, 2 Stunden und 12 Stunden - insgesamt sechs Versuche. Danach ist die Zustellung **Aufgegeben**; sie bleibt im Protokoll, und **Erneut senden** startet einen frischen Durchlauf.
- Antworte **410 Gone**, um „Hör auf“ zu sagen: Der Webhook schaltet sich ab. Er schaltet sich auch nach 25 fehlgeschlagenen Versuchen in Folge ab; die Seite zeigt den Grund. Schalte ihn wieder ein, wenn der Empfänger repariert ist.
- Öffne **Protokoll** auf der Webhooks-Seite für die letzten 30 Tage: Zeit, Ereignis, Zustand, Versuche und die Antwort des Empfängers. Ältere Einträge werden entfernt.

- Antworte schnell und arbeite später: erst bestätigen, dann verarbeiten. Ein Empfänger, der länger als 10 Sekunden braucht, wird wiederholt und erhält dasselbe Ereignis doppelt.
-

5. n8n

Die Datei `docs/webhooks/n8n-task-done.json` ist ein fertiger Ablauf: **Webhook** → **Verify signature** → **Task finished?** → **What was finished** (Titel, wer, Team, Link) → ein Platzhalter für deinen eigenen Schritt.

- 1 Wähle in n8n **Import from file** und nimm die JSON-Datei.
- 2 Öffne **Verify signature** und ersetze `PASTE-YOUR-WEBHOOK-SECRET-HERE` durch das Geheimnis. n8n muss das crypto-Modul erlauben: Starte es mit `NODE_FUNCTION_ALLOW_BUILTIN=crypto`.
- 3 Öffne den **Webhook**-Knoten und kopiere seine **Production URL** in einen neuen Webhook in Resourcecloud, Ereignis `task.done`. Veröffentliche (aktiviere) den Ablauf.
- 4 Klicke in Resourcecloud auf **Testen** und erledige dann eine Aufgabe. Der Ablauf läuft; der `ping` wird von **Task finished?** herausgefiltert.

Ersetze den letzten Knoten durch das, was du brauchst: eine Slack- oder Teams-Nachricht, eine Zeile in einer Tabelle, ein Ticket in einem anderen Werkzeug. Für andere Ereignisse änderst du die Bedingung in **Task finished?**. Der Ablauf antwortet sofort (*Respond: immediately*); eine Signatur, die nicht passt, erscheint in n8n als fehlgeschlagene Ausführung.

6. Make und Zapier

Make: Füge ein Modul **Webhooks** → **Custom webhook** hinzu, kopiere dessen Adresse in einen Resourcecloud-Webhook, klicke auf **Testen**, damit Make die Struktur lernt, und baue darauf das Szenario. Make prüft die Signatur nicht von selbst; halte die Adresse geheim oder ergänze einen *Tools*-Schritt, der den HMAC berechnet (siehe oben).

Zapier: Lege einen Zap mit dem Trigger **Webhooks by Zapier → Catch Hook** an, kopiere die Custom Webhook URL in einen Resourcecloud-Webhook und klicke auf **Testen**. Die Felder von `data` erscheinen dann als `data title`, `data assigned_worker` ... in den nächsten Schritten. Auch Zapier prüft Signaturen nicht von selbst; ein Schritt **Code by Zapier** kann es.

7. Slack und Teams

Ein Webhook kann statt JSON eine fertige Nachricht in einen Chat-Kanal schreiben. Wähle beim Anlegen **Senden an: Slack-Kanal** oder **Microsoft-Teams-Kanal**. Der Webhook gehört dann zum **Kanal eines Teams** (wähle das Team unter **Ereignisse aus**) - oder zur ganzen Instanz, wenn ein Kanal alles hören soll.

- 1 **Slack:** Lege für den Kanal einen *Incoming Webhook* an (Slack-App → Incoming Webhooks → Add New Webhook to Workspace) und kopiere seine URL (`https://hooks.slack.com/services/...`).
- 2 **Teams:** Wähle im Kanal **Workflows** und die Vorlage **Post to a channel when a webhook request is received** (*Beim Empfang einer Teams-Webhook-Anforderung*) und kopiere die angezeigte URL. Microsoft stellt den älteren „Incoming Webhook“-Connector ein; nimm Workflows.
- 3 Lege in Resourcecloud einen Webhook an, wähle den Typ, füge die URL ein, wähle das Team und speichere. Ein Kanal startet mit den vier Ereignissen, für die er da ist: **neue Incidents**, **Eskalationen** (ein Incident hat Reaktions- oder Eskalationszeit überschritten), **Erwähnungen** (`@name` in einem Kommentar) und **Projektfreigaben** (ein Projekt wartet auf Genehmigung, wird genehmigt oder abgelehnt, startet, wird abgebrochen oder abgeschlossen). Ändere sie nach Wunsch.
- 4 Klicke auf **Testen**: Der Kanal erhält eine Testnachricht.

Die Nachricht sagt in einer Zeile, was passiert ist, mit Team, Bearbeiter und Fälligkeit als Fakten und einem Link **Öffnen** zur Aufgabe oder zum Projekt. Sie ist in der Standardsprache der Instanz geschrieben. Die Adresse ist das einzige Geheimnis - wer sie kennt, kann in den Kanal schreiben -, darum ist ein Slack- oder Teams-Webhook **nicht signiert** und hat kein **Neues Geheimnis**. Wiederholung, Protokoll und die Regeln aus Abschnitt 4 gelten gleich.

Aufgabenänderungen, neue Kommentare und „Aufgabe erledigt“ lassen sich pro Kanal in derselben Liste zuschalten. Eine Erwähnung in einer privaten Aufgabe erreicht nie einen Kanal.

8. REST-API und MCP

Alles von der Seite gibt es in der 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`. Es gelten dieselben Regeln: RC-Admin oder Teamleitung/Vertretung für die Webhooks des Teams. Das Geheimnis steht in der Antwort des POST, der den Webhook anlegt, und in der von rotate-secret - sonst nirgends (ein Slack- oder Teams-Webhook, `"format": "slack"` oder `"teams"`, hat keines).

```
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"
```

Der gehostete MCP-Endpunkt hat die Lese-Tools `list_webhook_events`, `list_webhooks` und `list_webhook_deliveries`. Einen Webhook anzulegen (und damit ein Geheimnis) bleibt der Seite und der REST-API vorbehalten.

Regeln und Grenzen

- Ereignisse werden dort erfasst, wo Menschen und die API arbeiten: Aufgabenformulare und -editor, Schnellaufgabe, Kanban-Board, Erledigt-Knopf, REST-API, E-Mail-zu-Aufgabe, Kommentare, die Projekt-Genehmigung und die Urlaubsgenehmigung. Arbeit, die kein Mensch tut - der Autoclose-Job, ein Jira-Import, das Erzeugen geplanter Aufgaben -, löst **keine** Ereignisse aus, ebenso wenig die kleineren Ein-Feld-Änderungen an einer Aufgabe (Umbenennen, Typ aus einer Liste ändern), die noch nicht angebunden sind.
- Webhooks liefern mindestens einmal: Ein Empfänger kann ein Ereignis zweimal erhalten (Wiederholung nach einer Zeitüberschreitung). Nutze `X-RC-Delivery`, um eine Wiederholung zu übergehen.

- Die Reihenfolge der Ereignisse ist die ihres Entstehens, aber eine Wiederholung kann nach einem späteren Ereignis eintreffen.
- Zustellungsdatensätze bleiben 30 Tage.
- Löscht du einen Webhook, verschwindet sein Protokoll, und was noch wartete, entfällt.