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

Middleware overview

site/docs/02-architecture/overview.md

Architecture Overview

Purpose

Connect Zapier automations to Verae blockchain timestamping without:

- exposing raw Verae JWTs to end users,
- requiring Zapier to poll async jobs,
- coupling billing/plans to the core timestamping API,
- running multi-instance middleware with in-memory only job queues.

Components

1. `verae-zapier` (Zapier Platform CLI app)

Runs on: Zapier's cloud when a Zap step executes.

Responsibilities:

- Custom auth field `api_key` (middleware-issued `zmw_…` keys).
- Map Zapier actions/searches/triggers to middleware HTTPS routes.
- Attach `Authorization: Bearer <api_key>` on every request.
- Translate middleware `402` / `403` into user-visible Zapier errors.

Does not:

- Call `api.veraetime.net` directly.
- Speak NATS.
- Enforce plan quotas (middleware does).

2. `verae-zapier-middleware` HTTP edge

Runs on: Your infrastructure (public HTTPS).

Responsibilities:

- Tenant identity and API keys.
- Auth bridge: API key → Verae login → JWT (server-side).
- Entitlement checks and usage metering.
- Synchronous API surface under `/zapier/v1/*`.
- REST Hook subscribe/unsubscribe storage.
- Publish async work to NATS when `NATS_ENABLED=true`.

3. NATS + JetStream

Runs on: Private network with middleware.

Responsibilities:

- Durable work queue for job status polling.
- Event stream for terminal job states.
- Work queue for Zapier webhook HTTP delivery with retries.

4. Workers

Runs on: Same deploy as middleware or separate worker processes.

Worker	Consumes	Calls
Job poller	`verae.zapier.jobs.watch`	`GET /api/status/{jobId}` on Verae
Event router	`verae.zapier.jobs.events`	Enqueues webhook deliveries
Webhook deliver	`verae.zapier.webhooks.deliver`	`POST` Zapier `targetUrl`

5. `api.veraetime.net` (Verae Timestamping Service)

Source of truth for login, timestamp jobs, status, and verification.

OpenAPI: production Swagger / `Verae-Swagger.yaml`.

Request flows

A. Create Timestamp (async)

```
Zapier → POST /zapier/v1/timestamp
Middleware: authenticate, checkEntitlement, POST /api/timestamp
Middleware: publish jobs.watch → return 202 { jobId }
Worker: poll status until terminal → publish jobs.events
Event router: match webhooks → publish webhooks.deliver
Webhook worker: POST hooks.zapier.com/...
```

B. Create Timestamp and Wait

```
Zapier → POST /zapier/v1/timestamp/wait
Middleware: create + wait for jobs.events (or in-process wait if NATS off)
→ return StatusResponse (completed/failed) or pending+jobId on timeout
```

C. Auth connection test

```
Zapier → GET /zapier/v1/auth/me  Authorization: Bearer zmw_…
Middleware: resolve API key → tenant → optional validate Verae token
→ { tenantId, plan, usage, ... }
```

Feature flags

Flag	Effect
`NATS_ENABLED=false`	In-process job poller; still full HTTP API (Phase 6 path)
`NATS_ENABLED=true`	JetStream workers; no in-process poller
`MOCK_VERAE=true`	No live Verae; deterministic mock jobs for tests
`DEBUG_VERAE=…`	Runtime failure tracing (see debugging.md)

Security boundaries

```
Public Internet
  ├─ Zapier → Middleware HTTPS only
  └─ Middleware → Zapier webhook HTTPS only

Private
  ├─ Middleware ↔ NATS
  └─ Middleware/Workers → api.veraetime.net HTTPS
```

Never expose NATS ports to the public internet.

Scaling model

- **HTTP edge:** scale replicas behind a load balancer (stateless except shared store).
- **NATS consumers:** queue groups — adding workers increases poll/deliver throughput.
- **Store:** file JSON is MVP single-node; Postgres/Redis required for multi-node (Phase 15).

Related documents

- nats-subjects.md — subjects, streams, payload schemas
- ../plans/phase-gates.md — test gates
- ../../TODO.md — full implementation order