zapier.georgelambert.org · Markdown indexes
VERAE TIME × ZAPIER
Operations & usage
site/packages/zappier/docs/USER-MANUAL.md

Zappier — Operations & Usage Manual

Version: 0.2.0 · **Last updated:** 2026-07-28

Zappier is a metered API platform: every API call your customers make is priced per endpoint, adjusted by their customer type, tracked in a usage ledger, and billed through Stripe once a day — or invoiced manually by purchase order. Customers self-serve through the portal at `/portal`. This manual covers running and operating the system. For internals, see DEVELOPER.md; for accounting procedures see ACCOUNTING.md; for the end-user view see CUSTOMER-PORTAL.md.

Table of contents

1. Quick start

```
cd /Users/marchon/zappier
npm install
npm run dev          # starts the API on http://localhost:3000
```

Environment variables (all optional except `STRIPE_SECRET_KEY` for billing):

Variable	Default	Purpose
`PORT`	`3000`	HTTP port for the API server
`ZAPPIER_DB`	`<install root>/zappier.db`	SQLite database file location
`ADMIN_KEY`	`admin-dev-key`	Key for the admin console and admin API
`ADMIN_USER`	`admin`	Admin console primary username
`DEMO_ADMIN_USER` / `DEMO_ADMIN_PASSWORD`	`demo` / `$$$Adm1n###`	Demo sign-in — override in production
`STRIPE_SECRET_KEY`	— (required for billing)	Billing job + portal reloads, loaded from `.env`

All runtime paths (database default, `.env`, OpenAPI spec, static assets) resolve from the installation root — the compiled server (`node dist/index.js`) runs from any working directory, under systemd, Docker, or cron.

The `.env` file at the repo root holds `STRIPE_SECRET_KEY`. It is gitignored and owner-only (`chmod 600`). A template is in `.env.example`.

On first start the database is created and seeded with:

- **Rate card:** `status` (free), `storage-list` (free), `transform` (fixed 4¢), `storage` (variable: 10¢ base + 1¢/KB metadata + 50¢/MB attachments)
- **Customer types:** Free (×1.0, 100¢/month credit), Pro (×0.5, 1000¢ credit), Business (×0.25, 10000¢ credit)
- **Demo customers:** `key-ada` (Free), `key-grace` (Pro), `key-linus` (Business)

> Seeding only happens into an **empty** database. Existing data is never overwritten on restart.

---

2. The three surfaces

Surface	URL / location	Who it's for
Public API	`http://localhost:3000/v1/*`	Your API customers
Interactive API docs	`http://localhost:3000/docs`	Developers integrating with you
Admin console	`http://localhost:3000/admin`	You (operations & accounting)
Customer portal	`http://localhost:3000/portal`	End-user customers (self-service)
Zapier app	`zapier-app/` directory	No-code users via Zapier

Interactive API docs
Interactive API docs

3. How pricing works

Every priced call returns its **quote** in the response, so customers always know what a call cost:

```
{
  "quote": {
    "endpointId": "storage",
    "listCents": 62,
    "totalCents": 31,
    "breakdown": { "baseCents": 10, "metadataCents": 2, "attachmentCents": 50 }
  }
}
```

The price of a call is computed in three steps:

1. **Endpoint rule** (from the rate card):
   - `free` — always 0¢
   - `fixed` — a flat `fixedCents` per call
   - `variable` — `baseCents` + `perKbCents` × ceil(metadata bytes / 1024)
     - `perMbCents` × ceil(attachment bytes / 1 MB)

2. **Customer-type multiplier** — `totalCents = round(listCents × multiplier)`. A per-customer **multiplier override** (set in the Customers tab) wins over the type multiplier — use it for negotiated enterprise deals.
3. **Monthly credit** — at billing time, each customer's type credit (e.g. Pro = 1000¢) is subtracted from their month-to-date total. Only the excess is billed.

Worked example. A Pro customer (×0.5) uploads a 1 MB file with 2 KB of metadata to `storage`:

- list = 10¢ base + 2¢ metadata + 50¢ attachment = **62¢**
- Pro multiplier: 62 × 0.5 = **31¢** charged to their usage ledger
- If their month-to-date is 1500¢ and the Pro credit is 1000¢, the daily billing job reports **500¢** to Stripe.

---

4. Operating the Pricing Admin UI

Open `http://localhost:3000/admin` and sign in. Two accounts are available:

Username	Password	Purpose
`admin`	the `ADMIN_KEY` env value (default `admin-dev-key`)	Primary operator
`demo`	`$$$Adm1n###` (env `DEMO_ADMIN_PASSWORD`)	Demo / stakeholder access

Sessions are token-based and remembered in browser local storage until you click **Sign out** or the server restarts (tokens are in-memory — just sign in again). The legacy `x-admin-key` header still works for scripts and curl.

4.1 Rate card tab

Rate card
Rate card

One row per API endpoint (matched by OpenAPI `operationId`).

- **Change a price:** edit the kind (`free` / `fixed` / `variable`) and the cent fields, then click **Save**. Takes effect on the next API call — no restart needed.
- **Add an endpoint:** enter the `operationId` (must match `openapi.yaml`), pick a kind, click **Add**.

- **Delete** removes the rule. If an endpoint has no rule and the customer's type has no default rule, calls to it are rejected with 403 — deletion is how you turn an endpoint **off**.

4.2 Customer types tab

Customer types
Customer types

Types are your pricing tiers.

- **Multiplier** scales every price for that type (0.5 = 50% of list).
- **Monthly credit (cents)** is the free included usage per month.
- **Add customer type:** id, name, multiplier. New types start with 0 credit; edit after adding.

4.3 Customers tab

Customers
Customers

- **Create** a customer: name + type. **The API key is shown once** in the status line at the bottom — copy it immediately and send it to the customer.
- **Change type** with the dropdown, then **Save**.
- **Email** — used for portal sign-in/claiming and email invoicing.
- **Billing** — `Stripe` (metered by the daily job) or `Purchase order` (manual invoicing with PO numbers; see ACCOUNTING.md).
- **Multiplier override**: a number here replaces the type multiplier for this customer only. Leave blank to inherit from the type.
- Stripe customer IDs are attached via the admin API (`PUT /admin/api/customers/:id` with `{"stripeCustomerId": "cus_..."}`) — see section 6.

4.4 Invoices, Reports, System tabs

The **Invoices** tab generates monthly invoices from metered usage (per period, optionally per customer, with optional PO number), walks them draft → issued → paid, and opens print-ready invoice pages. The **Reports** tab produces date-ranged billing reports (all customers, one customer, or one billing type) with CSV download, plus daily/weekly usage-trend charts. The **System** tab shows Zapier integration health and the current-period billing snapshot. Full procedures: ACCOUNTING.md.

Invoices
Invoices

4A. Customer portal

Your customers self-serve at `http://localhost:3000/portal`: signup (or claiming an account you created, by email), sign-in with optional TOTP two-factor authentication, month-to-date usage, invoice history with print view, prepaid balance reloads (drawn down automatically at invoice issue), email-invoicing preferences, API-key regeneration, and live pricing. The full end-user guide is CUSTOMER-PORTAL.md.

---

5. Using the public API

All calls need the customer's API key in the `x-api-key` header. Interactive docs with a "Try it out" console are at `/docs`.

```
# Free status check
curl -H 'x-api-key: key-ada' http://localhost:3000/v1/status

# Fixed-price call (4¢ list)
curl -X POST -H 'x-api-key: key-grace' -H 'content-type: application/json' \
  -d '{"text":"hello"}' http://localhost:3000/v1/transform

# Variable-price call: metadata + attachments
curl -X POST -H 'x-api-key: key-grace' \
  -F 'metadata={"title":"Q3 report"}' \
  -F 'attachments=@report.pdf' \
  http://localhost:3000/v1/storage

# List your stored items (free)
curl -H 'x-api-key: key-grace' http://localhost:3000/v1/storage

# Your month-to-date usage, with credit applied
curl -H 'x-api-key: key-grace' http://localhost:3000/v1/usage
```

Limits & validation: requests are validated against `openapi.yaml` (bad requests get 400); attachments are capped at **25 MB per file**; `metadata` must be valid JSON (400 otherwise).

---

6. Billing operations (Stripe)

6.1 How it works

A scheduled job runs the billing reporter **daily at 06:17 America/New_York** (Kimi cron job "Zappier billing · report usage to Stripe"). For each customer with a Stripe ID it:

1. Sums their usage since the 1st of the month, subtracts their type's monthly credit → **billable cents**.
2. Reports only the **delta** above what was already reported this month to Stripe as a meter event (`zappier.api_cents`, value = cents).
3. Records the new cumulative total in the `billing_reports` ledger.

Re-running is always safe: the ledger makes repeats no-ops, a per-run lock prevents overlapping executions, and a deterministic Stripe `identifier` (`customer:period:billable`) dedupes crash retries.

6.2 One-time Stripe setup (test mode)

1. Dashboard (test mode ON) → **Billing** → **Meters** → **Create meter**: event name `zappier.api_cents`, aggregation **Sum** of `value`, customer mapping `stripe_customer_id`.
2. **Product catalog** → **Add product** "Zappier API usage" → price: recurring, monthly, metered against that meter, **$0.01 per unit** (1 unit = 1 cent).
3. For each billable customer: create the Stripe Customer, attach a payment method, and add a **subscription** with the metered price. Meter events for customers without a metered subscription are recorded but never invoiced.
4. Put the `sk_test_...` key into `.env` (replace the placeholder).
5. Attach Stripe IDs to Zappier customers:

```
curl -X PUT -H 'x-admin-key: admin-dev-key' -H 'content-type: application/json' \
  -d '{"stripeCustomerId":"cus_..."}' \
  http://localhost:3000/admin/api/customers/cust_2
```

6.3 Verifying a run

```
npx ts-node src/jobs/report-usage.ts
```

Expected output per customer:

- `skip <id> <period> (nothing to report)` — no billable usage yet
- `skip <id> <period> (already reported Nc)` — no new usage since last run
- `<id>: reported N billable cents to Stripe` — delta sent
- `report-usage: another run holds the lock, abort run` — safe concurrent abort

Then check the meter's **Events** tab in the Stripe dashboard and the test customer's **upcoming invoice**.

6.4 Going live

Repeat 6.2 steps 1–3 in live mode, replace `.env` with the `sk_live_...` key from the **same Stripe account**, and keep the same cron. The live product `prod_Uxv9SAeIOZzyx1` (currently deactivated) can be reactivated or recreated.

---

7. The Zapier integration

The `zapier-app/` directory contains the Zapier Platform app:

- **Authentication:** API key — the user pastes their API base URL (e.g. `http://localhost:3000`) and their `key-...` customer key; the connection is tested against `/v1/status`.
- **Trigger "New Item":** polls `GET /v1/storage` for newly stored items.
- **Action "Store Data":** calls `POST /v1/storage` with metadata and optional file attachments — billed per the rate card.

To publish: create a Zapier developer account, `cd zapier-app && npm install && zapier login && zapier push`, then share the app or submit it to the Zapier marketplace.

---

8. Day-to-day runbook

Task	How
Change a price	Admin console → Rate card → Save
Add a customer	Admin console → Customers → Create → copy the one-time API key
Set a customer's billing type/email	Customers tab → Billing dropdown / Email field → Save
Invoice a period	Invoices tab → Generate → Issue → Mark paid (ACCOUNTING.md)
Export accounting data	Reports tab → filters → Download CSV
Give a customer a deal	Customers tab → multiplier override, or a new customer type
Turn an endpoint off	Rate card → Delete (calls get 403)
Check a customer's usage	Their portal dashboard, `GET /v1/usage` with their key, or query `zappier.db`
Check billing ran	Kimi notification after each 06:17 run; or run the job manually
Backup	Copy `zappier.db` (SQLite, single file)
Update dependencies	`npm outdated`, then `npm test` must stay green (165 tests)

9. Troubleshooting

Symptom	Likely cause	Fix
`401 invalid or missing API key`	Wrong/absent `x-api-key`	Re-issue key from Customers tab
`403 No price rule for ...`	Endpoint deleted from rate card and tier has no default rule	Re-add the rule, or set a tier `defaultRule`
`403 Unknown tier`	Customer's `tierId` doesn't exist	Fix the customer's type in the admin UI
`400 invalid metadata JSON`	`metadata` form field isn't valid JSON	Send e.g. `{"key":"value"}`
`413 ... file too large`	Attachment over 25 MB	Split or compress the file
Billing run: Stripe auth error	Placeholder/wrong key in `.env`, or key from a different Stripe account	Use the `sk_test` / `sk_live` key from the account that holds the meter
Stripe shows `METER_NOT_FOUND` invalid events	Meter missing or event name mismatch	Meter event name must be exactly `zappier.api_cents`
`another run holds the lock, abort run`	Overlapping runs, or a crash left a stale lock	Safe by design; stale locks expire after 1 hour
Admin UI: "invalid username or password"	Wrong credentials, or env overrides changed them	Check `ADMIN_KEY` / `DEMO_ADMIN_PASSWORD`; sign in again
Admin UI: "Session expired"	Server restarted (admin tokens are in-memory)	Sign in again
Portal login: `totp_required`	2FA is enabled on the account	Enter the current 6-digit authenticator code
Portal: "invalid or expired session"	7-day session expired	Sign in again
Portal reload didn't credit	Real `STRIPE_SECRET_KEY` configured → PaymentIntent awaits confirmation	Balance credits when the payment confirms; in dev (placeholder key) credit is instant
Invoice went straight to `paid` on Issue	Customer's prepaid balance fully covered the amount due	Working as designed — balance was drawn down