zapier.georgelambert.org · Markdown indexes
VERAE TIME × ZAPIER

MESSAGE FLOWS

site/packages/docs-master/MESSAGE-FLOWS.md

Message flows

1. Create timestamp (async)

1. Zapier → `POST https://zappier-edge/v1/timestamp` (`x-api-key`)
2. zappier-edge meters → `POST middleware/zapier/v1/timestamp`
3. splitter: chain sha256; optional `verae.archive.put`
4. middleware → chain create or existing lookup
5. middleware publishes `verae.zapier.jobs.watch` `{ jobId, tenantId, … }`
6. HTTP 202 `{ jobId }` back to Zapier
7. job-poller consumes watch, GET chain status
8. On terminal: `verae.zapier.jobs.events`
9. webhook-deliver POSTs Zapier REST Hook if subscribed

2. Wait

Same as (1) but HTTP holds until `jobs.events` or `WAIT_TIMEOUT_MS` → `{ status: pending, jobId }`.

3. Hash already registered

Step 4 returns original jobId + original seal. No second chain write. `receipts[0].kind = seal`. Later archive attaches become `receipts[1…]` if `includeAttached`.

4. includeAttached

After seal is known, aggregator publishes `verae.archive.query`. Each WORM: bloom miss = no packet; hit = `verae.archive.reply.<correlationId>`. Aggregator merges into wait JSON.

5. Multipart attachments

Splitter emits one `archive.put` per file (`kind: file` , `contentSha256`). Chain never stores bytes.

6. Batch Merkle (bulk summary)

1. Zapier → `POST /v1/timestamp/batch` (one item per line)
2. Middleware hashes each item (leaves), builds Merkle tree
3. Chain seals **only the root**
4. Each leaf proof → `verae.archive.put` `kind: tree` on a sharded tree node
5. Central `GET /hashes/{leaf}` → miss (`itemizedOnMainChain: false`)
6. `GET /hashes/{leaf}?includeTree=true` → aggregator broadcasts `verae.archive.query` `kinds=["tree"]`
7. Tree node bloom hit → `archive.reply.<id>` with proof; others silent
8. Response: root seal + `tree-leaf` receipt

7. Simulator

`packages/verae-zapier-simulator` replays flows 1–6 in-process with a trace console, fault injection, and modification suggestions. It does not connect to live NATS.