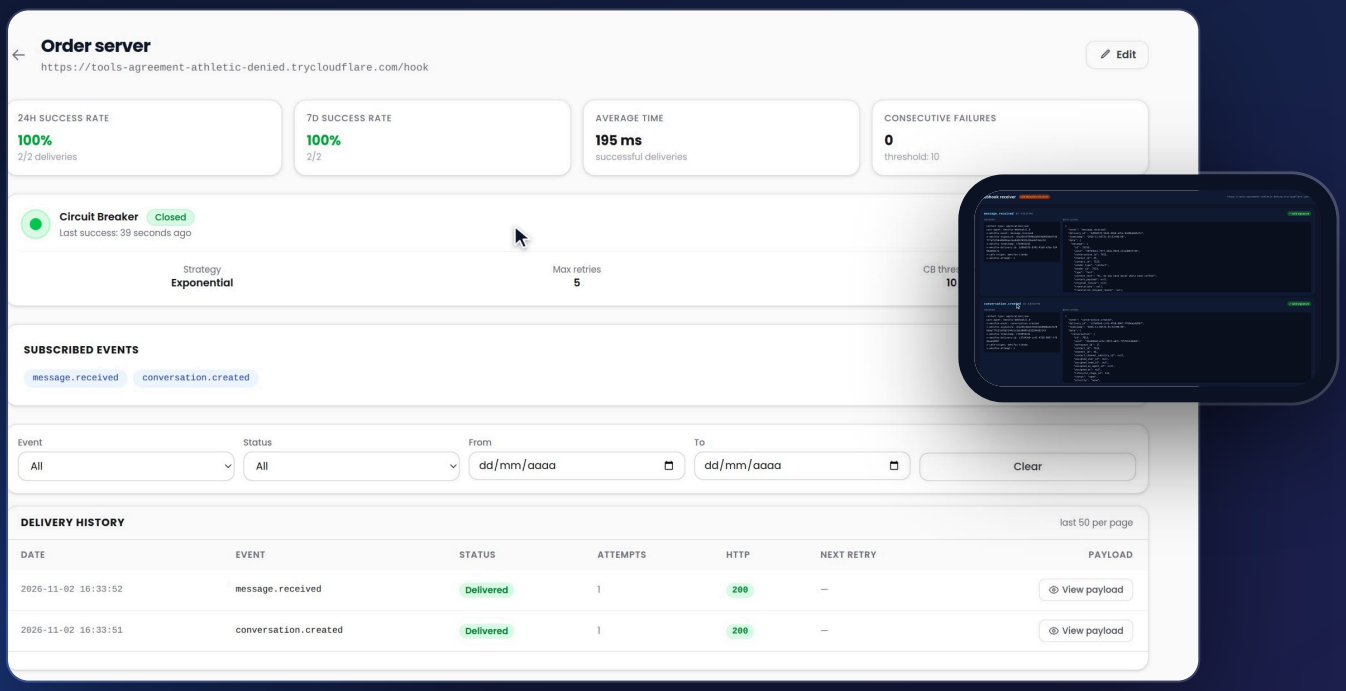
Omnifox

INTEGRATIONS AND API · COMPLETE GUIDE

Integrations and API

API keys, real calls, signed webhooks, n8n, MCP and the Node, Python and PHP SDKs.



CONTENTS

Table of contents

How to read this manual

The screenshots are real Omnifox screens, in English, with a sample business: **Café Montañero**, a fictional coffee roaster in Loja that sells coffee on its website. Everything shown was really done: the API key was created during the capture, the commands ran against Omnifox, the webhooks reached a test receiver server, and n8n and the MCP server ran their published versions. Customers are made up.

Button and field names are in **bold**. The commands use `curl` and `jq`, and two environment variables: `OMNIFOX_API_KEY` (your key) and `API=https://app.omnifox.io/api/v1`. This manual follows the video "Integraciones y API · Guía completa" on the Omnifox YouTube channel (in Spanish).

01 The ways to connect Omnifox 3
02 Requirements: plan and permissions 4
03 Create an API key 5
04 Scopes: what the key can do 6
05 Copy the key only once 7
06 The key list: edit, regenerate and delete 8
07 The API documentation 8
08 Your first call 9
09 Contacts via API 10
10 Scopes in action and errors 12
11 Conversations and messages via API 13
12 CRM and calendar via API 14
13 Pagination and limits 16
14 Webhooks: create a subscription 16
15 Signing secret, retries and headers 18
16 Test delivery 19
17 A real event 21
18 Verify the signature on your server 22
19 Delivery history 23
20 The integrations page 24
21 Custom webhook: inbound and outbound 25
22 n8n: install the Omnifox node 26
23 n8n: the Omnifox credential 27
24 n8n: reply only when they ask for the price 28
25 n8n: all the operations 30
26 MCP: let an AI operate Omnifox 31
27 Test the MCP server with the Inspector 32
28 Official SDKs: Node, Python and PHP 33
29 Security best practices 34
30 Troubleshooting 35

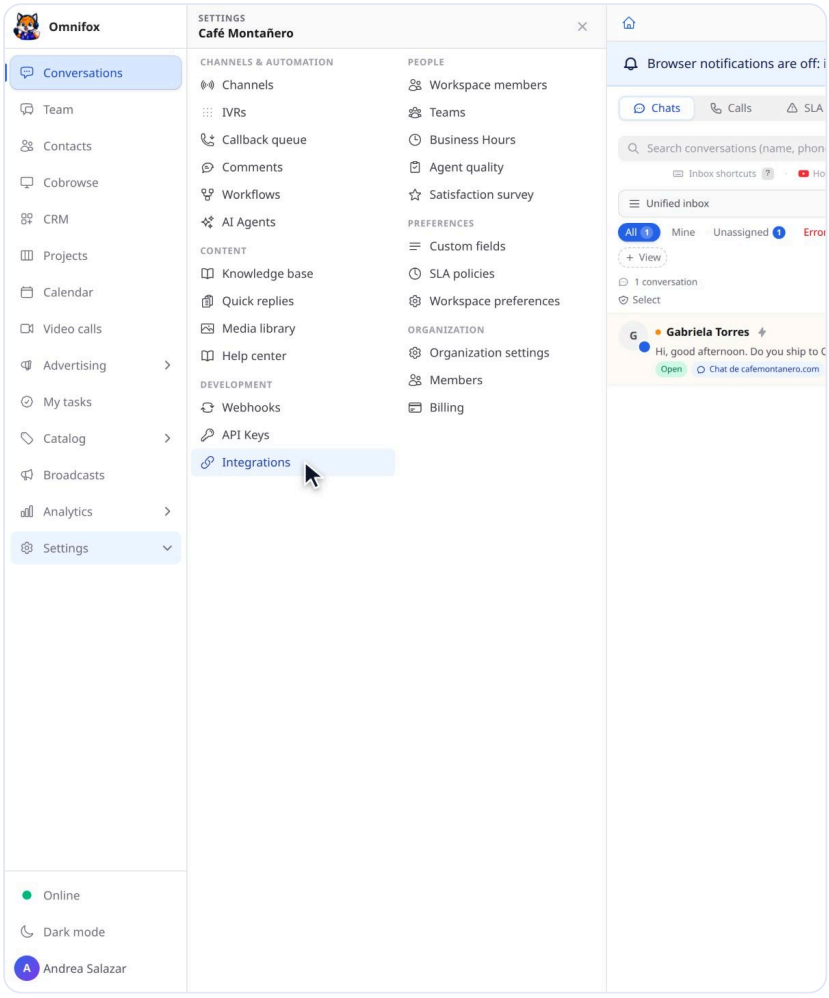
01

CHAPTER 1

The ways to connect Omnifox

Your system talks to Omnifox with the API; Omnifox notifies your system with webhooks.

Everything for developers is in **Settings → Development**: **Webhooks**, **API keys** and **Integrations**.



Settings → Development: Webhooks, API keys and Integrations.

Tool	Direction	What for
REST API	Your system → Omnifox	Create and update contacts, reply to conversations, create deals, book appointments, read reports...
Webhooks	Omnifox → your system	Get an instant notice when a message comes in, a contact is created, an appointment is booked...
Integrations	Both	Ready-made connectors: n8n, Make, Zapier, Custom webhook, Azur, Cal.com.
n8n	Both	Official Omnifox nodes (140 operations + a trigger) to automate without coding.
MCP	AI → Omnifox	Let an AI assistant (Claude, Cursor, ChatGPT...) operate Omnifox with 72 tools.
SDK	Your code → Omnifox	Official libraries for Node, Python and PHP.

TIP

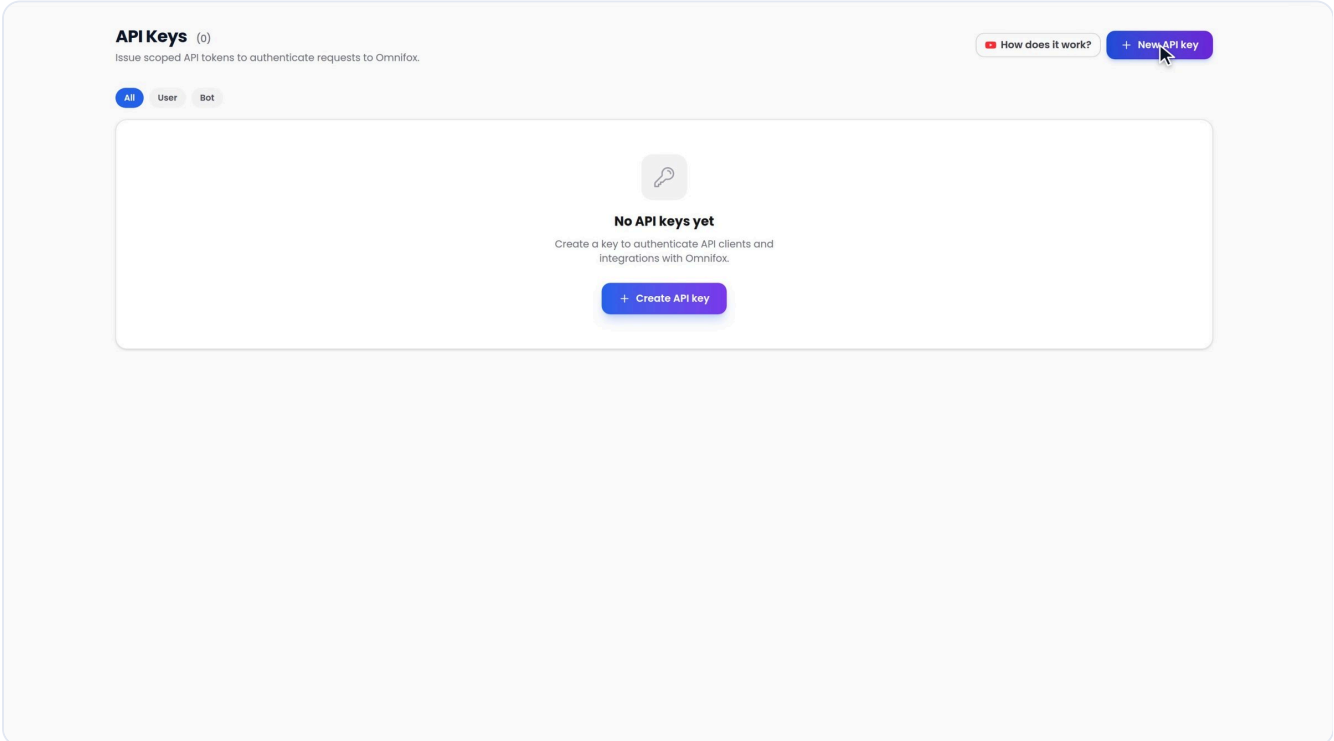
They all use the same API key. If you create one key per integration, you can revoke one without affecting the others.

02

CHAPTER 2

Requirements: plan and permissions

The API comes with Crece, Escala and Enterprise.



API keys, still empty.

- The API and webhooks are on the **Crece**, **Escala** and **Enterprise** plans. On Lite, Impulsa and Conecta, the API replies that your plan doesn't include it.
- Creating API keys requires the permission to manage the workspace settings. Webhooks can only be created by an **administrator**.
- Each key is tied to **one** workspace: whatever you do with it happens in that workspace.

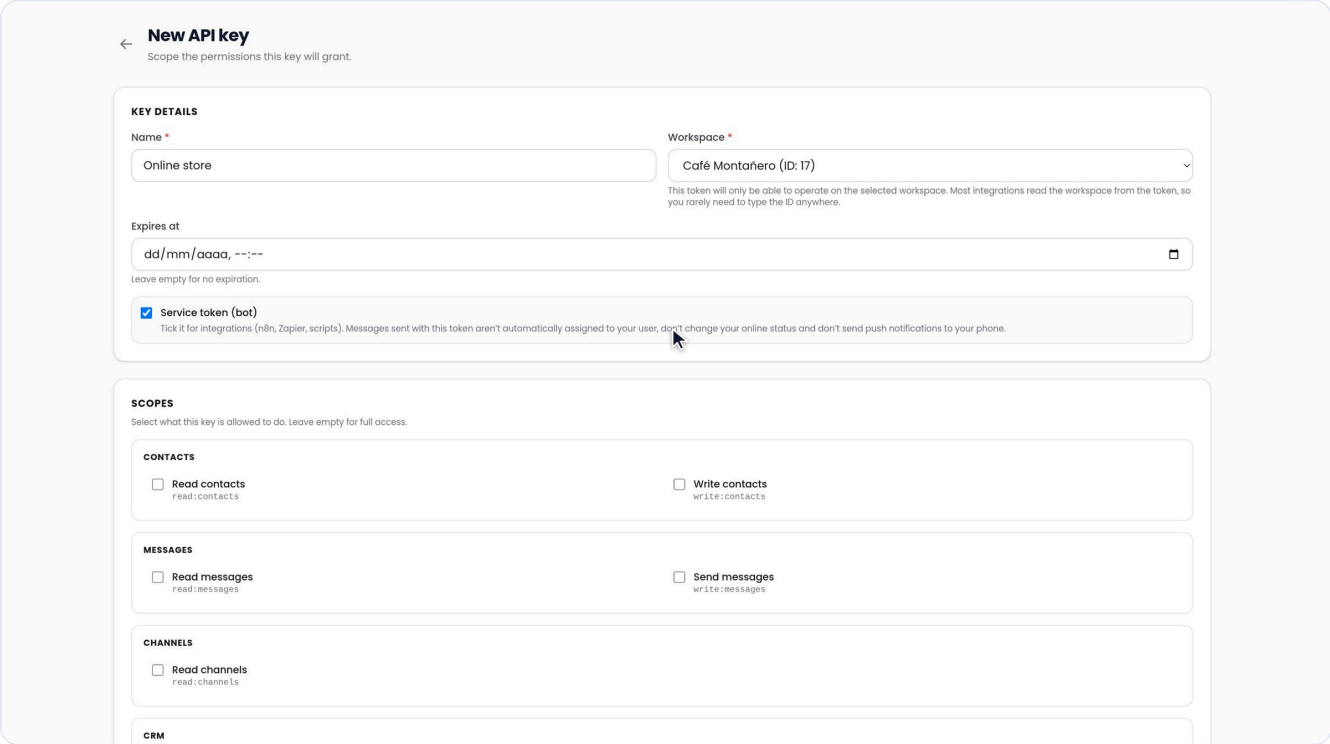
03

CHAPTER 3

Create an API key

Name, workspace, expiry and type.

1. Go to **Settings** → **API keys** and press **New API key**.
2. Type a **Name** that says what it's for (for example, "Online store").
3. Choose the **Workspace**. The key can only work in that workspace, which is why you almost never need to send the workspace number in your calls.
4. Optional: **Expires on**. Empty = never expires. For an external provider or a test, set a date.
5. Tick **Service token (bot)** for integrations.



New API key: name, workspace, expiry and service token (bot).

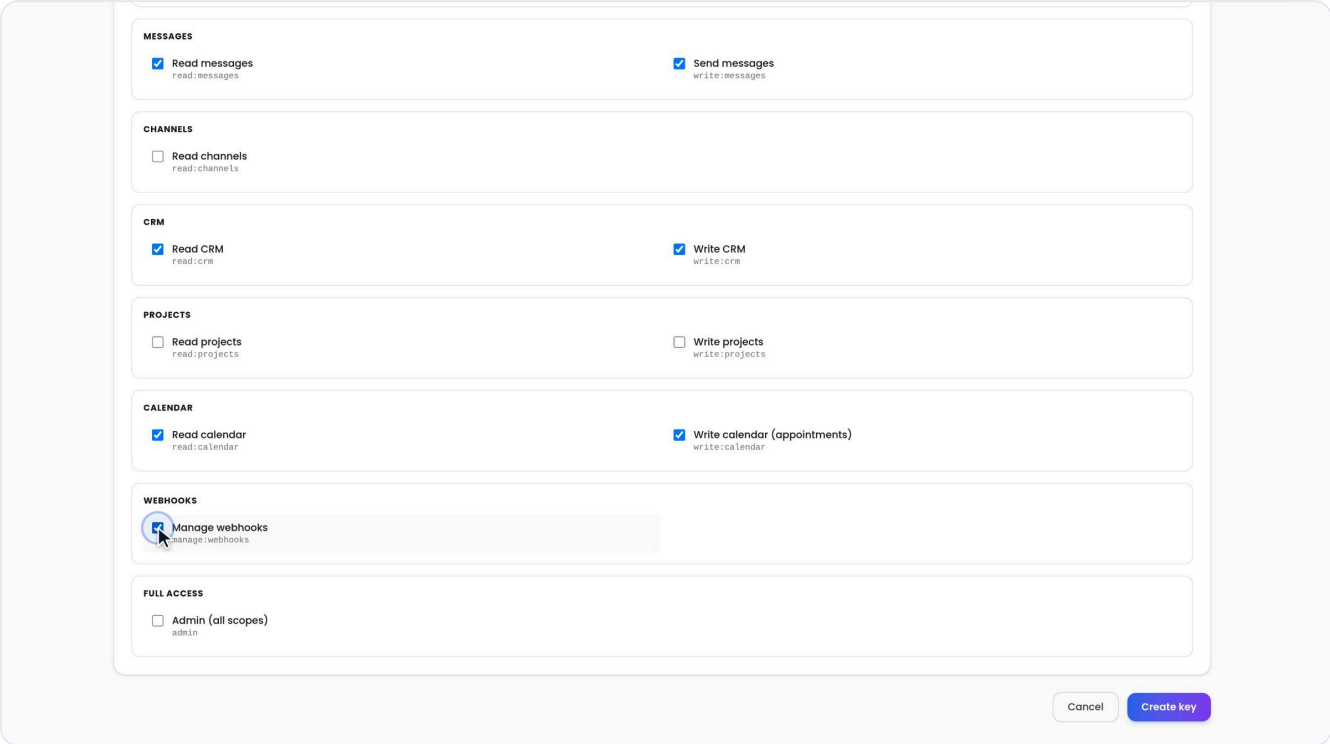
Type	Behavior
User	Acts as you: what it sends is assigned to your user and counts as your activity.
Bot (service token)	What it sends isn't assigned to your user, doesn't change your online status and doesn't send you push notifications. Recommended for n8n, Zapier, scripts and servers.

04

CHAPTER 4

Scopes: what the key can do

Give each key only what it needs.



The scopes ticked for the "Online store" key. Channels is left unticked on purpose.

Module	Scopes	Covers
Contacts	`read:contacts` · `write:contacts`	Contacts, search, contact tags, merge.
Messages	`read:messages` · `write:messages`	Conversations and messages: read, send, assign, close.
Channels	`read:channels`	See the workspace channels.
CRM	`read:crm` · `write:crm`	Pipelines, stages, deals, companies and activities.
Projects	`read:projects` · `write:projects`	Boards, items and columns.
Calendar	`read:calendar` · `write:calendar`	Calendars, services, free slots and appointments.
Webhooks	`manage:webhooks`	Create, view, rotate and delete subscriptions (used by the n8n trigger).
Full access	`admin`	All scopes.

- If you don't tick any, the key has **full access**.

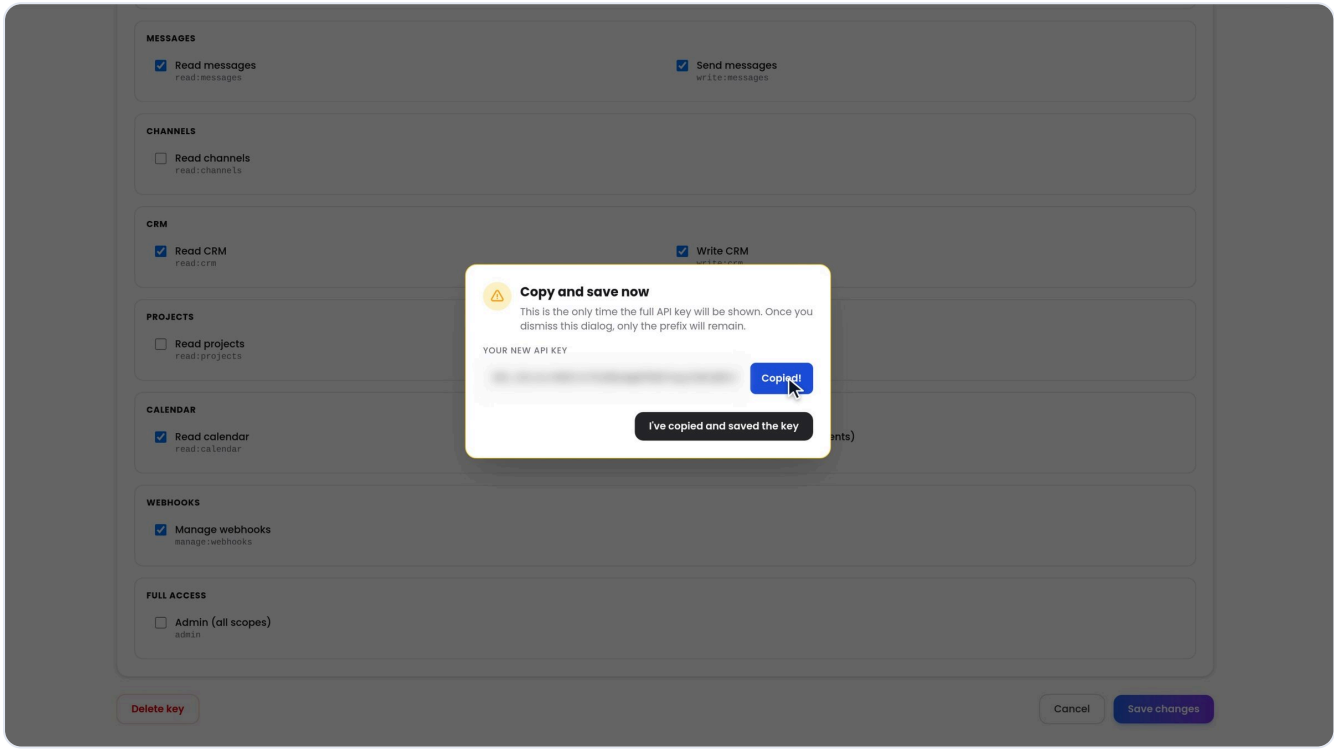
- **Write** includes **read** in the same module.
- A call without the required scope returns **403** with the missing scope (chapter 10).

05

CHAPTER 5

Copy the key only once

You see it once. Keep it safe.



The key is shown only once (blurred in the screenshots).

When you press **Create key**, Omnifox shows the full key **only once**. Press **Copy**, store it in your server's secrets manager or environment variables and confirm that you copied and saved the key. After that only its prefix is shown.

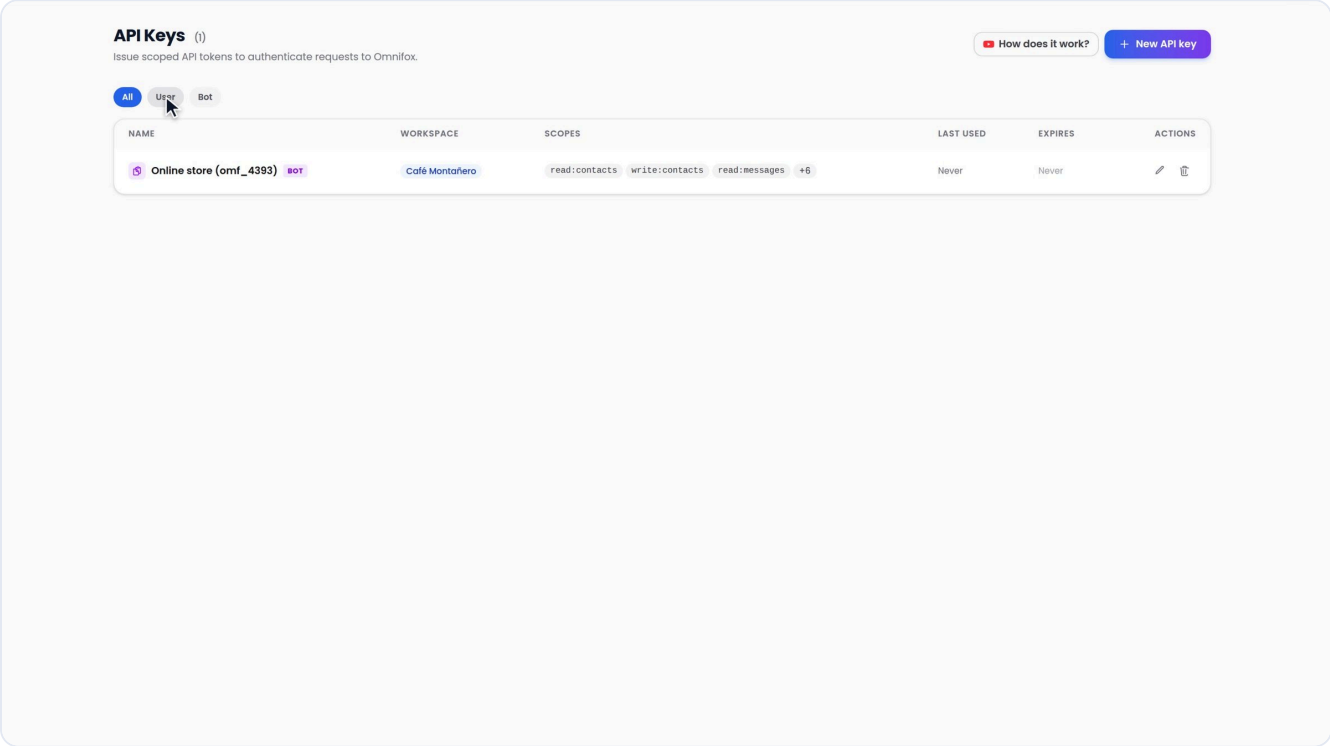
> **CAREFUL**
>
> If you lose it, you can't recover it: regenerate it (chapter 6). The previous one stops working at that moment.

06

CHAPTER 6

The key list: edit, regenerate and delete

Edit without breaking, regenerate if it leaked, delete to cut access.



The key list: type, workspace, scopes, last used and expiry.

Action	What it does
All / User / Bot filters	Shows the keys of each type.
Edit (pencil)	Changes the name, scopes or expiry. The token stays the same.
Regenerate (inside edit)	Creates a new token with the same permissions and shows it once. The previous one stops working.
Delete (trash can)	Revokes the key on the spot.
Last used	When it was last used: useful to spot abandoned keys.

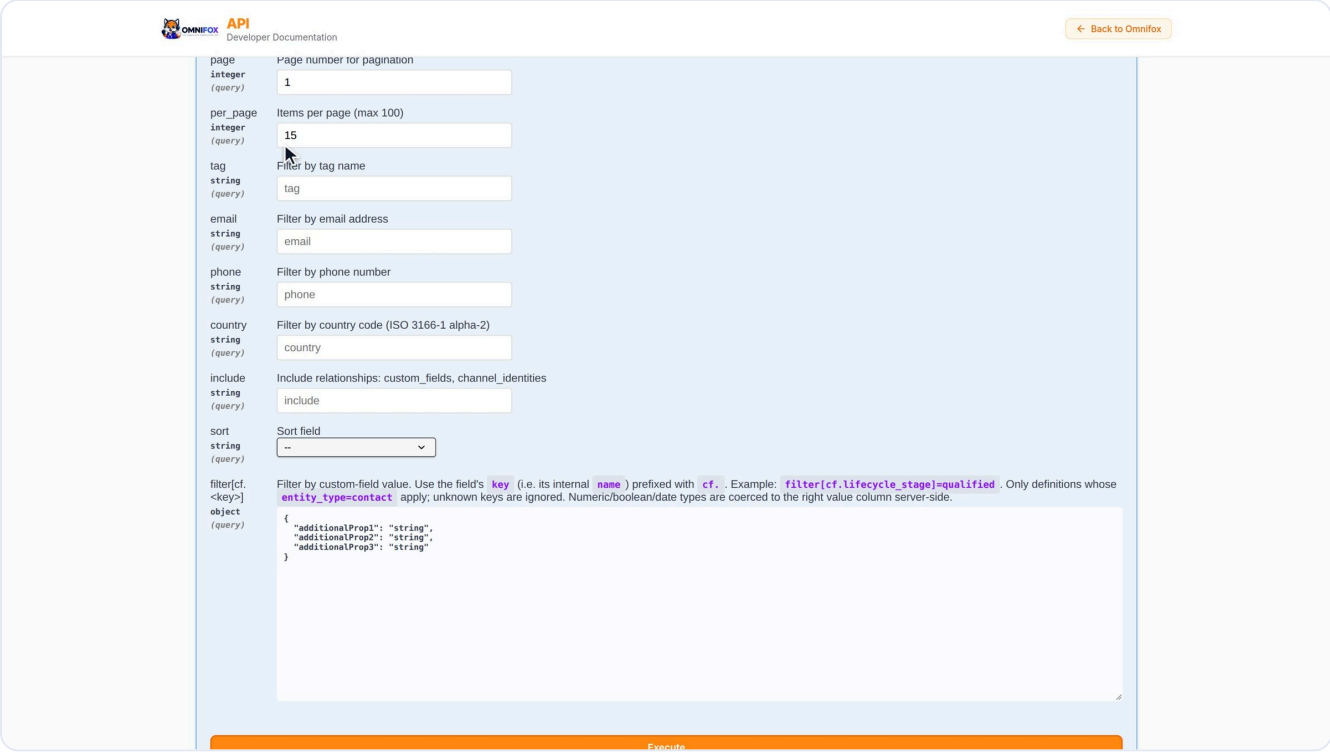
07

CHAPTER 7

The API documentation

Every endpoint, with parameters, examples and errors.

The documentation is at **omnifox.io/docs**. At the top, shortcuts to the n8n nodes, the MCP server and the SDKs. Below, the full reference grouped by module.



The API reference at /docs, with every endpoint and its example.

- **Authorize**: paste your key and try any call from the same page.
- **openapi.json**: download it for Postman, Insomnia or a code generator.

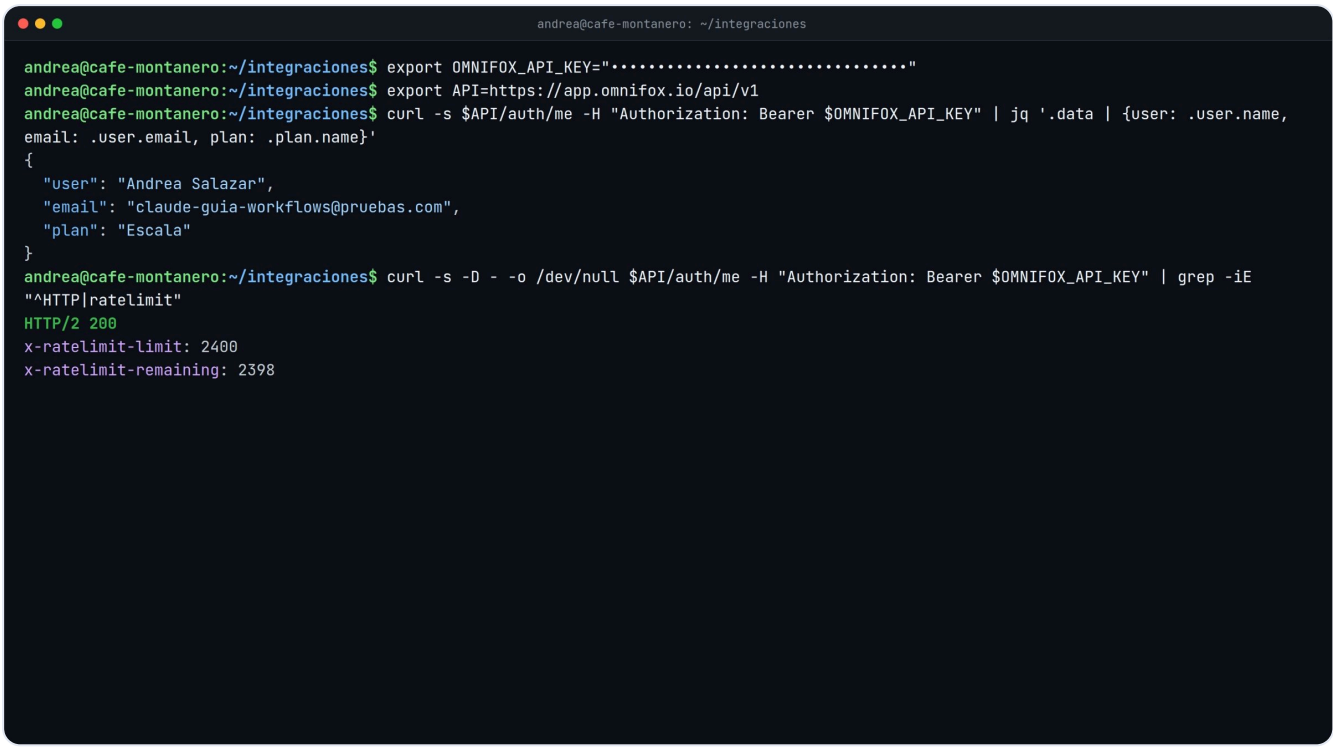
CHAPTER 8

08 Your first call

One header, and the API answers.

```
export OMNIFOX_API_KEY="your-key"
export API=https://app.omnifox.io/api/v1

curl -s $API/auth/me -H "Authorization: Bearer $OMNIFOX_API_KEY" | jq '.data | {user: .user.name, plan:
.plan.name}'
```



First call from the terminal and the rate limit headers.

All successful responses have the same shape: `success`, `message` and `data`. Lists also include pagination (chapter 13). The `X-RateLimit-Limit` and `X-RateLimit-Remaining` headers tell you how many calls per minute your plan gives you and how many you have left.

09

CHAPTER 9

Contacts via API

Create, search and update.

```
curl -s -X POST $API/contacts -H "Authorization: Bearer $OMNIFOX_API_KEY" -H "Content-Type: application/json" \
  -d '{"display_name":"Valentina Ríos","phone":"+593990710055","email":"valentina.rios@example.com","city":"Quito"}'

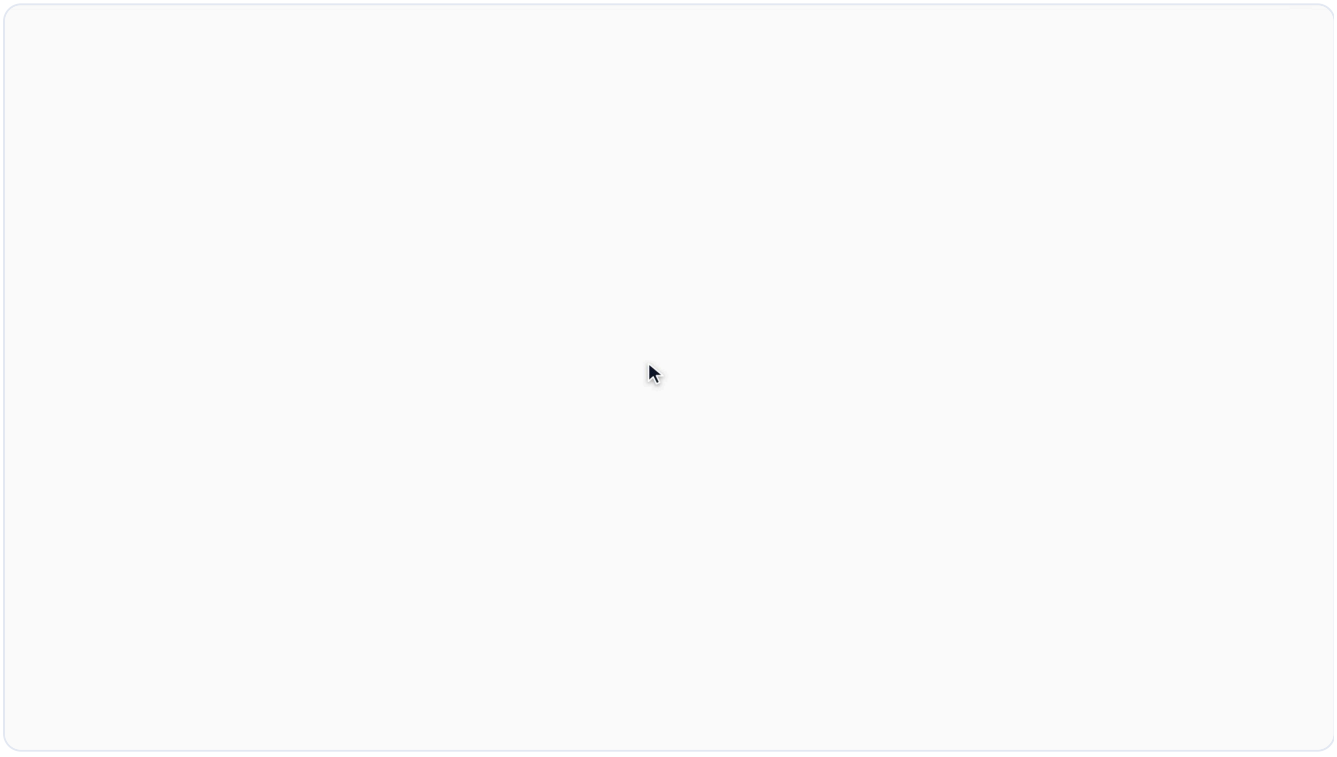
curl -s "$API/contacts/search?q=Valentina" -H "Authorization: Bearer $OMNIFOX_API_KEY"

curl -s -X PUT $API/contacts/$ID -H "Authorization: Bearer $OMNIFOX_API_KEY" -H "Content-Type: application/json" -d '{"company":"Cafetería La Ronda"}'
```

```
andrea@cafe-montanero: ~/integraciones

andrea@cafe-montanero:~/integraciones$ curl -s -X POST $API/contacts -H "Authorization: Bearer $OMNIFOX_API_KEY" -H "Content-Type:
application/json" -d '{"display_name":"Valentina
Ríos","first_name":"Valentina","last_name":"Ríos","phone":"+593990710055","email":"valentina.rios@example.com","city":"Quito"}' | jq
'.data | {id, display_name, phone, city}'
{
  "id": 7522,
  "display_name": "Valentina Ríos",
  "phone": "+593990710055",
  "city": "Quito"
}
andrea@cafe-montanero:~/integraciones$ ID=$(curl -s "$API/contacts/search?q=Valentina" -H "Authorization: Bearer $OMNIFOX_API_KEY" | jq
'.data[0].id'); echo "id: $ID"
id: 7522
andrea@cafe-montanero:~/integraciones$ curl -s -X PUT $API/contacts/$ID -H "Authorization: Bearer $OMNIFOX_API_KEY" -H "Content-Type:
application/json" -d '{"company":"Cafetería La Ronda"}' | jq '.data | {id, display_name, company}'
{
  "id": 7522,
  "display_name": "Valentina Ríos",
  "company": "Cafetería La Ronda"
}
```

Create, search and update a contact via API.



The contact created via API, in Contacts.

> **TIP**
>
> To avoid duplicating customers use `POST /contacts/upsert` : it creates or updates based on phone or email.

10

CHAPTER 10

Scopes in action and errors

Errors tell you exactly what happened.



A 403 with the missing scope, a 401 with an invalid key and a 422 validation error.

Code	When	What to do
401	The key doesn't exist, expired or was revoked/regenerated.	Check it was copied correctly; create or regenerate one.
403	A scope is missing (`required_scope`) or your plan doesn't include that feature.	Add the scope to the key or check the plan.
404	The resource doesn't exist in the key's workspace.	Check the ID and that the key belongs to the right workspace.
422	A value isn't valid.	The response includes the field and the reason.
429	You exceeded the per-minute limit.	Wait a few seconds and retry (the SDKs do it by themselves).

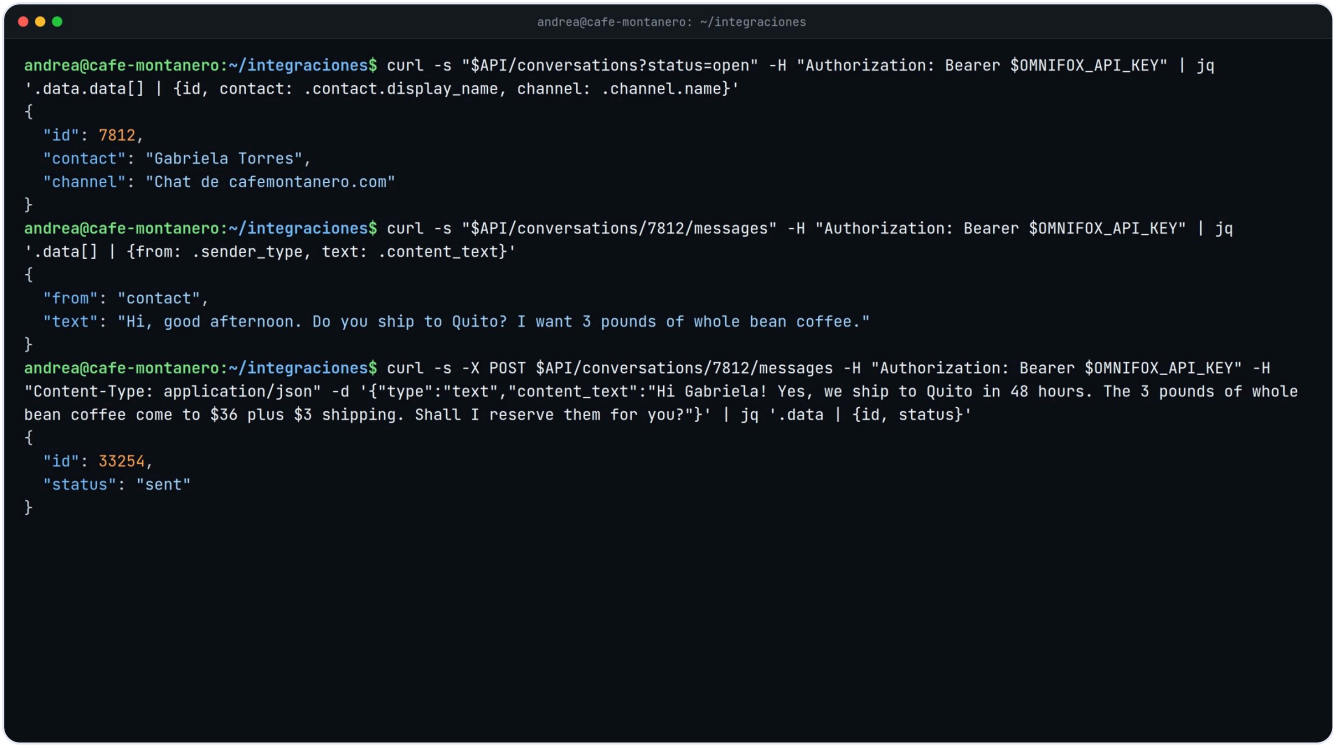
11

CHAPTER 11

Conversations and messages via API

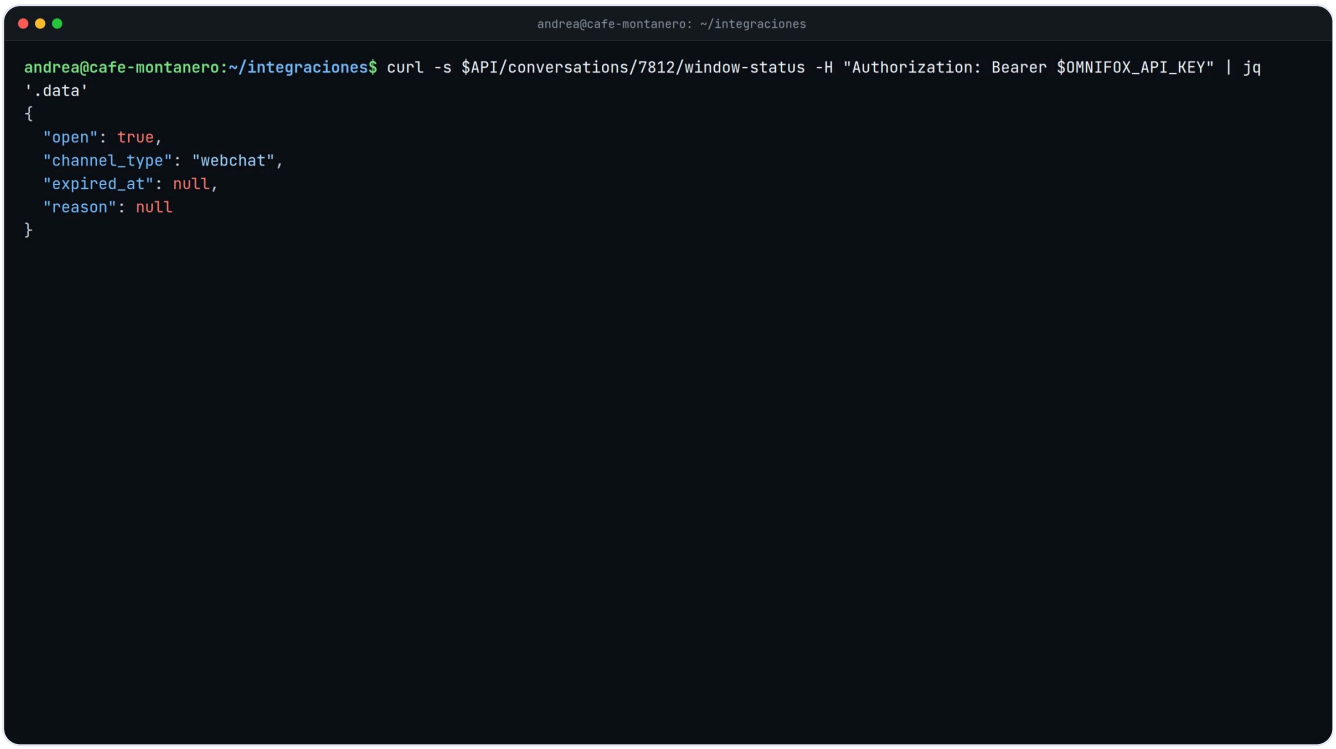
Reply to your customers from your system, through the same channel.

```
curl -s "$API/conversations?status=open" -H "Authorization: Bearer $OMNIFOX_API_KEY"
curl -s "$API/conversations/$CID/messages" -H "Authorization: Bearer $OMNIFOX_API_KEY"
curl -s -X POST $API/conversations/$CID/messages -H "Authorization: Bearer $OMNIFOX_API_KEY" -H
"Content-Type: application/json" \
  -d '{"type":"text","content_text":"Hi Gabriela! Yes, we ship to Quito in 48 hours."}'
curl -s $API/conversations/$CID/window-status -H "Authorization: Bearer $OMNIFOX_API_KEY"
```



Read the conversation and reply via API.

Omnifox sends the reply through the channel the customer wrote from (in the example, the website chat, where the reply appears instantly).



The conversation's 24-hour window (window-status).

> **CAREFUL**
>
> On WhatsApp you can only send free text within the 24-hour window since the customer's last message. Check `window-status` before sending: if it's closed, use an approved template.

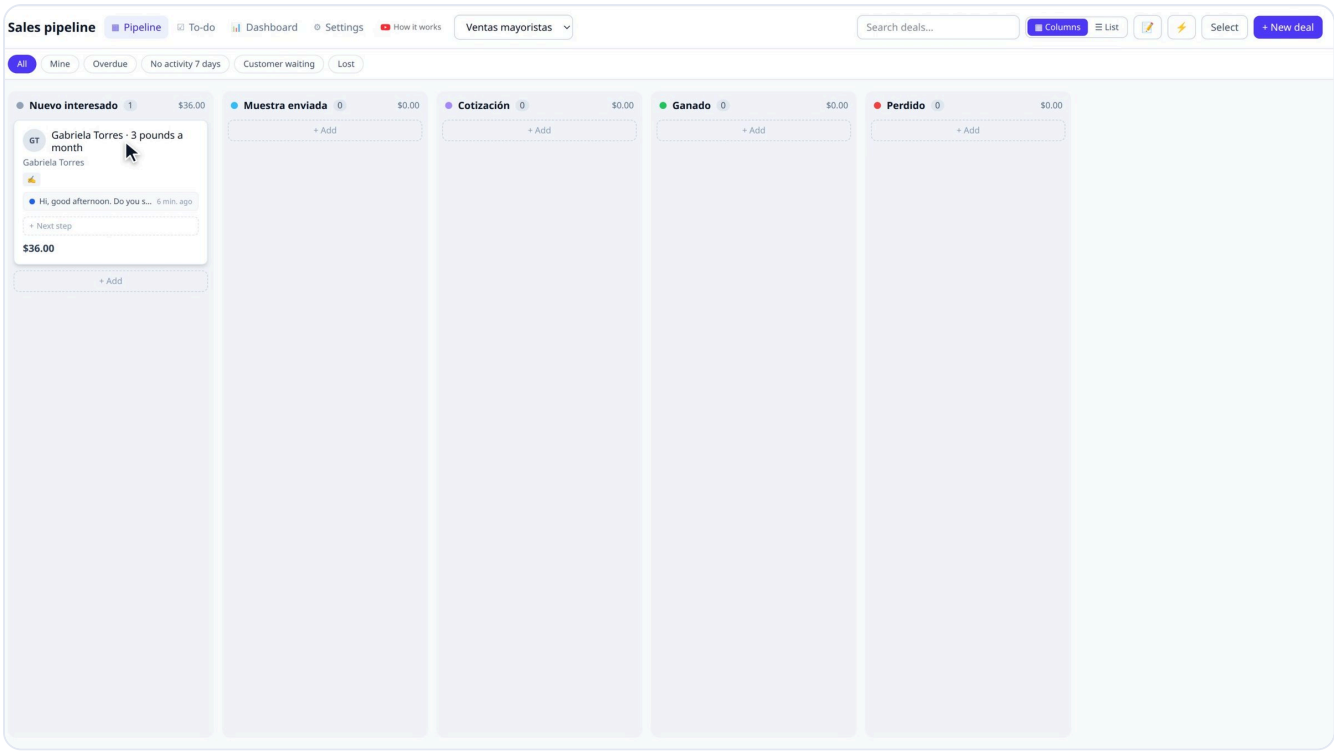
CHAPTER 12

12 CRM and calendar via API

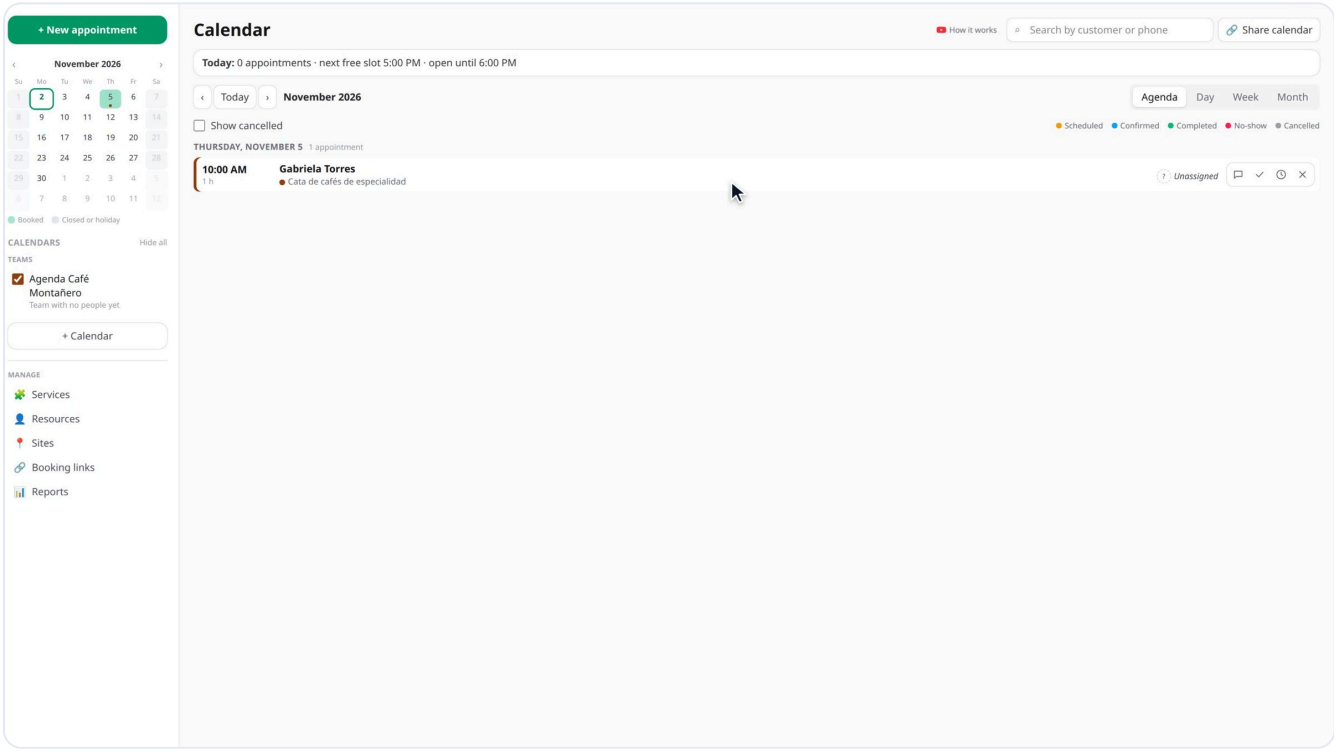
The API reaches the whole platform.

```
curl -s -X POST $API/crm/deals -H "Authorization: Bearer $OMNIFOX_API_KEY" -H "Content-Type:
application/json" \
  -d '{"title":"Gabriela Torres · 3 pounds a
month","pipeline_id":62,"stage_id":266,"amount":36,"contact_id":5051}}'

curl -s "$API/calendar/slots?calendar_id=139&service_id=141&from=2026-11-05&to=2026-11-05" -H
"Authorization: Bearer $OMNIFOX_API_KEY"
curl -s -X POST $API/calendar/appointments -H "Authorization: Bearer $OMNIFOX_API_KEY" -H "Content-
Type: application/json" \
  -d '{"calendar_id":139,"service_id":141,"contact_id":5051,"starts_at":"2026-11-05T10:00:00-05:00"}'
```



The deal created via API, in the CRM pipeline.



The appointment booked via API, in the calendar.

Appointments can also be rescheduled (`PUT /calendar/appointments/{id}`) and cancelled (`POST /calendar/appointments/{id}/cancel`). The same goes for projects, catalog, orders, reports, teams, tags and broadcasts: it's all in /docs.

13

CHAPTER 13

Pagination and limits

Go through the pages and respect the limit.

```
curl -s "$API/contacts?per_page=4&page=2" -H "Authorization: Bearer $OMNIFOX_API_KEY"
```



Pagination: page, per page, total and last page.

The response includes `current_page`, `per_page`, `total` and `last_page`. Keep going until `last_page`. If you get **429**, wait and retry.

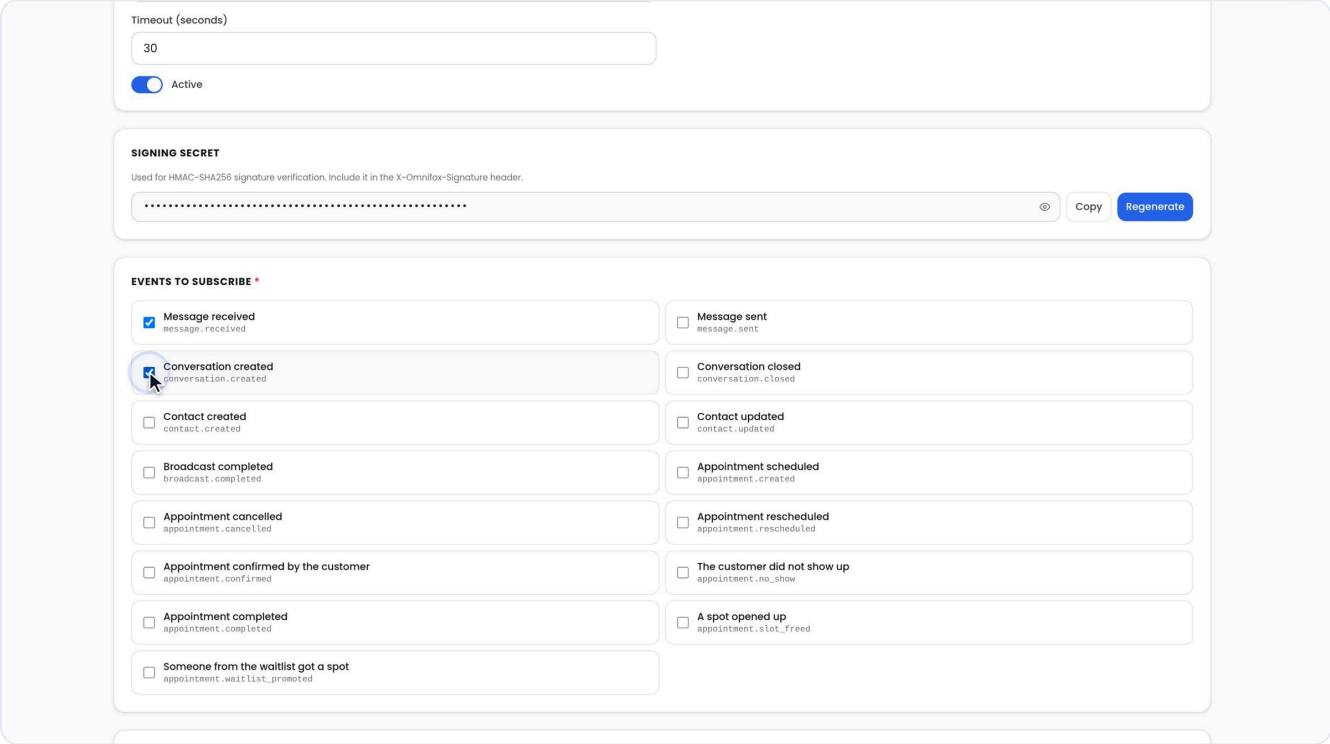
14

CHAPTER 14

Webhooks: create a subscription

Let Omnifox notify you instantly.

1. Go to **Settings** → **Webhooks** and press **New webhook**.
2. Type a **Name** and your server's **Destination URL** (it must be **https** and public).
3. Tick the **Events to subscribe**.



New webhook: name, https URL and events.

Event	When it arrives
`message.received`	A customer message comes in.
`message.sent`	Your team, the AI or a workflow sends a message.
`conversation.created` · `conversation.closed`	A conversation opens or closes.
`contact.created` · `contact.updated`	A contact is created or changes.
`broadcast.completed`	A broadcast finished.
`appointment.*`	Appointment booked, cancelled, rescheduled, confirmed by the customer, no-show, attended, a slot freed up, someone came in from the waiting list.

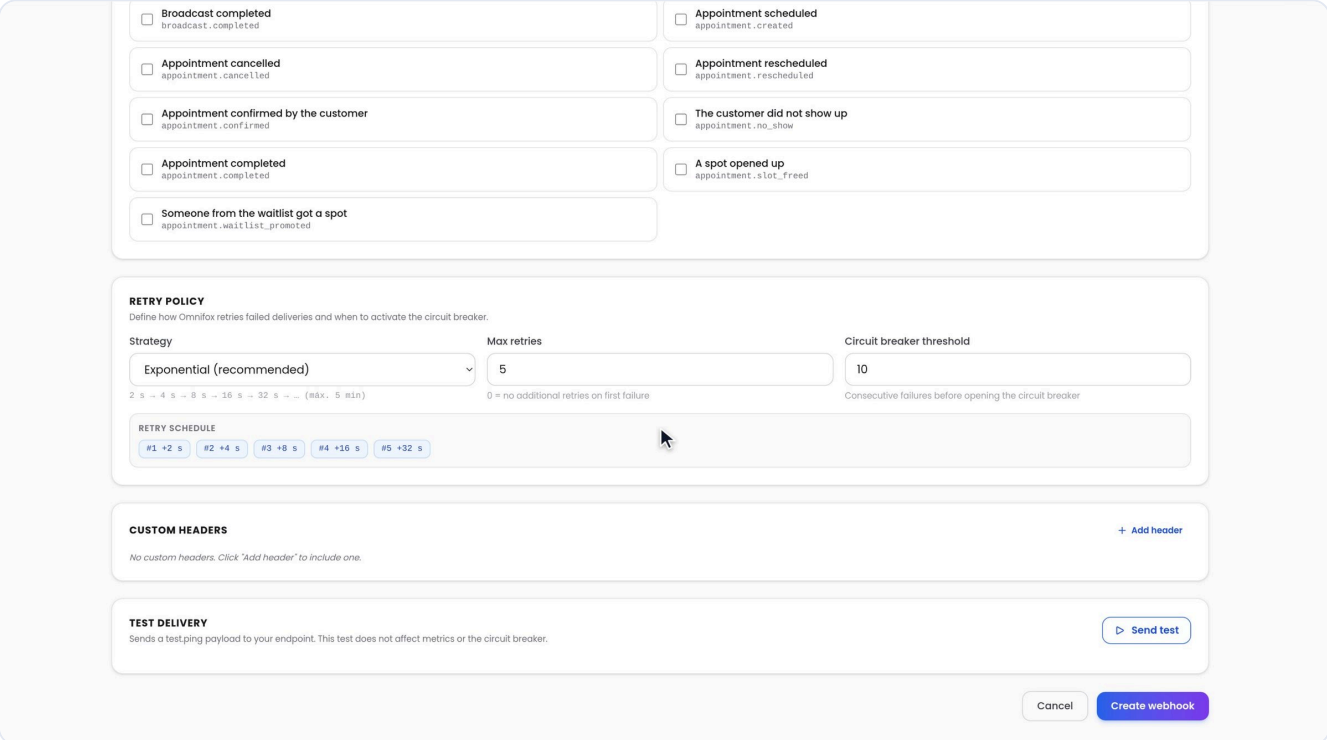
15

CHAPTER 15

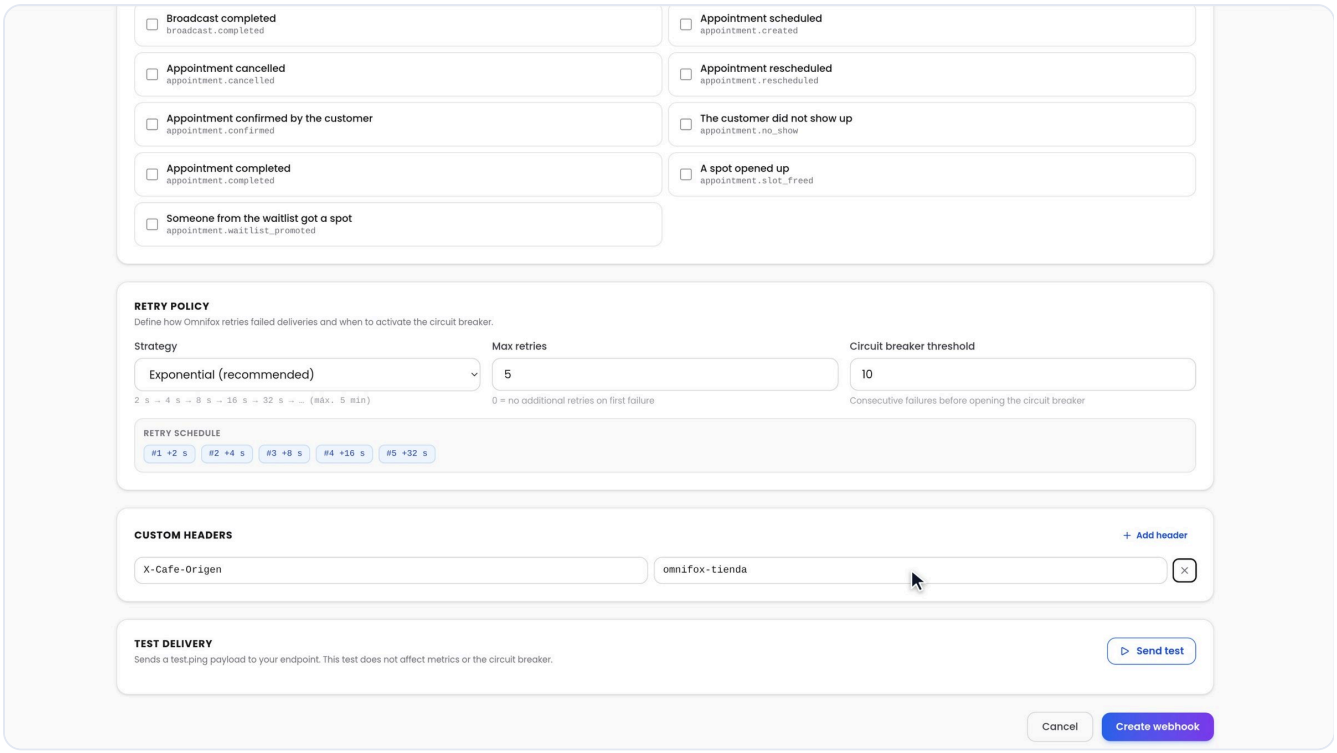
Signing secret, retries and headers

Signed, with retries and protected.

Setting	What for
Signing secret	Omnifox signs each delivery with this secret. Copy it to your server to verify (chapter 18). **Regenerate** creates a new one.
Strategy	**Exponential** (recommended: 2 s, 4 s, 8 s, 16 s... up to 5 min), **Linear** (every 60 s) or **No retries**.
Max. retries	How many times to retry a failed delivery.
Circuit breaker threshold	After that many consecutive failures, Omnifox stops sending for a while and then tries again.
Timeout	Seconds to wait for your server's response.
Custom headers	Your own headers on each delivery (for example, a key on your side).



Retry policy and retry schedule.



Custom headers.

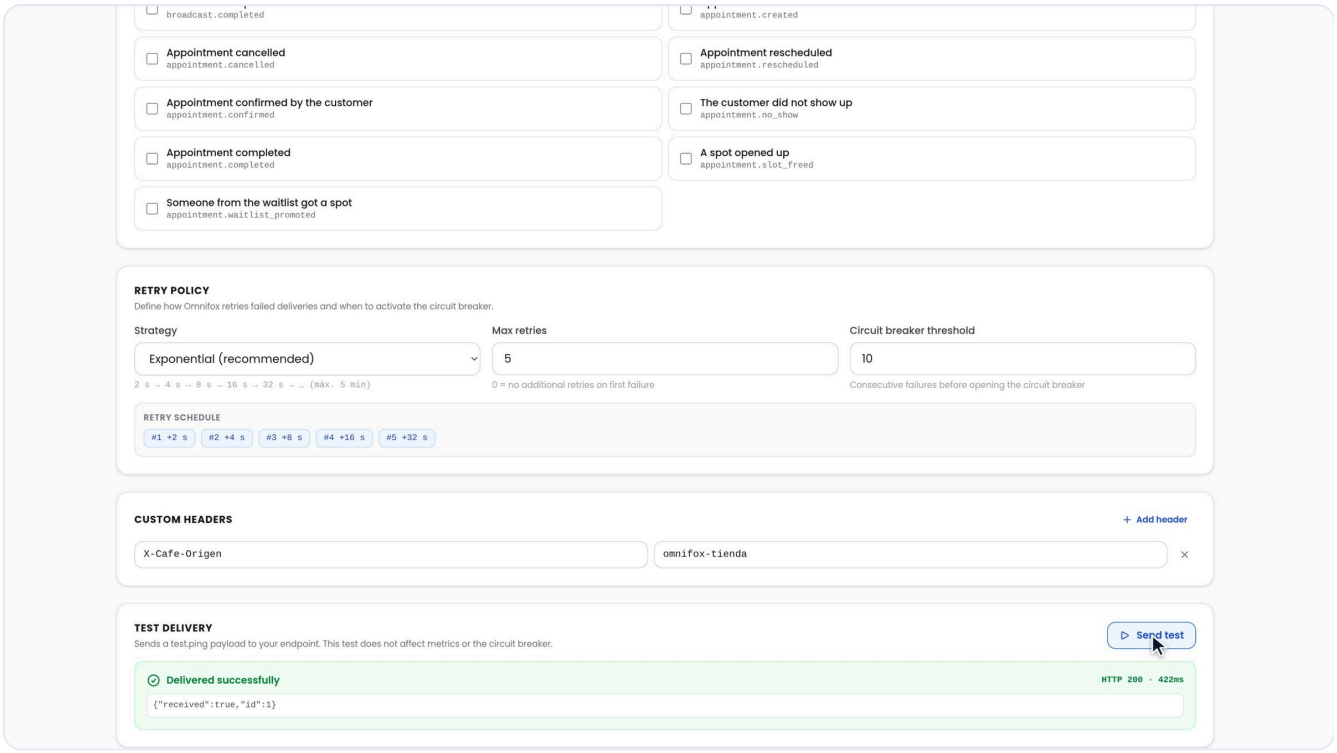
16

CHAPTER 16

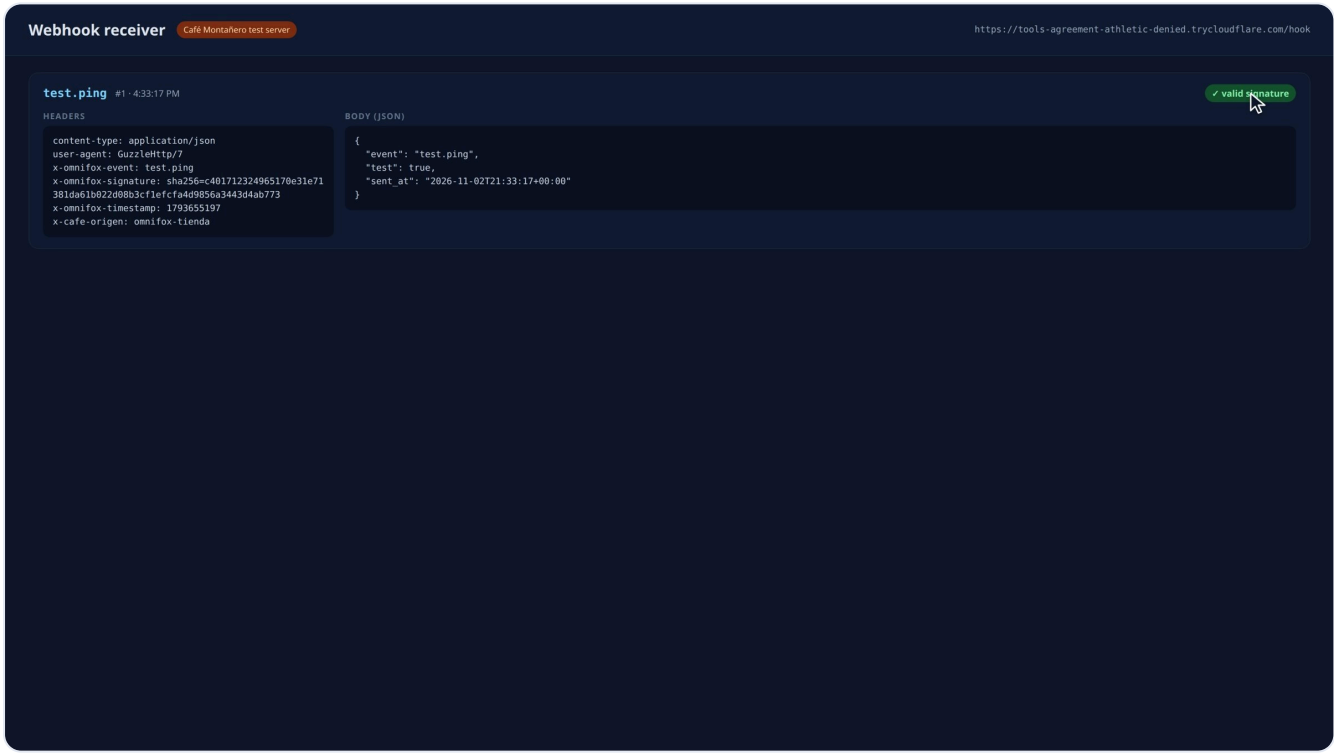
Test delivery

Test before saving.

Send test sends a `test.ping` event to your URL and shows the HTTP code, the time and the response. It doesn't count toward the metrics or the circuit breaker.



Test delivery result.



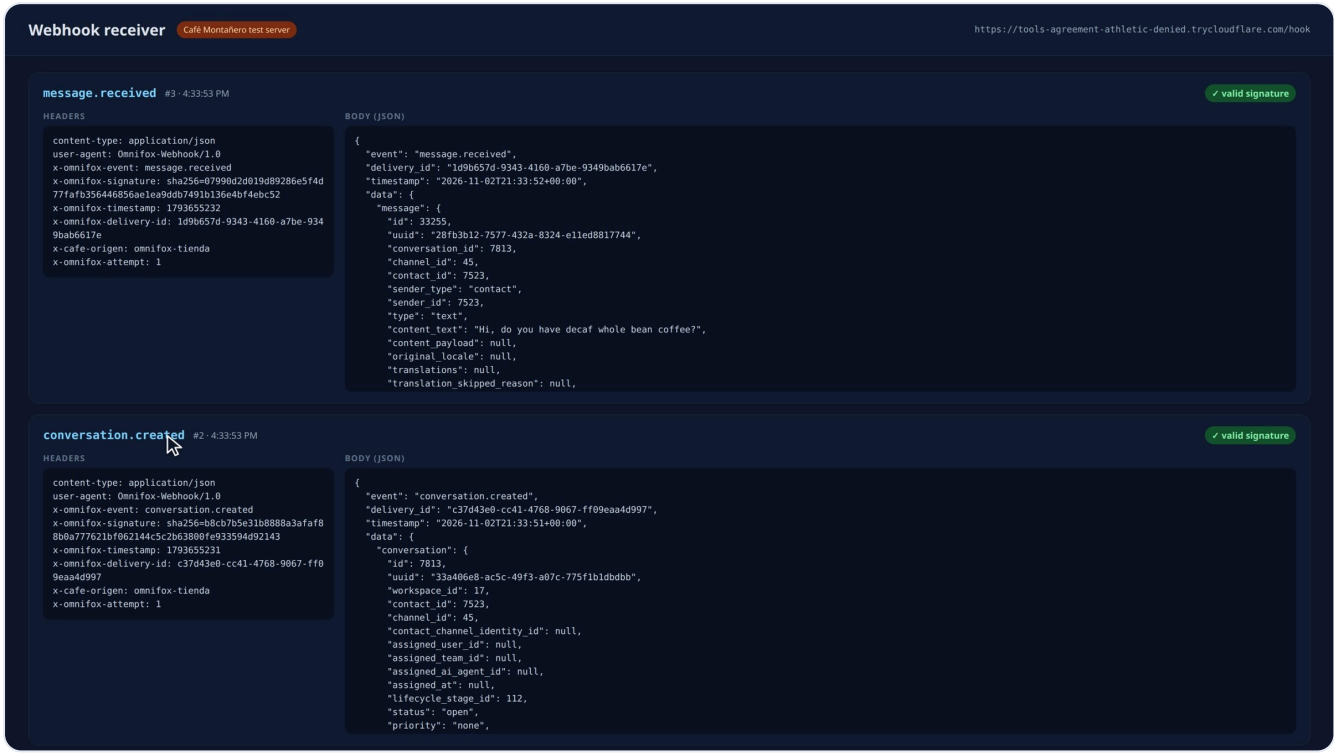
The ping on the receiver, with the signature verified.

17

CHAPTER 17

A real event

A customer writes and your server finds out.



The message.received and conversation.created events on the receiver.

Header	Content
`X-Omnifox-Event`	The event (`message.received` ...).
`X-Omnifox-Delivery-Id`	Unique delivery identifier: store it so you don't process the same event twice.
`X-Omnifox-Attempt`	Attempt number (1 the first time).
`X-Omnifox-Timestamp`	Unix timestamp of the signature.
`X-Omnifox-Signature`	`sha256=` + HMAC-SHA256 of "timestamp.body".

The body includes `event` , `delivery_id` , `timestamp` and `data` (in `message.received` : the message with `content_text` , the contact, the conversation and the channel).

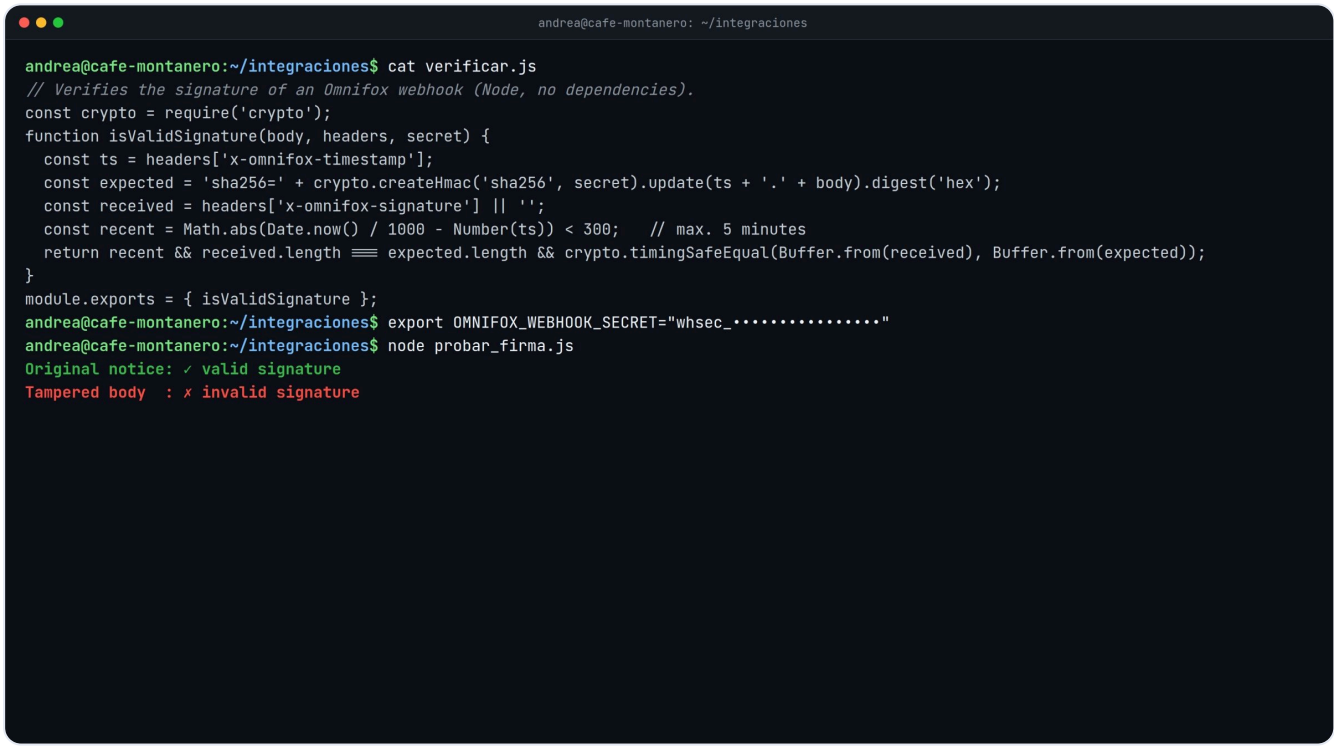
18

CHAPTER 18

Verify the signature on your server

Ten lines to know the notice is authentic.

```
const crypto = require('crypto');
function isValidSignature(body, headers, secret) {
  const ts = headers['x-omnifox-timestamp'];
  const expected = 'sha256=' + crypto.createHmac('sha256', secret).update(ts + '.' +
body).digest('hex');
  const received = headers['x-omnifox-signature'] || '';
  const recent = Math.abs(Date.now() / 1000 - Number(ts)) < 300;   // max. 5 minutes
  return recent && received.length === expected.length &&
         crypto.timingSafeEqual(Buffer.from(received), Buffer.from(expected));
}
```



Signature verification: the original notice passes, the tampered one doesn't.

CAREFUL

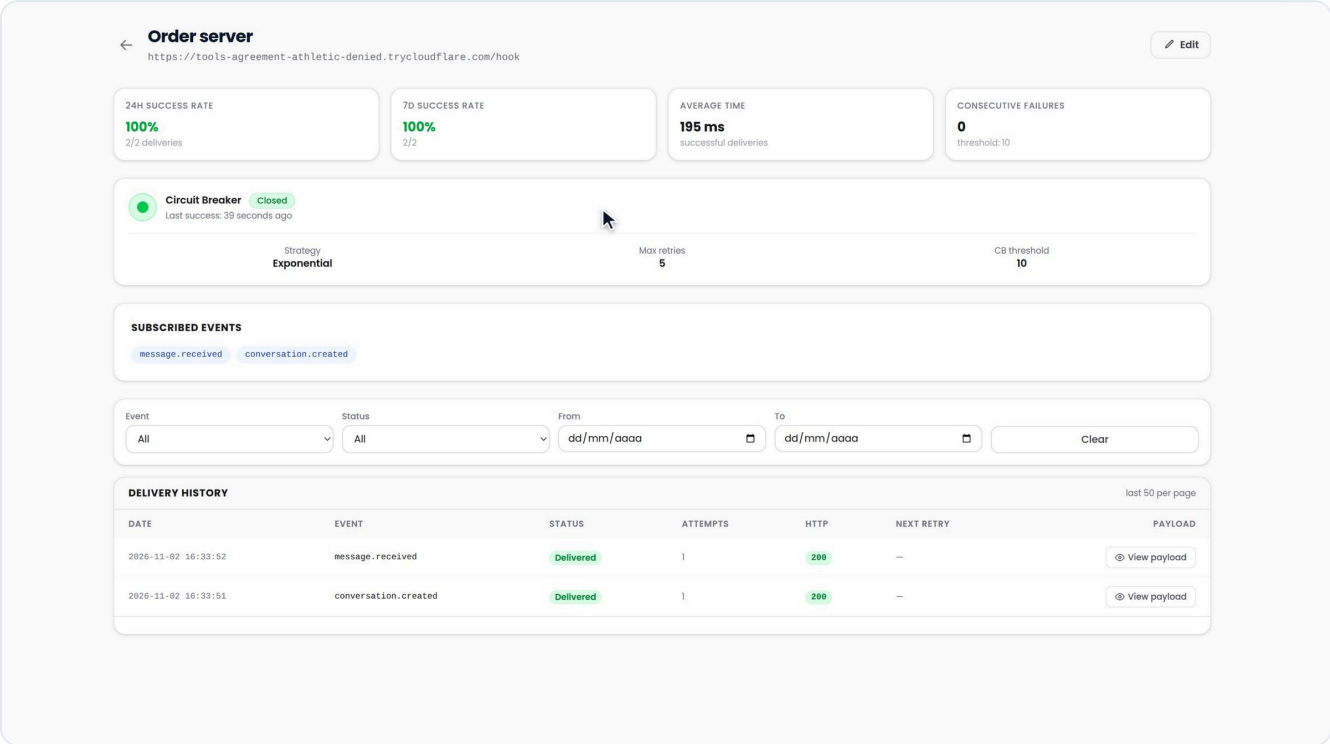
Use the EXACT body you received (without re-serializing the JSON) and a constant-time comparison.

19

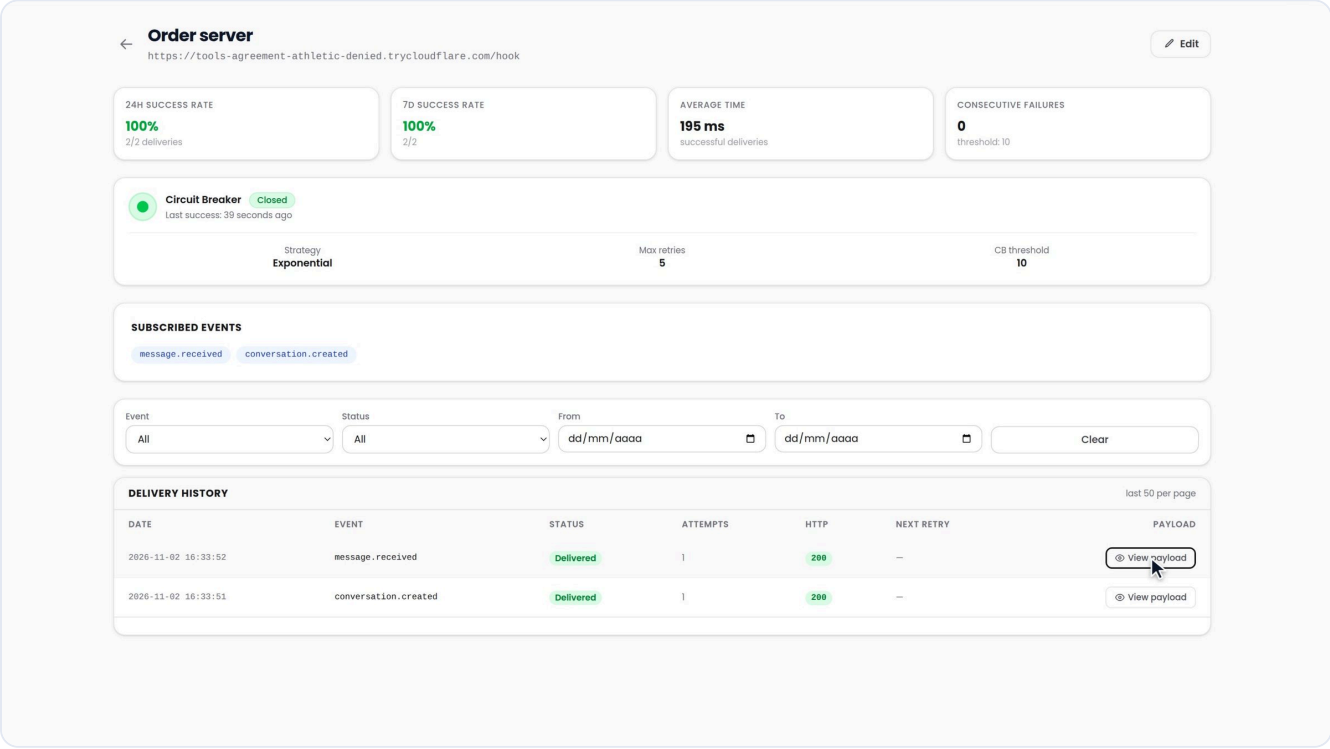
CHAPTER 19

Delivery history

What was sent, what your server replied and when.



Webhook detail: success rates, average time and consecutive failures.



The delivery history and the payload of one delivery.

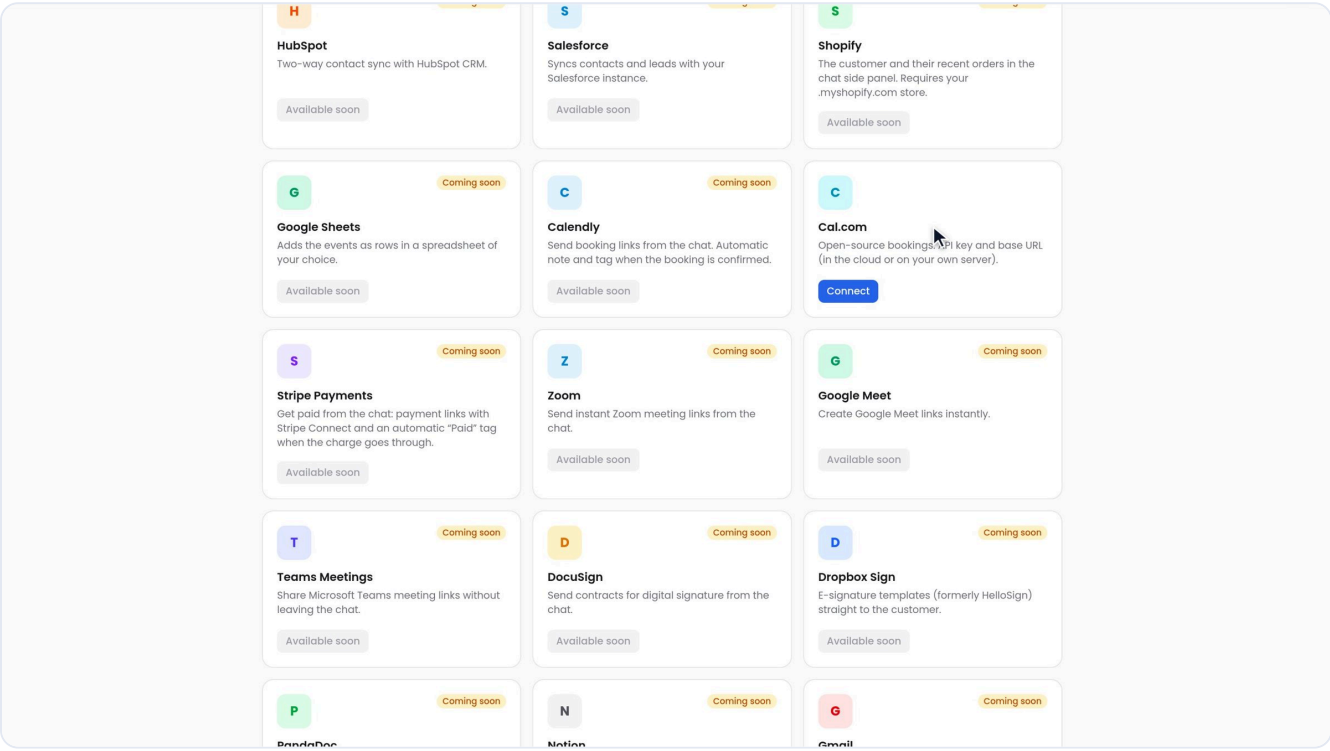
- Success rate for 24 hours and 7 days, average time and consecutive failures.

- **Circuit breaker** status; if it was left open, force it closed once your server is back.
- **Delivery history** with filters by event, status and date; **View payload** shows what was sent and the response.
- From the list: pause, test, regenerate the secret, edit or delete.

CHAPTER 20

20 The integrations page

Ready-made connectors.



The integrations page.

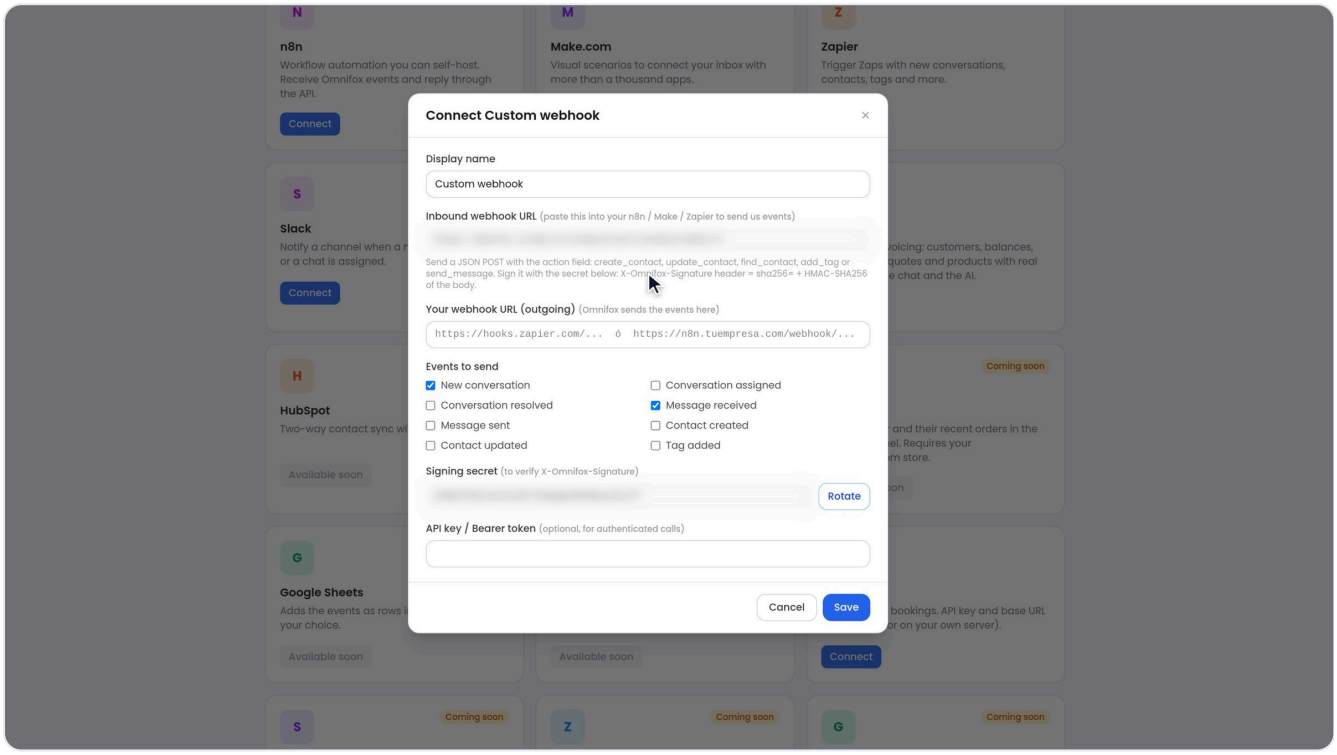
Integration	How it works
n8n, Make, Zapier, Custom webhook	Webhooks in both directions: an inbound URL with actions and signed outbound events (chapter 21).
Azur	Ecuador's electronic invoicing: customers, balances, invoices, quotes and products, from the chat and the AI.
Cal.com	Your Cal.com bookings in the Omnifox calendar.
Coming soon	HubSpot, Salesforce, Shopify, Google Sheets, Calendly, Stripe Payments, Zoom, Google Meet, Teams, DocuSign, Dropbox Sign, PandaDoc, Notion, Gmail and Outlook. Meanwhile: API + webhooks + n8n.

21

CHAPTER 21

Custom webhook: inbound and outbound

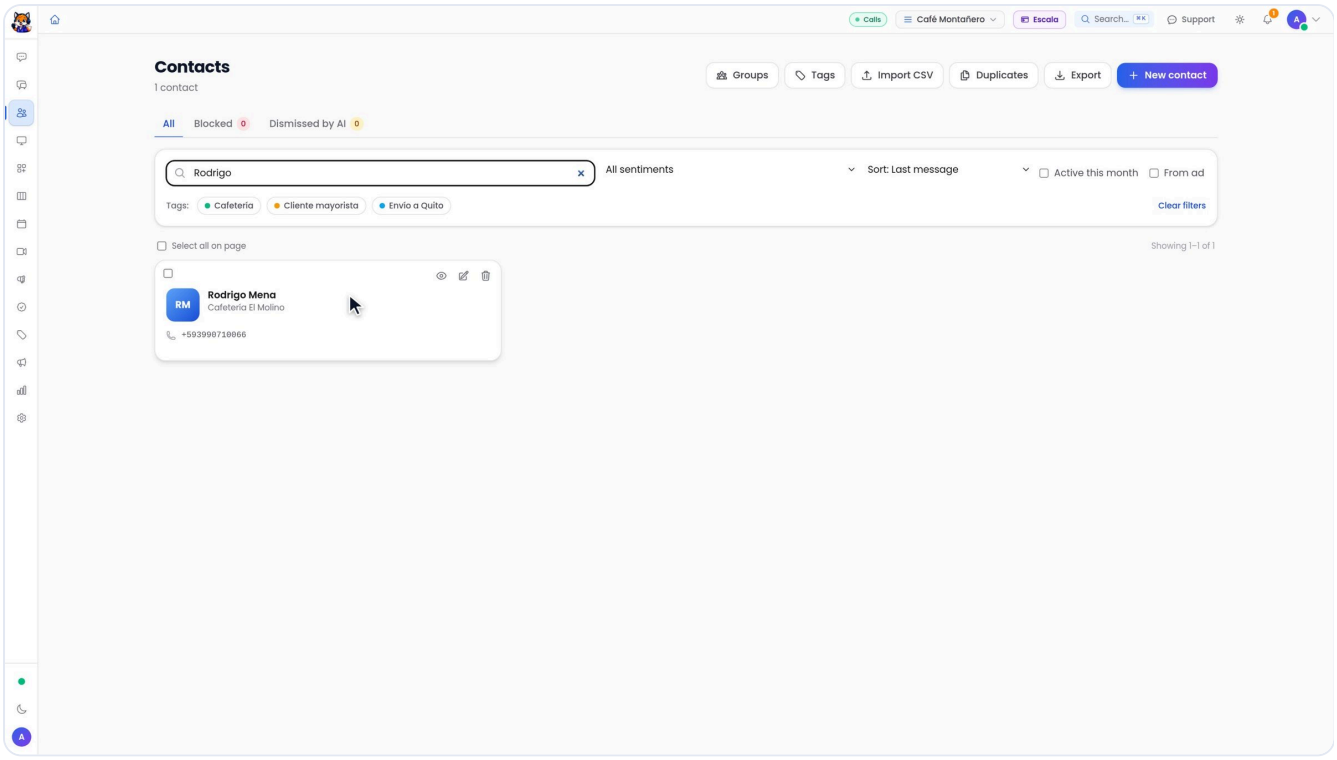
One URL to receive actions and another to send events.



Custom webhook: inbound URL, outbound URL, events and signing secret.

Inbound action (`action`)	Fields
`create_contact`	`display_name` , `email` , `phone` , `company` , `notes` , `external_id`
`update_contact`	The contact and the fields to change
`find_contact`	Phone or email
`add_tag`	Contact and tag
`send_message`	Contact or conversation and the text

```
BODY='{"action":"create_contact","display_name":"Rodrigo Mena","phone":"+593990710066"}'
SIG=$(printf '%s' "$BODY" | openssl dgst -sha256 -hmac "$SECRET" | cut -d' ' -f2)
curl -s -X POST https://omnifox.io/api/v1/integrations/custom/<organization>/<workspace> \
  -H "Content-Type: application/json" -H "X-Omnifox-Signature: sha256=$SIG" -d "$BODY"
```



A signed inbound action (create_contact) from the terminal.

The **outbound URL** and the **Events to send** create a real webhook subscription, signed with the same **Signing secret**. **Pause** stops it and **Disconnect** deletes it.

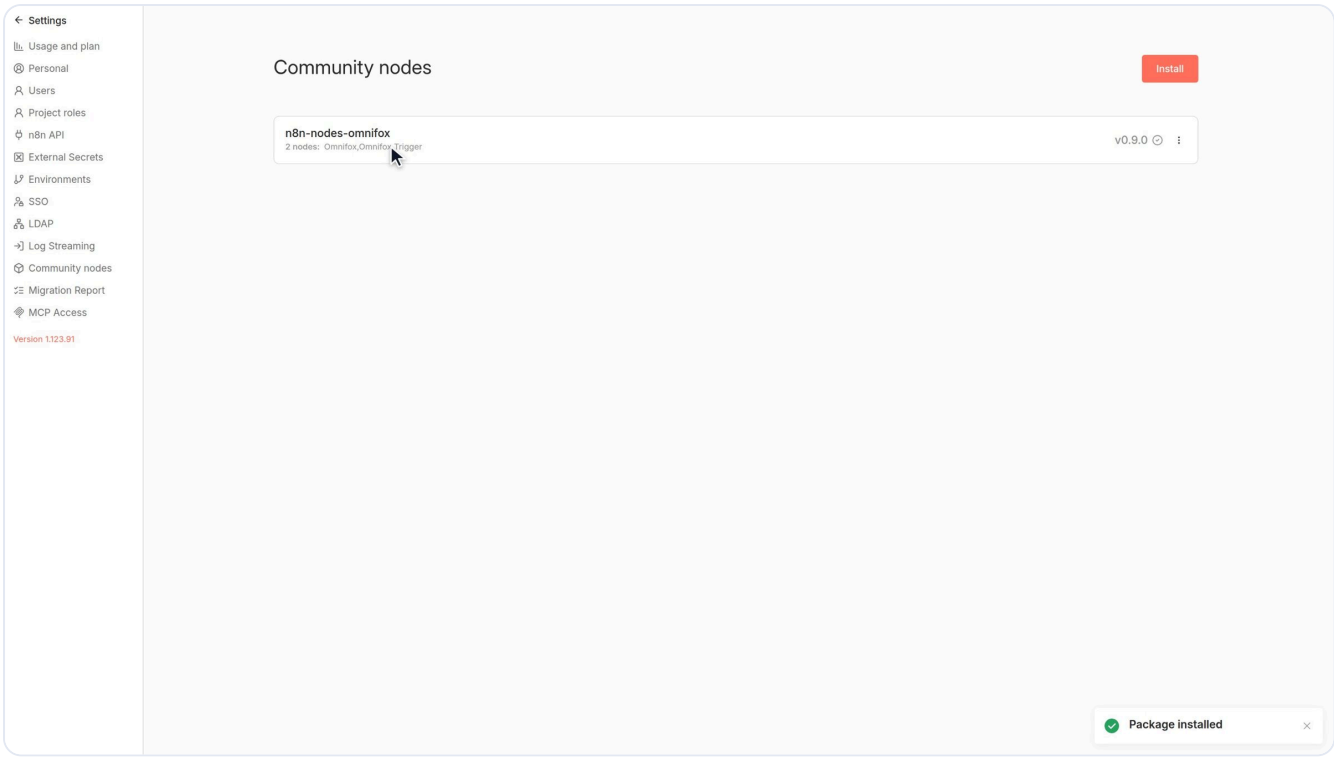
22

CHAPTER 22

n8n: install the Omnifox node

A community package, official from Omnifox.

1. In n8n go to **Settings** → **Community nodes** and press **Install**.
2. Type `n8n-nodes-omnifox`, accept the risk notice and press **Install**.
3. Two nodes appear: **Omnifox** (actions) and **Omnifox Trigger** (events).



n8n → Settings → Community nodes with n8n-nodes-omnifox installed.

NOTE

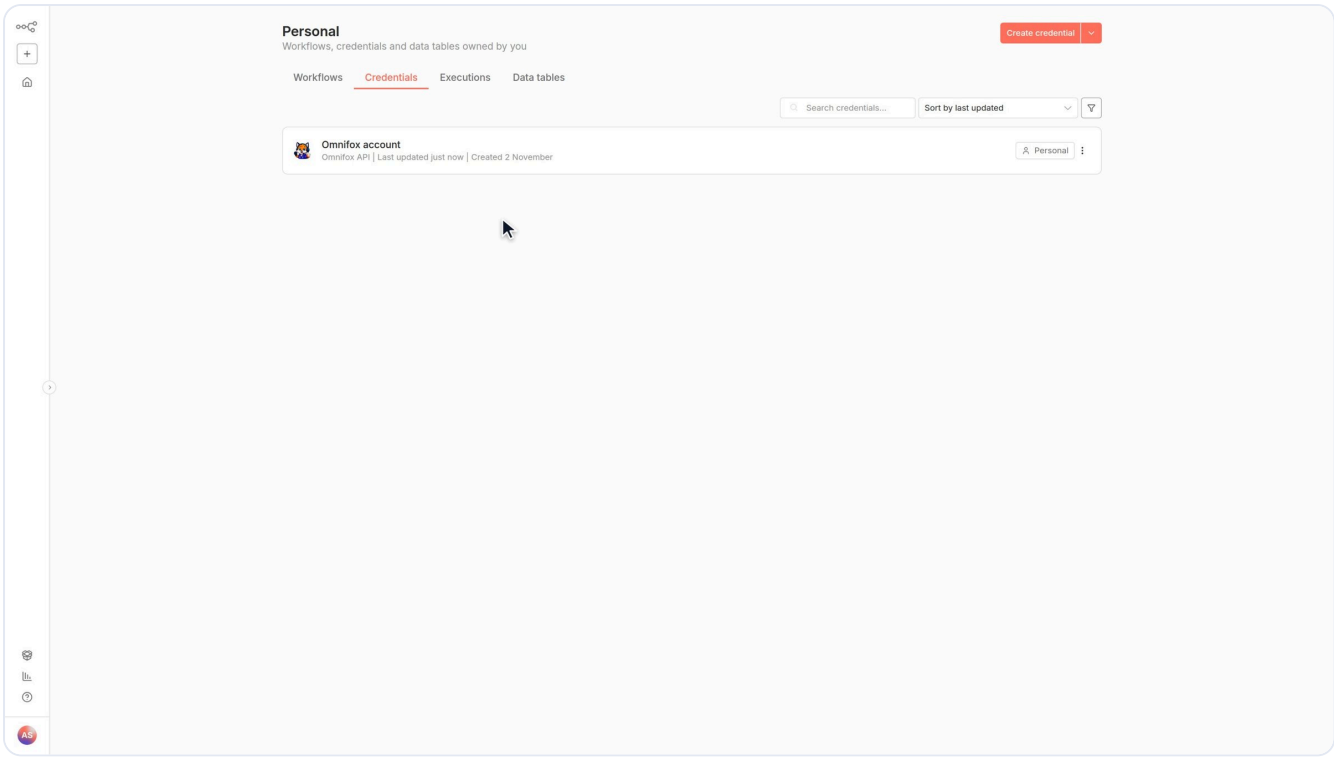
It works on n8n Cloud and on your own installation. For the trigger, your n8n must be reachable over https from the internet.

CHAPTER 23

23 n8n: the Omnifox credential

One credential, every operation.

Field	Value
API Token	Your API key (preferably a bot key just for n8n).
Workspace ID	0: taken from the key.
Base URL	`https://app.omnifox.io` (without /api: the node adds it).

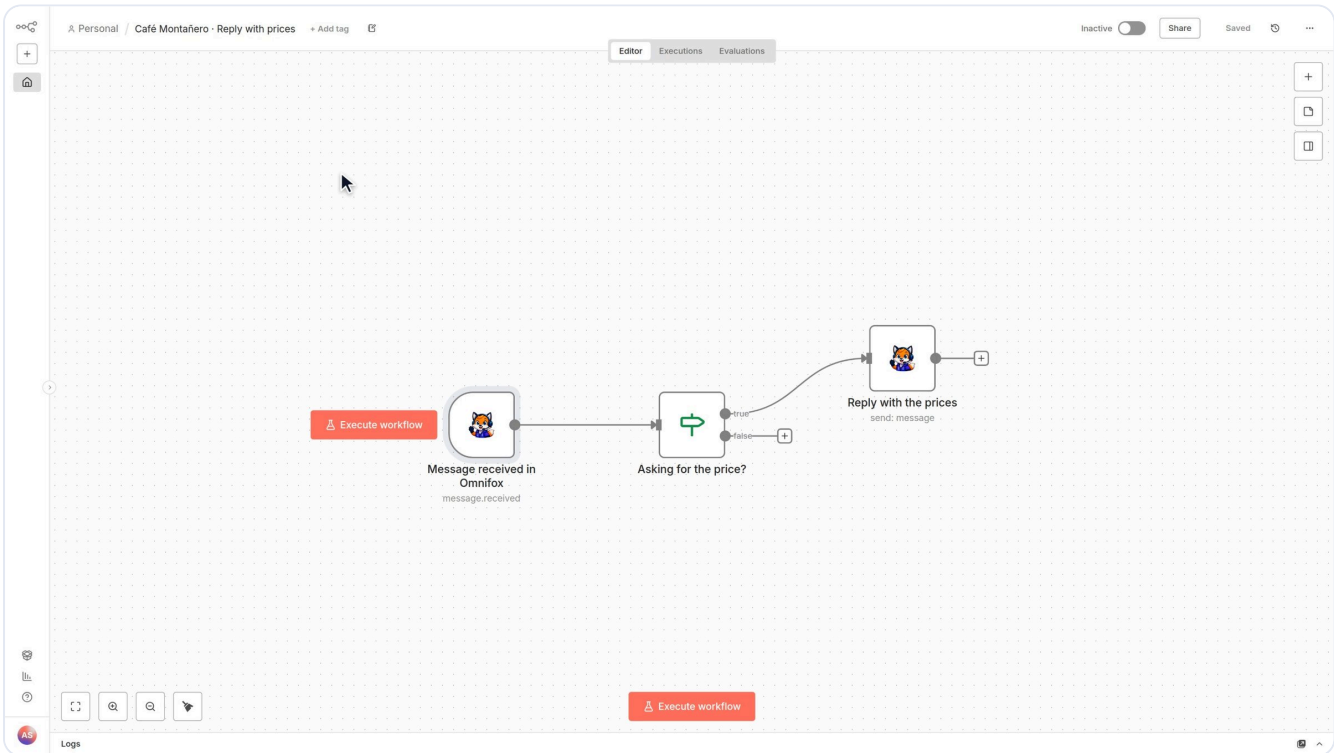


The “Omnifox API” credential in n8n, with the connection tested.

CHAPTER 24

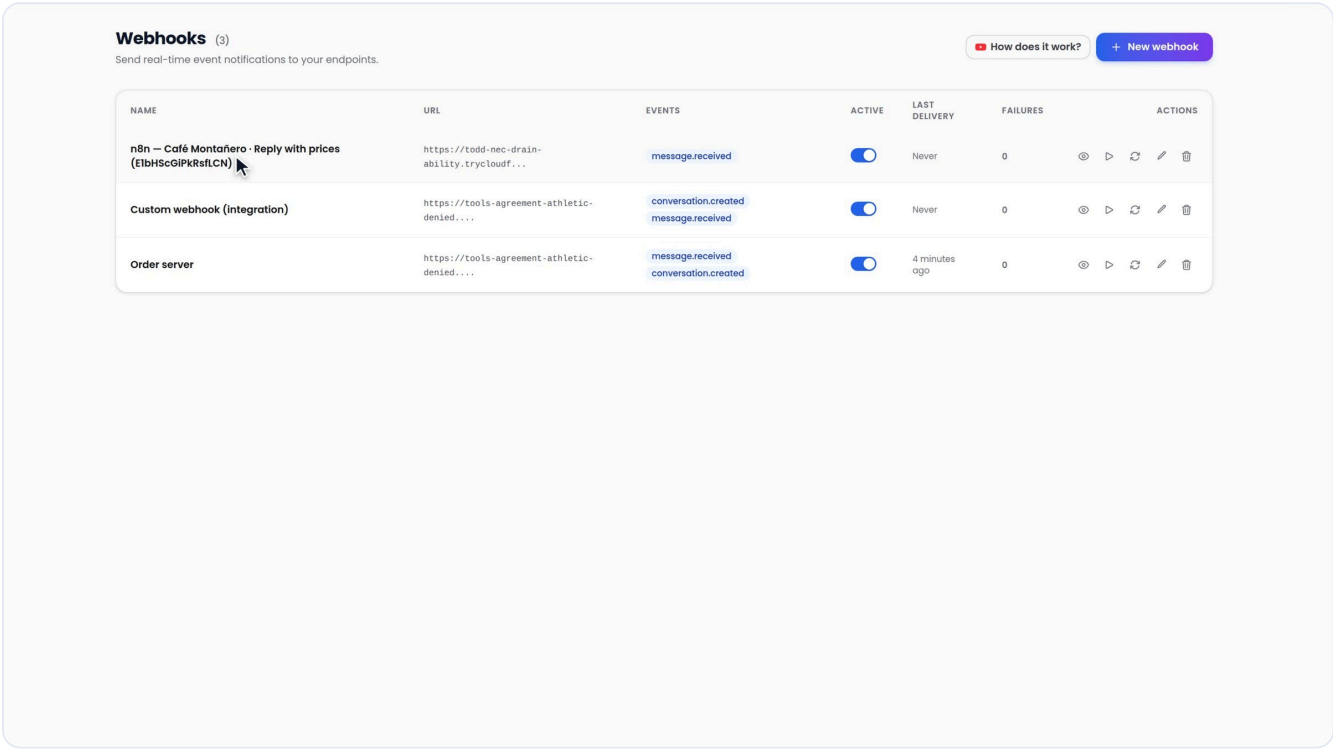
24 n8n: reply only when they ask for the price

A three-node workflow, really tested.

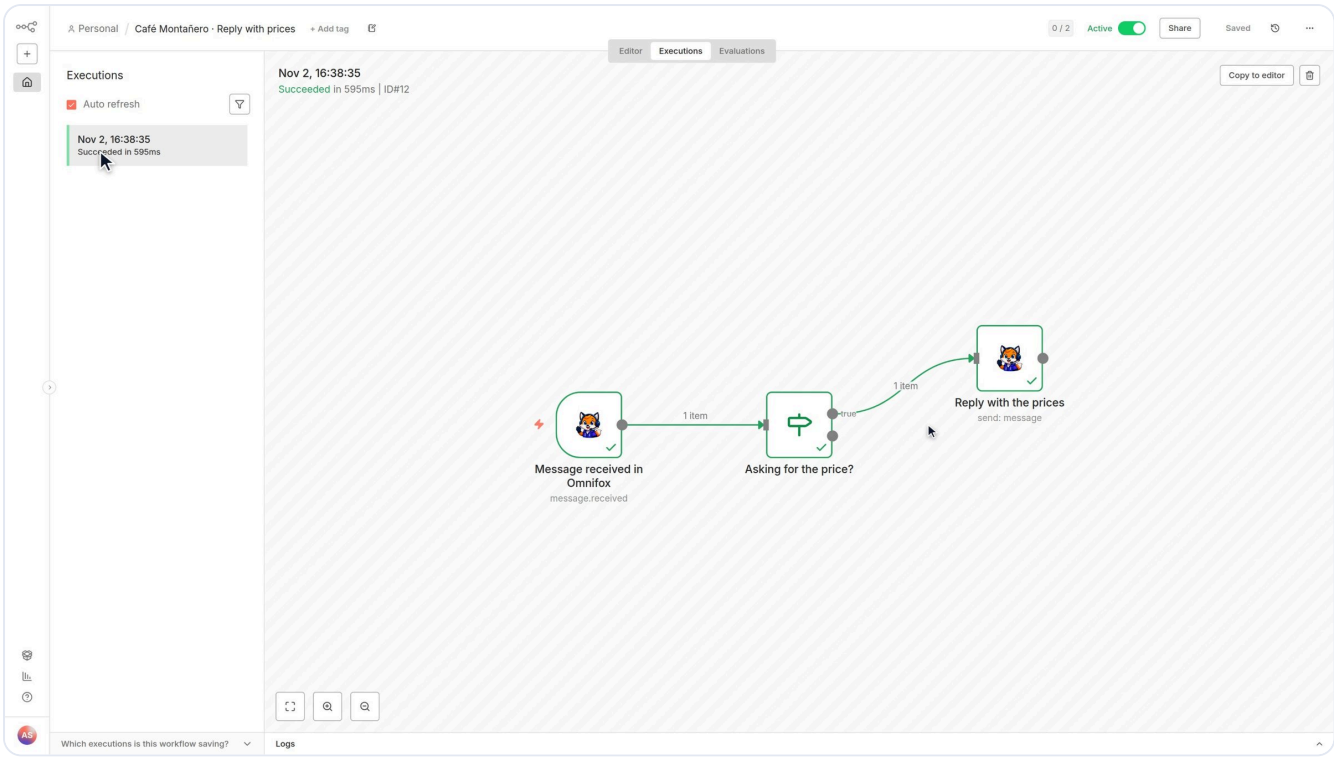


The n8n workflow: Omnifox trigger, condition and reply.

Node	Settings
Omnifox Trigger	Event `message.received`. When activated it creates the subscription in Omnifox and verifies each notice's signature.
If	If `{{ $json.data.message.content_text }}` contains "price" (case-insensitive).
Omnifox	Resource **Message**, operation **Send**, conversation `{{ ...data.conversation.id }}`, text with the price list.



The subscription n8n created, in Omnifox Webhooks.



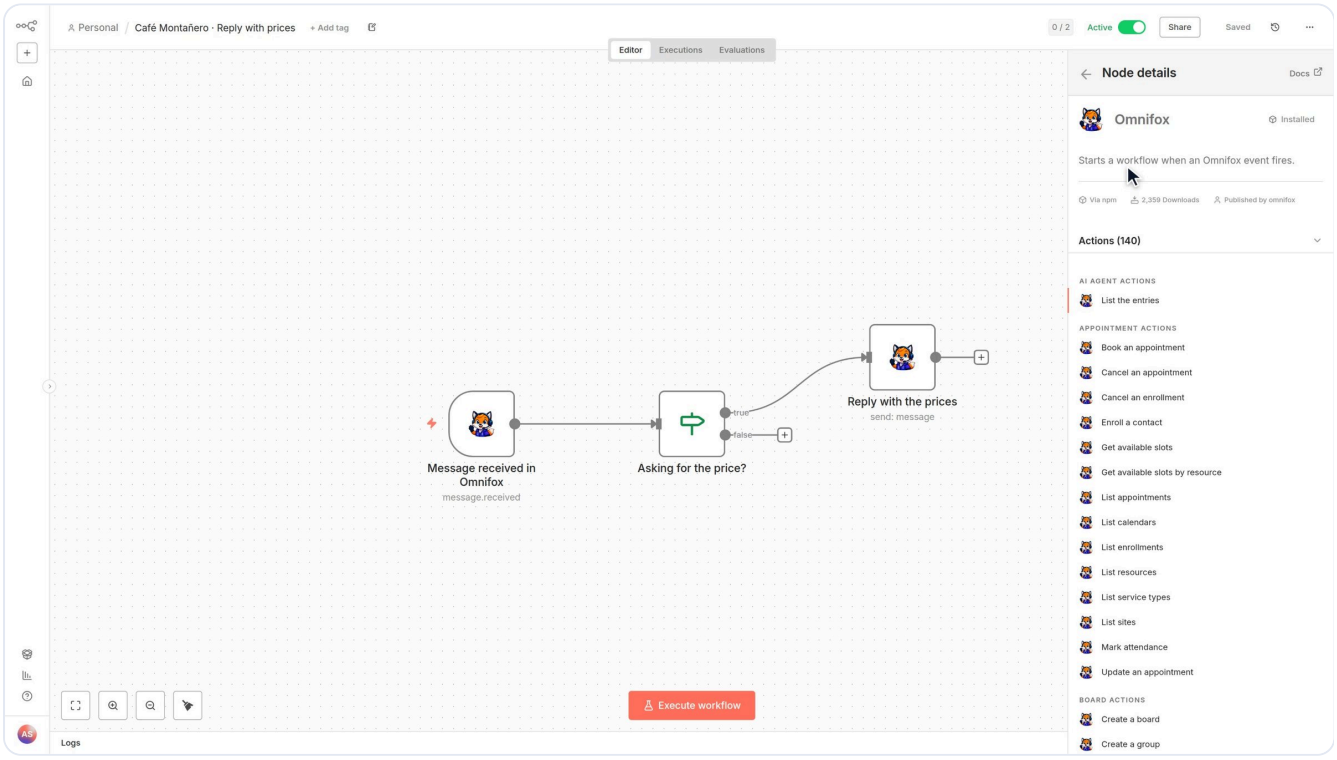
The workflow execution in n8n.

25

CHAPTER 25

n8n: all the operations

140 operations in a single node.



The Omnifox node actions in n8n.

Conversations, contacts, messages, tags, campaigns, channels, teams, AI agents, CRM, boards, calendar, catalog and orders. Lists (channels, tags, teams, templates) load from your account.

Typical error	Solution
Empty data or 403	The key belongs to another workspace or is missing a scope.
404 on every call	Base URL with `/api` at the end: remove it.

26

CHAPTER 26

MCP: let an AI operate Omnifox

72 tools for your AI assistant.

The Omnifox MCP server (`@omnifox/mcp-server`) exposes 72 tools: contacts, conversations, messages, tags, fields, calendar and appointments, courses, catalog, orders, workflows, broadcasts, webhooks and reports. It runs with `npx` and needs two variables:

```
OMNIFOX_API_KEY=your-key
OMNIFOX_BASE_URL=https://app.omnifox.io/api/v1
```

CAREFUL

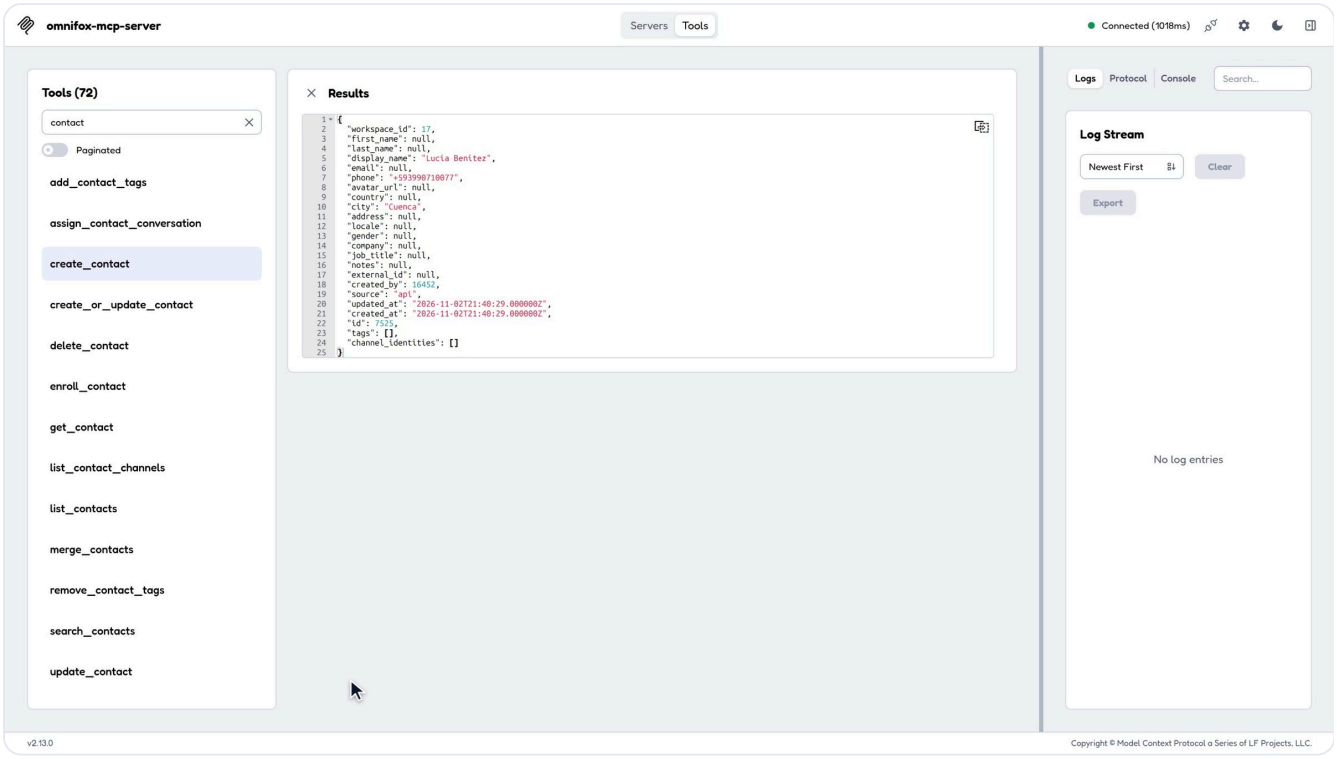
The AI can only do what the key's scopes allow. For an assistant, start with read scopes.

27

CHAPTER 27

Test the MCP server with the Inspector

Try the tools without connecting an assistant.



MCP Inspector: the Omnifox server tools and the result of a call.

```
{
  "mcpServers": {
    "omnifox": {
      "command": "npx",
      "args": ["-y", "@omnifox/mcp-server"],
      "env": { "OMNIFOX_API_KEY": "your-omnifox-api-key", "OMNIFOX_BASE_URL":
"https://app.omnifox.io/api/v1" }
    }
  }
}
```

That block goes into the MCP configuration of Claude Desktop, Cursor or another client. For remote servers, MCP also works over HTTP.

28

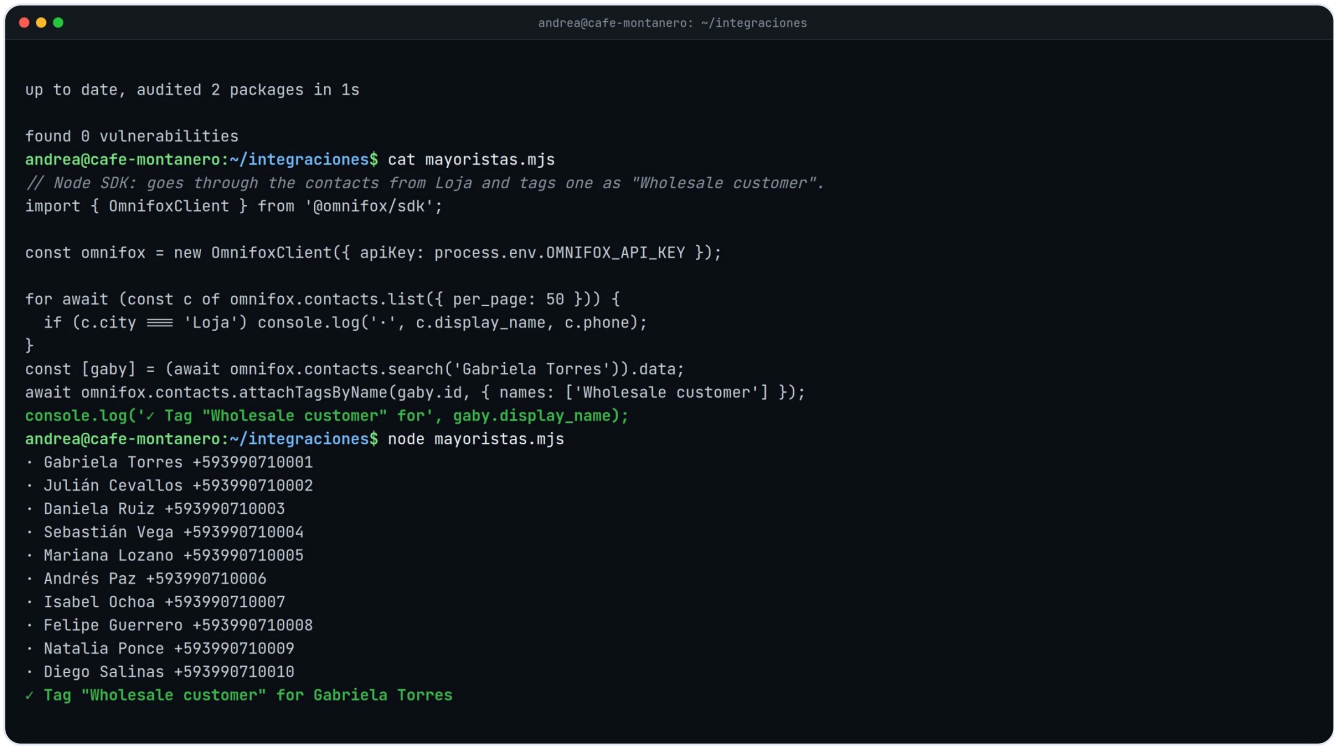
CHAPTER 28

Official SDKs: Node, Python and PHP

Authentication, pagination, retries and errors, solved.

Language	Install	Client
Node	`npm install @omnifox/sdk`	`new OmnifoxClient({ apiKey })`
Python	`pip install omnifox`	`Omnifox(api_key=...)` and `AsyncOmnifox`
PHP	`composer require omnifox/sdk`	`new \\Omnifox\\OmnifoxClient(['apiKey' => ...])`

```
import { OmnifoxClient } from '@omnifox/sdk';
const omnifox = new OmnifoxClient({ apiKey: process.env.OMNIFOX_API_KEY });
for await (const c of omnifox.contacts.list({ per_page: 50 })) {
  if (c.city === 'Loja') console.log('·', c.display_name, c.phone);
}
const [gaby] = (await omnifox.contacts.search('Gabriela Torres')).data;
await omnifox.contacts.attachTagsByName(gaby.id, { names: ['Wholesale customer'] });
```



The Node SDK in action.

```
andrea@cafe-montanero: ~/integraciones

from zoneinfo import ZoneInfo
from omnifox import Omnifox

omnifox = Omnifox(api_key=os.environ["OMNIFOX_API_KEY"])
for appt in omnifox.appointments.list(status="scheduled"):
    time = datetime.fromisoformat(appt.starts_at).astimezone(ZoneInfo("America/Guayaquil"))
    print("·", time.strftime("%m/%d %H:%M"), appt.title, "·", appt.status)
andrea@cafe-montanero:~/integraciones$ python3 citas.py
· 11/05 10:00 Cata de cafés de especialidad · scheduled
andrea@cafe-montanero:~/integraciones$ composer require omnifox/sdk
./composer.json has been updated
Running composer update omnifox/sdk
Loading composer repositories with package information
Updating dependencies
Nothing to modify in lock file
Writing lock file
Installing dependencies from lock file (including require-dev)
Nothing to install, update or remove
Generating autoload files
5 packages you are using are looking for funding.
Use the `composer fund` command to find out more!
No security vulnerability advisories found.
Using version ^0.4.1 for omnifox/sdk
andrea@cafe-montanero:~/integraciones$ php etiquetas.php
· Wholesale customer
· Envío a Quito
· Cafetería
· Cliente mayorista
```

The Python and PHP SDKs.

CHAPTER 29

29 Security best practices

Least privilege, secrets out of the code and always verify.

- One key per integration, with minimum scopes and of bot type.
- Keep it in environment variables or a secrets manager; never in code, in the repository or in the browser.
- Give it an expiry if it’s temporary and regenerate it if someone who knew it leaves.
- Webhooks: verify the signature, reply 200 quickly and process in a queue; use `X-Omnifox-Delivery-Id` to avoid duplicates.

30

CHAPTER 30

Troubleshooting

What usually fails and how to fix it.

Symptom	Likely cause	Solution
401 Unauthenticated	Key copied wrong, expired or regenerated	Copy it again or create a key
403 with `required_scope`	Missing scope	Edit the key and add it
Plan 403	Your plan doesn't include the API or the module	Crece or higher; CRM and Projects depending on plan
Empty lists	Key from another workspace	Create the key in the right workspace
The webhook doesn't arrive	URL not public/https, server down or circuit open	Check the delivery history and the HTTP code; close the circuit
Invalid signature	Re-serialized body or old secret	Use the raw body and the current secret

Need help? Write to us from the support chat in the app or at **omnifox.io/help**.