INSTALL GUIDE · DEVELOPERS AND MSPS

# Alert webhooks

A webhook sends every alert to your own software as JSON. This guide covers what Uplivra sends, how to check that a message really came from your server, and what happens when your end is down.

Uplivra Technologies LLC · Guide for Uplivra 26.10 · Updated 28 September 2026 · Latest version: https://uplivra.com/guides/alert-webhooks.html

## Set one up

Open **Settings › Where alerts go**, choose **Webhook (for other software)**, and paste the address that should receive alerts. Fill in a **signing secret**: a long random string that only your server and the receiver know. Then click **Send test**. The result shows what your end answered and how long it took.

Uplivra won't send webhooks to itself (localhost) or to link-local addresses such as cloud metadata. Addresses on your own network are fine.

## What's sent

Each message is a `POST` with a JSON body:

```
{
  "event": "opened",
  "severity": "down",
  "subject": "[Uplivra] DOWN: fw-01 is DOWN",
  "body": "fw-01 (10.0.0.1) stopped answering ping at 14:02.",
  "alert": { "id": 4211, "title": "fw-01 is DOWN", "opened_at": "2026-09-27T14:02:11Z", … },
  "sent_at": "2026-09-27T14:02:14Z",
  "delivery_id": "dlv_98231",
  "attempt": 1
}
```

`event` is `opened`, `escalated`, `resolved`, `unacked` (a reminder that nobody has acknowledged it) or `test`.

The headers are:

| Header | Contents |
|---|---|
| `X-Uplivra-Event` | The same as `event`. |
| `X-Uplivra-Delivery` | The message's ID. It stays the same on every retry, so if you've already handled an ID, answer 200 and do nothing. |
| `X-Uplivra-Timestamp` | When this try was sent, in Unix seconds. |
| `X-Uplivra-Signature-V2` | `t=<timestamp>,v1=<hex HMAC-SHA256>` over the timestamp, a full stop, then the body. Only sent when there's a signing secret. |
| `X-Uplivra-Signature` | `sha256=<hex HMAC-SHA256>` over the body only. It's kept for receivers built before V2. |

When someone presses **Send again**, the copy gets a new `delivery_id`, and `resent_from` holds the original's.

### Intelligence context and notices

New and escalated alerts also carry `context`: Uplivra Intelligence's likely cause, and with Intelligence the related events, log lines and next steps:

```
"context": {
  "likely_cause": "3 of the 3 devices depend on core-sw1, which is affected too.",
  "related": ["core-sw1 is DOWN at 09:40"],
  "next_steps": ["Start with core-sw1: get it back and this will probably clear by itself."],
  "full": true
}
```

Intelligence notification templates sent to a webhook use `"event": "insight"`, have no `alert`, and carry `template` (its name, such as `rogue_dhcp`) and `notices`: each with `kind`, `level`, `device`, `instance` (port, disk or source address), `title`, `detail`, `evidence` and `suggestion`. They're signed the same way.

## Check the signature

Use `X-Uplivra-Signature-V2`. It covers the time as well as the body, so someone who captures a message can't replay it later.

1. Split the header into `t` and `v1`.
2. Refuse the message if `t` is more than 5 minutes from your clock.
3. Work out HMAC-SHA256 with your signing secret over `t`, then `.`, then the raw body exactly as received. Don't parse and re-encode the body first.
4. Compare the result with `v1` using a constant-time comparison.

Python:

```
import hmac, hashlib, time

def verify(secret: bytes, header: str, body: bytes, tolerance=300) -> bool:
    parts = dict(p.strip().split("=", 1) for p in header.split(",") if "=" in p)
    t, sig = parts.get("t", ""), parts.get("v1", "")
    if not t.isdigit() or abs(time.time() - int(t)) > tolerance:
        return False
    want = hmac.new(secret, t.encode() + b"." + body, hashlib.sha256).hexdigest()
    return hmac.compare_digest(want, sig)
```

Node.js:

```
const crypto = require("crypto");

function verify(secret, header, rawBody, tolerance = 300) {
  const parts = Object.fromEntries(header.split(",").map(p => p.trim().split("=", 2)));
  const t = Number(parts.t);
  if (!Number.isInteger(t) || Math.abs(Date.now() / 1000 - t) > tolerance || !parts.v1)
return false;
  const want = crypto.createHmac("sha256",
secret).update(`${parts.t}.`).update(rawBody).digest("hex");
  return want.length === parts.v1.length && crypto.timingSafeEqual(Buffer.from(want),
Buffer.from(parts.v1));
}
```

## Retries and delivery history

Any answer in the 200s counts as delivered. Anything else, or no answer within 45 seconds, is retried: after 30 seconds, then 1, 2, 4 minutes and so on, up to an hour apart, for 8 tries in all. After that the message is marked **gave up**.

**Settings › Where alerts go › Delivery history** lists every message, newest first. Each one shows every try, with the HTTP status, the time it took, the error, and the start of your end's answer. Filter by channel or by result.

- **Send again** queues a new copy of one message.
- **Send all failed again** covers every message that gave up in the last 7 days and hasn't been sent again already. Use it after fixing a changed address or an outage at your end.

Both are recorded in the audit log.