

zapier.georgelambert.org · Markdown indexes

VERAE TIME × ZAPIER

02 endpoint io

site/packages/dev-overview/02-endpoint-io.md

Endpoint I/O

Each product NATS address: **what it is for, who sends, who receives, how to attach.**

Connect with a **long-lived** client (reconnect-per-message is the anti-pattern):

```
NATS_URL=nats://10.10.10.21:4222,nats://10.10.10.22:4222,nats://10.10.10.23:4222
```

Public edge (token in `/etc/nats/public-edge.token` on NS1): `nats://<token>@70.88.205.138:4222`. Do not put that URL in Zapier or in Verae.

verae.zapier.jobs.watch

For	Work queue: “poll Verae until this <code>jobId</code> is terminal.”
In from	middleware <code>enqueueWatch</code> after <code>POST /api/timestamp</code> (or mock) returns <code>jobId</code>
Out to	job-poller worker (queue <code>job-poller</code> , durable on <code>ZAPIER_JOBS</code>)
Then	poller calls Verae <code>GET /api/status/{jobId}</code> over HTTPS , not NATS

Payload: `tenantId`, `veraeUserId`, `jobId`, `tokenRef`, `enqueuedAt`, `attempt`, `maxAttempts`, `intervalMs`, `traceId`. JWT stays in middleware (`tokenRef`); never on Zapier.

Connect Verae-side: do **not** subscribe this from `api.veraetime.net`. The poller is the Verae HTTP client. To add another watcher, start another `job-poller` consumer in the same queue group.

verae.zapier.jobs.events

For	Terminal outcomes: <code>timestamp.completed</code> / <code>failed</code> / <code>timeout</code>
In from	job-poller after Verae status is not <code>pending</code>
Out to	(1) middleware <code>waitForJobEvent</code> (wait HTTP); (2) webhook router → <code>webhooks.deliver</code> ; (3) optional <code>verae-jobs-events</code> mailbox <code>:3030</code>

If `JOBS_EVENTS_EXCLUSIVE=1`, only `verae-jobs-events` is the durable consumer and republishes `verae.internal.jobs.events`. Default lab: middleware wait + webhook worker share the stream.

Payload: `event`, `tenantId`, `jobId`, `status`, `traceId`, `emittedAt`.

verae.zapier.webhooks.deliver

For	HTTPS POST to <code>hooks.zapier.com</code> (REST Hook)
In from	webhook worker on completed/failed events
Out to	Zapier cloud HTTPS , never NATS

Payload: `hookId`, `tenantId`, `targetUrl`, `event`, `payload`, `attempt`, `traceId`. Slow consumer: keep replica floors on the worker if you split it out of middleware.

verae.zapier.usage

For	Optional meter of Zap actions on NATS (JetStream)
In from	designed: middleware / edge
Out to	designed: <code>USAGE_WRITER</code> → books
Running	stream exists; no writer; middleware uses local usage files

Prepaid meter that **is** wired uses `verae.billing.usage.recorded` (core pub), not this stream. Finish: either publish here from `recordUsage` or drop the stream.

Billing (`verae.billing.*`) — account-balance source of truth

Request-reply unless noted. Queue `account-balance` . **Not** JetStream.

Address	For	In from	Out to
<code>statement.get</code>	prepaid, credits, usage, payments	portal, CS, sales, accounting, access-web/api	reply to requester
<code>balance.get</code>	cents only	internal	reply
<code>balance.adjust</code>	CS credit or portal reload	CS, access-web (reload kind only)	books + reply
<code>usage.recorded</code>	meter pub	zappier-edge <code>meter()</code> <code>onUsage</code> when <code>NATS_URL</code> set	account-balance subscriber (debits prepaid)
<code>payment.recorded</code>	reload/ invoice pub	portal	books
<code>credit.applied</code>	goodwill pub	CS/admin	books
<code>customer.put</code>	upsert customer	edge ledger	books

Authz **must** pass `verae.access.authz.check` before these from a public plane. Leaf plane **cannot** call billing (policy).

Middleware `recordUsage` still writes **local tenant files** and does **not** publish `usage.recorded` . Two meters until that is unified.

Access planes

Address	For	In	Out
<code>verae.access.authz.check</code>	allow/deny <code>{ plane, subject, principal }</code>	every plane + edge <code>billing-nats.ts</code>	<code>{ allow, reason }</code> ; deny also pubs <code>authz.deny</code>
<code>verae.access.authz.deny</code>	audit	authz on deny	nowhere yet (no SIEM)
<code>verae.access.web.in /</code> <code>verae.access.web.billing.*</code>	browser portal	access-web HTTP	<code>statement.get /</code> <code>balance.adjust</code> kind=reload after authz
<code>verae.access.api.*</code>	customer <code>x-api-key</code>	access-api	statement; HTTPS to middleware for jobs
<code>verae.access.zapier.*</code>	Zapier HTTPS only	access-zapier	jobs.watch via HTTP to middleware , not Zapier NATS; usage.recorded allowed by policy
<code>verae.access.leaf.in</code>	S2S <code>{ token, target, payload }</code>	remote leaf	archive + jobs after authz; never billing
<code>verae.access.staff.*</code>	CS/sales/ accounting	access-staff	billing after authz

Ingress mapping: `accessAddress(plane, 'verae.billing.statement.get')` →
`verae.access.<plane>.billing.statement.get`.

Identity

Address	For	In	Out
<code>verae.identity.bind</code>	bind email/ customerId → <code>veraeUserId</code>	portal / middleware	reply
<code>verae.identity.lookup</code>	lookup by customerId or <code>veraeUserId</code>	same	reply

Queue `identity`. Fleet keepFloor.

Archive / tree

Address	For	In	Out
<code>verae.archive.put</code>	WORM/tree write (<code>kind</code> : publicMeta / privateMeta / file / tree)	splitter (in- process from middleware today)	every worm/tree (JS)
<code>verae.archive.query</code>	broadcast lookup	aggregator	all worm/tree, no queue group
<code>verae.archive.reply.<correlationId></code>	bloom hit only	worm/tree	aggregator

Silence = miss. Chain seal does not wait on archive. Bytes never go to the chain.

Template / unused

Address	For	In	Out	Running
<code>verae.splitter.in</code>	future split-out splitter process	middleware	splitter	no
<code>verae.example.process.in</code>	template worker	any	example- process Q	no
<code>verae.example.process.out</code>	template fan-out	template	peers	no
<code>verae.example.process.reply.<id></code>	template reply	template	requester	no
<code>verae.internal.jobs.events</code>	exclusive mailbox republish	jobs-events	wait/ webhook	no unless env

How to connect Verae API to NATS

`api.veraetime.net` is **HTTPS**. It does not subscribe. The glue is middleware:

1. Set `VERAE_API_BASE_URL=https://api.veraetime.net` and `MOCK_VERAE=false`.
2. Middleware login (username/password) → JWT stored as `tokenRef`.
3. `POST /api/timestamp` → 202 { `jobId` }.
4. Publish `verae.zapier.jobs.watch`.
5. Poller `GET /api/status/{jobId}` until `completed` / `failed`.
6. Publish `verae.zapier.jobs.events`.
7. Wait HTTP and/or REST Hook.

Do **not** point Verae at `NATS_URL`. Do **not** put NATS in Zapier env. A new worker that must see seals uses `jobs.events` or HTTP `GET /zapier/v1/status/{jobId}`.

Template for a new addressed process: clone `verae-nats-process`, rename `verae.example.process.*`, add the subject to `verae-access-authz` policy for the plane that may call it.