

zapier.georgelambert.org · Markdown indexes

VERAE TIME x ZAPIER

03 zapier verae path

site/packages/dev-overview/03-zapier-verae-path.md

Zapier → Verae API → Zapier

How a timestamp job is submitted, watched, and returned. Lab today: `MOCK_VERAE=true`. Live path is the same HTTP contract with `MOCK_VERAE=false`.

Rule



Zapier never has `NATS_URL`. Verae never has `NATS_URL`.

1. Submit (async create)

1. Zap action **Create Timestamp** (or curl) `POST https://zap.zapier.georgelambert.org/zapier/v1/timestamp` with portal API key `zmw_...` (not a Verae JWT).
2. access-zapier / edge: authz, then `meter()` records cents; if `NATS_URL` is set, `natsPublish(verae.billing.usage.recorded)`.
3. Forwards to middleware `POST /zapier/v1/timestamp`.

4. Middleware `checkEntitlement` (local plan limits today).
5. `veraeClient.createTimestamp`:
 - **Mock**: in-process job, fake `jobId`.
 - **Live**: `POST https://api.veraetime.net/api/timestamp` with JWT → **202** `{ jobId }`.
6. Splitter (in-process): hash vs attachments; optional `verae.archive.put` (bytes never go to chain).
7. `enqueueWatch` → JetStream `verae.zapier.jobs.watch`.
8. HTTP **202** `{ jobId, sha256, existing }` back to Zapier immediately.

Same SHA-256 is idempotent (`existing: true`).

2. Monitor (job poller)

The poller is **in the middleware Node process** when `NATS_ENABLED=true` (`startJobPollerWorker`). Fleet does **not** spawn a second copy (`job-poller.enabled: false`).

1. Durable consumer `job-poller` on `ZAPIER_JOBS` / `verae.zapier.jobs.watch`.
2. Resolve `tokenRef` → tenant Verae password → `POST` login → JWT (stays in middleware).
3. `GET /api/status/{jobId}` (mock or live timestamp server).
4. `pending` → `nak` after `intervalMs` (retry).
5. `completed` → publish `jobs.events { event: timestamp.completed, status, jobId, tenantId, traceId }`.
6. `failed` → `timestamp.failed`. Too many attempts → `timestamp.timeout`.
7. `ack` the watch message.

Live OpenAPI: poll until `completed` / `failed`. Hash lookup is **501** on live (`lookupHash`). Map `certificate`, `timestamp`, `blockIndex`, `sha256` into the receipt Zap samples expect.

3. Wait (sync Zap step)

Create Timestamp and Wait → `POST /zapier/v1/timestamp/wait`.

1. Same as submit (watch is already on NATS).
2. `waitForJobEvent(jobId, WAIT_TIMEOUT_MS)` subscribes to `verae.zapier.jobs.events`.
3. On matching `jobId` event: return status JSON (certificate / receipt).
4. On timeout: `{ status: pending, jobId }` so the REST Hook can finish the Zap.

`includeAttached`: after seal, aggregator should `verae.archive.query`; bloom hits merge extra receipts. **Simulator does this; live aggregator hook is TBD.**

4. Push back to Zapier

- **Wait response:** HTTP body on the same Zap step.
- **REST Hook:** Zapier subscribes `POST /zapier/v1/hooks`. On `jobs.events`, middleware enqueues `verae.zapier.webhooks.deliver`. Worker POSTs `targetUrl` (`hooks.zapier.com`).
- **Poll status:** Zap action **Find Job Status** → `GET /zapier/v1/status/{jobId}` → middleware `GET` Verae (or last event).

5. Verify

`POST /zapier/v1/verify` → middleware → Verae `POST /api/verify`. No job watch.

6. Batch Merkle (designed; same bus)

From `MESSAGE-FLOWS.md` :

1. `POST /v1/timestamp/batch` — one item per line.
2. Middleware hashes leaves, builds Merkle tree.
3. Chain seals **only the root** (`POST /api/timestamp` on the root).
4. Each leaf proof → `verae.archive.put kind: tree`.
5. Central hash lookup of a leaf misses unless `includeTree=true`.
6. Aggregator broadcasts `verae.archive.query kinds=["tree"]`.
7. Tree node bloom hit → `archive.reply.<id>` with proof.

Live wait path does not yet call the aggregator, so tree-leaf receipts are not merged on NS1.

Connect live `api.veraetime.net`

Still blocked on operator bind credentials. When ready:

```
# middleware env
MOCK_VERAЕ=false
VERAE_API_BASE_URL=https://api.veraetime.net
NATS_ENABLED=true
NATS_URL=nats://10.10.10.21:4222,nats://10.10.10.22:4222,nats://10.10.10.23:4222
JETSTREAM_REPLICAS=3
```

Then: working Verae login, one live seal, confirm 202/status/verify, map receipt fields, REST Hook to a private Zap. See `zapier-decisions/STATUS.md` “Once MOCK_VERAE=false”.

Billing on this path

Hop	Designed	Running
Edge meter	<code>verae.billing.usage.recorded</code>	coded ; fires when edge has <code>NATS_URL</code>
Account-balance	subscribe + debit prepaid; 402 if empty	subscriber running ; 402 from books not proven on every Zap
Middleware quota	local files via <code>recordUsage</code>	running ; not NATS
<code>verae.zapier.usage</code>	optional JS meter	stream only

Connect a new internal process

```
export NATS_URL=nats://10.10.10.21:4222,nats://10.10.10.22:4222,nats://10.10.10.23:4222
# clone verae-nats-process, rename subjects, register in verae-fleet
```

Jobs stay: HTTP in from Zapier → HTTPS to Verae → NATS watch/events → HTTP back to Zapier.