Document Navigation

ZephyrCart

        - **Installation**
        - **Your first product**
        - **A test checkout, end to end**
    - **Intro**
    - **Changelog**

Projects
ZephyrCart
Intro

Intro

ZephyrCart is a headless commerce platform: storefronts call our HTTP API, we own carts, checkouts, orders, customers, and the tax + payment plumbing behind them. We deliberately ship no embeddable admin UI: the expectation is that you build your own.

What this documentation covers

- The HTTP API at `https://api.zephyrcart.io/v1`
- Webhook events and signature verification
- Official SDKs (TypeScript, Python, Go, Ruby) and the language-agnostic OpenAPI spec
- Reference architectures for the three deployment shapes we actively support

What it does not cover

- The marketing site at `zephyrcart.io` (out of scope; talk to the growth team)
- The legacy `v0` API — scheduled for end of life 2026-12-31, migration guide forthcoming
- Internal runbooks (those live on the intranet, authenticated)

A note on conventions

Every endpoint in this documentation is shown with `curl` because every visitor can read it. Language-specific snippets follow in the SDK section. When an example uses a placeholder we wrap it in angle brackets, e.g. `<api-key>` — never copy it verbatim.

Changelog

Every change that touches the public API surface is recorded here, newest first. Internal changes that don't affect integrators live in the intranet release notes instead.

This document deliberately collects many small commits so the **Revisions** view on this page has meaningful history to demonstrate — open the right-rail History panel.

2026-05-30 — Refunds API

`POST /v1/orders/{id}/refunds` now accepts a structured `reason` field. Allowed values: `requested_by_customer`, `duplicate`, `fraudulent`. The previous free-text `note` field is deprecated and will be removed in v2 (end of 2026); use `metadata.note` to carry context across.

2026-05-22 — Webhook delivery v2

Webhook deliveries now retry with exponential backoff up to 24 hours instead of the prior 4 hours. The `X-ZephyrCart-Delivery-Attempt` header now starts at `1` (was `0`) for clarity.

2026-05-15 — Tax zones in Sweden

Added `tax_zone.se-*` for Swedish counties to support the new VAT reporting requirements that took effect on May 1.

2026-05-08 — Stripe Tax handoff

When `payment_method.stripe.use_stripe_tax = true`, tax calculation is delegated to Stripe Tax instead of ZephyrCart's internal engine. The order's `tax_calculation_source` field reflects which engine ran.

2026-04-30 — Customer merge

`POST /v1/customers/{id}/merge` now consolidates carts, orders, and stored payment methods from the target into the source customer. Previously only the email and shipping address were merged.

2026-04-22 — Bulk cart fetch

`GET /v1/carts?ids=...` supports up to 100 IDs per request (was 25). Useful for storefronts that want to reconcile carts after a partial outage.

2026-04-14 — Idempotency keys

All `POST` endpoints now accept the `Idempotency-Key` header. Retries within 24 hours with the same key return the original response instead of creating a duplicate resource.

Products

A product is a sellable thing. Variants live inside products; if you sell tea in three sizes, that's one product with three variants.

Object

Field	Type	Notes
`id`	string	`prod_…`, ULID
`name`	string	Human-readable, # 256 chars
`sku`	string?	Optional, must be unique within tenant
`description`	string?	Markdown, # 16 KiB
`default_price_cents`	integer	Minor units of `currency`
`currency`	string	ISO 4217, e.g. `EUR`
`inventory`	integer	`-1` means infinite (digital goods)
`status`	enum	`active`, `archived`, `draft`
`metadata`	object	# 50 key/value pairs, string values # 500 chars
`created_at`	timestamp	RFC 3339, UTC
`updated_at`	timestamp	RFC 3339, UTC

Endpoints

Method	Path	Purpose
`GET`	`/v1/products`	List, paginated
`POST`	`/v1/products`	Create
`GET`	`/v1/products/{id}`	Retrieve
`PATCH`	`/v1/products/{id}`	Update (partial; `null` clears)
`DELETE`	`/v1/products/{id}`	Archive (soft); use `?hard=true` to delete
`POST`	`/v1/products/{id}/variants`	Add a variant

Filtering

```
# Active products in EUR, updated in the last day, page of 50
curl -sS "https://api.zephyrcart.io/v1/products" \
  --get \
  --data-urlencode "status=active" \
  --data-urlencode "currency=EUR" \
  --data-urlencode "updated_after=2026-05-30T00:00:00Z" \
  --data-urlencode "limit=50" \
  -H "Authorization: Bearer $ZEPHYR_API_KEY"
```

Common errors

Code	Cause	Fix
`validation.invalid_currency`	`currency` not ISO 4217	Use a valid 3-letter uppercase code
`validation.sku_already_exists`	Duplicate SKU within tenant	Pick a unique SKU or update the existing product
`inventory.negative_not_allowed`	`inventory < 0` and product is not digital	Pass `inventory=-1` only for digital goods
`metadata.too_many_keys`	More than 50 metadata keys	Consolidate or drop unused metadata

---

Orders

An order is a committed cart with a charged payment method. Orders are immutable in all fields except `status`, `fulfilment`, and `metadata`.

Object

Field	Type	Notes
`id`	string	`order_…`, ULID
`cart_id`	string	`cart_…`, frozen at order creation
`customer_id`	string?	`cust_…` if a known customer
`line_items[]`	array	Snapshot of cart at order creation
`currency`	string	ISO 4217
`subtotal_cents`	integer	Before tax + shipping
`tax_total_cents`	integer	
`shipping_total_cents`	integer	
`total_cents`	integer	Charged to the customer
`tax_calculation_source`	enum	`zephyr`, `stripe_tax`, `avalara`
`status`	enum	`created`, `paid`, `partially_refunded`, `refunded`, `cancelled`
`livemode`	boolean	`false` in test mode
`placed_at`	timestamp	When the customer confirmed the order

Endpoints

Method	Path	Purpose
`GET`	`/v1/orders`	List, paginated
`GET`	`/v1/orders/{id}`	Retrieve
`PATCH`	`/v1/orders/{id}`	Update (status, fulfilment, metadata only)
`POST`	`/v1/orders/{id}/refunds`	Refund full or partial
`POST`	`/v1/orders/{id}/cancel`	Cancel before fulfilment

Refunds

```
curl -sS https://api.zephyrcart.io/v1/orders/order_01HXY8.../refunds \
  -H "Authorization: Bearer $ZEPHYR_API_KEY" \
  -H "Idempotency-Key: $(uuidgen)" \
  -d '{
    "amount_cents": 1490,
    "reason": "requested_by_customer",
    "metadata": { "ticket": "ZD-31427" }
  }'
```

`reason` is one of `requested_by_customer`, `duplicate`, `fraudulent`. The deprecated free-text `note` field still works but will be removed in v2 — see the Changelog 2026-05-30.

Cancellation

An order can be cancelled before its first fulfilment is recorded. After that, you refund instead. The transition is irreversible.

```
curl -X POST https://api.zephyrcart.io/v1/orders/order_…/cancel \
  -H "Authorization: Bearer $ZEPHYR_API_KEY"
```

Webhook events

- `order.created` — fired once, on placement.
- `order.updated` — fired on any mutable-field change.
- `order.refunded` — fired on each refund, including partials.
- `order.cancelled` — fired once.

Full reference: Webhooks.

Customers

A customer is the buyer-side identity. Customers can have many orders, many addresses, and many stored payment methods.

Object

Field	Type	Notes
`id`	string	`cust_…`
`email`	string	Unique within tenant, case-folded
`name`	string?	Display name
`phone`	string?	E.164
`addresses[]`	array	Default address has `is_default: true`
`default_currency`	string	ISO 4217
`metadata`	object	
`created_at`	timestamp	

Endpoints

Method	Path	Purpose
`GET`	`/v1/customers`	List
`POST`	`/v1/customers`	Create
`GET`	`/v1/customers/{id}`	Retrieve
`PATCH`	`/v1/customers/{id}`	Update
`DELETE`	`/v1/customers/{id}`	Anonymise (GDPR-compliant)
`POST`	`/v1/customers/{id}/merge`	Merge target into source

Merging customers

A common shape: a returning customer signs up with a different email and the storefront creates a new identity. Merge consolidates everything onto the older record.

```
curl -sS https://api.zephyrcart.io/v1/customers/cust_OLDER/merge \
  -H "Authorization: Bearer $ZEPHYR_API_KEY" \
  -d '{ "target": "cust_NEWER" }'
```

What moves:

- All carts, orders, and refunds.
- All stored payment methods (deduplicated by fingerprint).
- All addresses (deduplicated by normalised form).
- All metadata (`target` wins on conflict).

The `target` customer is then deleted. The operation is not reversible — keep a backup if you need one.

Anonymisation

`DELETE /v1/customers/{id}` does not hard-delete. It clears PII (`email`, `name`, `phone`, `addresses`), marks the record `status=anonymised`, and preserves the order history for accounting. Hard deletion requires a separate, audited support ticket.

Webhooks

ZephyrCart pushes events to endpoints you register. Endpoints must respond `2xx` within 10 seconds to count as delivered.

Register an endpoint

```
curl -sS https://api.zephyrcart.io/v1/webhook_endpoints \
  -H "Authorization: Bearer $ZEPHYR_API_KEY" \
  -d '{
    "url": "https://hooks.example.com/zephyr",
    "events": ["order.created", "order.refunded"],
    "description": "ERP bridge — production"
  }'
```

The response includes a `signing_secret` starting `whsec_…`. Store it; you'll need it to verify signatures.

Verify signatures

Every delivery carries:

- `X-ZephyrCart-Delivery-Id` — unique ULID per delivery
- `X-ZephyrCart-Delivery-Attempt` — `1` on first send, increments on retry
- `X-ZephyrCart-Timestamp` — Unix seconds, used in the signature
- `X-ZephyrCart-Signature` — `t=<ts>,v1=<hex-hmac-sha256>`

The signed string is `<timestamp>.<raw-request-body>`. Compute HMAC-SHA256 with the signing secret and compare in constant time:

```
import hmac, hashlib, time

def verify(payload: bytes, header: str, secret: str, tolerance_seconds: int =
300) -> bool:
    parts = dict(p.split("=", 1) for p in header.split(","))
    ts, sig = int(parts["t"]), parts["v1"]
    if abs(time.time() - ts) > tolerance_seconds:
        return False
    expected = hmac.new(
        secret.encode(),
        f"{ts}.".encode() + payload,
        hashlib.sha256,
    ).hexdigest()
    return hmac.compare_digest(expected, sig)
```

A request that fails verification must be rejected with HTTP `401`. Do not log the body.

Retries

Deliveries retry on any `non-2xx` or timeout, with exponential backoff capped at 24 hours total — roughly `1m, 5m, 15m, 1h, 3h, 6h, 12h`, then we give up and mark the delivery `failed`. Persistent failures put the endpoint in `status=failing` after 24 hours; the dashboard shows the last 100 delivery attempts.

Event catalogue

Event	When fired
`cart.created`	New cart
`cart.updated`	Line items, addresses, or coupons changed
`cart.abandoned`	No activity for 60 minutes
`checkout.session.created`	Hosted checkout URL minted
`checkout.session.completed`	Payment authorised
`order.created`	Placed; first webhook after `checkout.completed`
`order.updated`	Mutable-field change
`order.refunded`	Each refund, full or partial
`order.cancelled`	Before fulfilment
`customer.created`	
`customer.merged`	`target` is folded into `source`
`product.created`	
`product.archived`	Soft-delete
`inventory.low`	Configurable threshold per product

How a request flows

A `POST /v1/checkouts` from a storefront in Berlin to the `eu-frankfurt` region, with timings representative of what we see at p50.

```
{"meta":"placeholder — this fenced block is rendered by the drawio escape hatch on the live server. The local app exposes an in-app editor for it."}
```

The hops:

1. **Storefront SDK** issues the HTTPS request. (~1 ms TLS terminate, browser-side.)

2. **Edge** (Cloudflare in the user's POP, Frankfurt for Berlin) does WAF + DDoS, holds the connection. (~3 ms.)
3. **Regional load balancer** routes to the API gateway. (~1 ms.)
4. **API gateway** authenticates the bearer token, applies rate limits, attaches the tenant context. (~2 ms.)
5. **Checkout service** validates the cart, calls Tax and Inventory in parallel, persists the session, signs the hosted-checkout URL. (~30 ms — the bulk of the budget.)
6. **PostgreSQL** writes the checkout session, commits. (~5 ms within the region.)
7. **Response** travels back through the gateway # LB # edge # SDK. (~5 ms aggregate.)

End-to-end p50 from Berlin is ~50 ms. p99 is dominated by tax calculation and ranges 80–140 ms.

What can go wrong at each hop

Hop	Failure mode	What we do
Edge	DDoS	Cloudflare absorbs; legit traffic unaffected
Gateway	Rate limit exceeded	Return 429 with `Retry-After`
Checkout service	Tax service down	Fail open with a stub, flag the order for review
PostgreSQL primary	Failover (rare, ~30 s)	Gateway returns 503 with `Retry-After: 5`
Response back	Network blip	SDK retries idempotently

Data model

The relationships between the top-level resources. Cardinality is shown next to the line; `1..n` means one customer can have many orders, never the inverse.

```
{"meta":"placeholder — this fenced block is rendered by the drawio escape hatch on the live server. The local app exposes an in-app editor for it."}
```

A textual version of the same shape, for screen readers and grep-ability:

- A **tenant** owns everything below.
- A **customer** belongs to one tenant. Has `1..n` orders, `1..n` carts, `0..n` payment methods, `1..n` addresses.
- A **product** belongs to one tenant. Has `1..n` variants. Variants carry the price and the inventory.
- A **cart** belongs to one tenant, optionally to one customer. Has `1..n` line items, each pointing at a variant.
- A **checkout session** belongs to one cart. Is the immutable, signed handoff to the hosted payment page.

- An **order** is the immutable promotion of a cart at the moment of payment. It carries a snapshot of the line items, not a reference, so price changes do not retroactively alter historical orders.
- A **refund** belongs to one order. Up to N refunds can sum to # the order total.

Invariants

- Currency is set on the cart at creation and cannot change. A cart cannot mix currencies.
- A product's `currency` does not constrain the cart's currency — pricing tables handle the cross-product per-currency price.
- Order line items snapshot product name, SKU, and unit price at the moment the order is created. Editing the product after the fact does not retroactively change the order.
- Customer email is unique per tenant, case-folded.

Deployment topology

ZephyrCart runs in four regions:

Region key	Cloud + AZ count	Latency to next region
`eu-frankfurt`	GCP `europe-west3`, 3 zones	13 ms to `eu-dublin`
`eu-dublin`	GCP `europe-west1`, 3 zones	13 ms to `eu-frankfurt`
`us-east`	GCP `us-east4`, 3 zones	79 ms to `eu-dublin`
`ap-singapore`	GCP `asia-southeast1`, 3 zones	168 ms to `eu-frankfurt`
`ap-tokyo` (beta)	GCP `asia-northeast1`, 3 zones	73 ms to `ap-singapore`

Within a region: full active/active across all three zones, PostgreSQL with synchronous replicas, Redis with sentinel.

Data residency

A tenant's data stays in its home region. There is no cross-region replication of customer PII or order data. Aggregated, de-identified telemetry flows to a central observability bucket in `eu-frankfurt` for the platform team — see the Data handling policy for the exact list of fields.

Region failover

There is no automated cross-region failover. If a region is unavailable, the dashboard surfaces a status page entry and storefronts in that region fail over to a degraded read-only mode (cached catalog, no new checkouts) until the region returns. This is a deliberate choice — replicating PII across regions would put us into legal regimes the company does not want to operate under.

Multi-region tenants

A small number of Enterprise customers run in two regions concurrently with their own application-layer sync. Talk to your solutions engineer; this is not a self-serve feature.

Stripe

Stripe is the default payment provider for new tenants. The integration covers:

- Card payments via Stripe Payment Intents
- Stored payment methods on the customer record
- Stripe Tax handoff for jurisdictions where we don't run our own engine
- Refunds, partial refunds, and disputes

Set up your Stripe account

Two keys, in your tenant's **Settings # Payments # Stripe** page:

Field	Where to find it
Publishable key	Stripe Dashboard # Developers # API keys # "Publishable key"
Secret key	Stripe Dashboard # Developers # API keys # "Secret key" (test or live)
Webhook signing secret	We register the webhook with Stripe on save and store the secret

When you save, ZephyrCart calls Stripe to register a webhook endpoint at `https://api.zephyrcart.io/v1/internal/stripe/webhook` scoped to your account.

Stripe Tax handoff

Inside the EU, ZephyrCart's tax engine handles VAT. In the US sales-tax jurisdictions and a few others, Stripe Tax is the smoother experience. Flip per-tenant:

```
zephyr settings set payments.stripe.use_stripe_tax true
```

Effect:

- Tax is computed on the Stripe side during Payment Intent creation
- The order's `tax_calculation_source` is `stripe_tax` (was `zephyr`)
- The line-item-level breakdown is preserved verbatim from Stripe

Code sample — TypeScript SDK

```
import { Zephyr } from "@zephyrcart/sdk";

```

```
const client = new Zephyr({ apiKey: process.env.ZEPHYR_API_KEY! });

const checkout = await client.checkouts.create({
  cart: cartId,
  payment: {
    method: "stripe",
    return_url: "https://shop.example.com/return",
  },
});

// checkout.checkout_url is the hosted page; redirect the customer there
```

Refund flow

A refund triggered through ZephyrCart's API also refunds through Stripe automatically. There is no "refund only in Stripe" mode — refunds must always originate in ZephyrCart so the order record stays the source of truth.

```
curl -sS https://api.zephyrcart.io/v1/orders/order_…/refunds \
  -H "Authorization: Bearer $ZEPHYR_API_KEY" \
  -d '{ "amount_cents": 1490, "reason": "requested_by_customer" }'
```

Disputes

When a chargeback is opened in Stripe, we emit `order.disputed`. The order's status is unchanged; the dispute is a sidecar object you query via `GET /v1/orders/{id}/disputes`.

Migrating from Shopify

A migration takes between an afternoon (10 SKUs, no historical orders) and several days (100k SKUs, two years of order history). Three phases.

Phase 1 — Export from Shopify

Use Shopify's GraphQL Admin API. The CLI has a helper that runs the exact queries we need:

```
zephyr migrate shopify pull \
  --shop my-shop.myshopify.com \
  --token shpat_… \
  --out ./shopify-export/
```

This writes one JSON file per resource type: `products.jsonl`, `customers.jsonl`, `orders.jsonl`, `collections.jsonl`. Each line is one Shopify record, untouched.

Phase 2 — Map and validate

Mapping is where most of the work lives. The CLI's default rules cover the 80 % case:

Shopify field	ZephyrCart field
`product.handle`	`product.sku` (if no other SKU set)
`product.product_type`	First collection assignment
`variant.sku`	`variant.sku`
`variant.price` (string)	`variant.default_price_cents` (× 100)
`customer.email`	`customer.email`
`customer.default_address`	`customer.addresses[0]`, `is_default: true`
`order.line_items[]`	`order.line_items[]`

Override any rule with a `mapping.toml` next to the export:

```
[product]
sku_from = "variants[0].sku"

[customer]
exclude_if_marketing_opt_out = false
```

Validate before importing:

```
zephyr migrate shopify validate --in ./shopify-export/ --mapping ./mapping.
toml
```

The validator reports field-by-field anything ZephyrCart can't accept (currencies it doesn't recognise, SKUs longer than its limit, etc.). Fix at the source, re-export, re-validate.

Phase 3 — Import

```
zephyr migrate shopify push --in ./shopify-export/ --mapping ./mapping.toml --
concurrency 8
```

The CLI streams resources to the API with idempotency keys derived from the Shopify ID, so a re-run of an interrupted import is safe. Progress is checkpointed to `./shopify-export/.zephyr-state`.

What we do not migrate

- **Shopify scripts** — no equivalent in ZephyrCart; checkout rules express most of what scripts do.
- **Shopify Markets** — model the same shape with our tax zones.
- **Custom apps' metafields** — these come through as `product.metadata` keys prefixed with the app handle; review and rename before going live.

ERP bridge

Order data flows from ZephyrCart into your ERP (Microsoft Dynamics 365 Business Central or SAP Business One in 90 % of the deployments we see). The recommended shape is a thin worker that subscribes to `order.created` and `order.refunded` webhooks and translates them into ERP API calls.

Reference architecture

```
ZephyrCart #### webhook ####  Bridge worker  #### ERP API #####  Dynamics /
SAP
                               (your code)
                                   #
                                   #
                             Dead-letter queue
                             (for replays)
```

Why a worker, not a direct webhook to the ERP

Three reasons:

1. ERPs do not authenticate webhooks the same way we sign them. The worker terminates the signature, translates, and reauthenticates with the ERP's own scheme.
2. Field mapping changes more often than the ZephyrCart event schema. Keeping it in your worker lets you ship mappings without touching either side.
3. ERPs go down for maintenance windows. The worker is where you buffer.

Recommended technology

- A small Cloud Run / AWS Lambda / Azure Function — request load is exactly "one execution per order event," which is the cheapest serverless shape there is.
- A persistent queue between webhook receipt and ERP push (Cloud Tasks, SQS, Service Bus) so the ERP outage doesn't lose data.

Idempotency at the ERP

Send the ZephyrCart `order.id` as the external reference on the ERP record. If the ERP returns "already exists," consider that a success and ack the webhook.

A worked example

A Node.js worker on Google Cloud Run, posting into Dynamics 365 Business Central:

```
import { verifyZephyrSignature } from "./signing";

export async function handle(req: Request): Promise<Response> {
  const body = await req.text();
  if (!verifyZephyrSignature(body, req.headers.get("x-zephyrcart-
signature")!)) {
    return new Response("bad signature", { status: 401 });
  }
  const event = JSON.parse(body);
  if (event.type !== "order.created") {
    return new Response("ignored", { status: 200 });
```

```
  }

  await dynamicsClient.createSalesOrder({
    externalReference: event.data.id,                // idempotency anchor
    customerEmail: event.data.customer.email,
    currency: event.data.currency,
    lines: event.data.line_items.map(toErpLine),
  });

  return new Response("ok", { status: 200 });
}
```

Failure modes worth handling

Failure	How to recover
ERP returns 5xx	Return 5xx from the worker — ZephyrCart retries
ERP returns 4xx because the SKU is unknown	Log to dead-letter queue, page the catalog team
Worker times out at 10 seconds	Ack early, push to your own queue, process async
Signature verification fails	Return 401; investigate clock skew first

---

Installation

Get the CLI

The CLI is a single Go binary, signed and notarised on macOS, signed via Authenticode on Windows.

```
# macOS — Homebrew
brew install zephyrcart/tap/zephyr

# Linux — apt
curl -fsSL https://pkg.zephyrcart.io/apt/key.gpg | sudo gpg --dearmor -o /usr
/share/keyrings/zephyr.gpg
echo "deb [signed-by=/usr/share/keyrings/zephyr.gpg] https://pkg.zephyrcart.io
/apt stable main" \
  | sudo tee /etc/apt/sources.list.d/zephyr.list
sudo apt update && sudo apt install -y zephyr-cli

# Windows — Scoop
scoop bucket add zephyr https://github.com/zephyrcart/scoop-bucket
scoop install zephyr
```

Confirm:

```
zephyr --version
# zephyr 1.18.3 (build 7c2a91f, 2026-05-19)
```

Authenticate

```
zephyr login
```

The CLI opens your browser, you sign into the dashboard, and the OAuth dance writes a token to `~/.config/zephyr/credentials.toml`. The token is scoped to a single tenant and a single role.

For CI use a project-level API key instead — see API keys.

Pick a region

ZephyrCart runs in four regions. Pick the closest to your customers; latency from anywhere else is the latency budget you spend forever.

Region key	Location	Status
`eu-frankfurt`	Frankfurt, Germany	GA
`eu-dublin`	Dublin, Ireland	GA
`us-east`	Virginia, USA	GA
`ap-singapore`	Singapore	GA

Set the default:

```
zephyr config set region eu-frankfurt
```

Now jump to Creating your first product.

Your first product

Two ways: CLI or API. Use whichever feels familiar.

CLI

```
zephyr products create \
  --name "Filter coffee, 250g" \
  --sku coffee-250 \
  --price-cents 1490 \
  --currency EUR \
  --inventory 200
```

Output:

```
{
  "id": "prod_01HXY4M9C0AVE0M1QGPS9XK7T2",
  "name": "Filter coffee, 250g",
```

```
  "sku": "coffee-250",
  "default_price_cents": 1490,
  "currency": "EUR",
  "inventory": 200,
  "status": "active",
  "created_at": "2026-05-31T09:14:12Z"
}
```

API

The same call over HTTP:

```
curl -sS https://api.zephyrcart.io/v1/products \
  -H "Authorization: Bearer $ZEPHYR_API_KEY" \
  -H "Idempotency-Key: $(uuidgen)" \
  -H "Content-Type: application/json" \
  -d '{
    "name": "Filter coffee, 250g",
    "sku": "coffee-250",
    "default_price_cents": 1490,
    "currency": "EUR",
    "inventory": 200
  }'
```

The `Idempotency-Key` is optional but recommended. Identical retries within 24 hours return the same response without creating a duplicate product. See the Idempotency reference.

Verify in the dashboard

The product should appear in **Catalog # Products** within ~1 second. If it doesn't, you're probably authenticated against a different tenant — check `zephyr config get tenant`.

Next: Run a checkout end to end.

A test checkout, end to end

In test mode, no money moves and no order email is sent. The flow walks the same code paths as production, including tax calculation and webhook delivery.

1. Create a cart

```
zephyr carts create --line-item prod_01HXY4M9C0AVE0M1QGPS9XK7T2:2
# # cart_01HXY4QH6QMEN0SEN1GHX7YK9C
```

2. Attach a customer

```
zephyr carts attach-customer cart_01HXY4QH6QMEN0SEN1GHX7YK9C \
  --email "ali@example.com" \
  --shipping-address-de
```

The `--shipping-address-de` flag uses a built-in test address for Germany — good enough to make the tax engine happy.

3. Create a checkout session

```
zephyr checkouts create --cart cart_01HXY4QH6QMEN0SEN1GHX7YK9C
```

The response includes a `checkout_url`. In test mode that URL renders a sandbox payment form.

4. Complete with a test card

Open the `checkout_url`, enter card `4242 4242 4242 4242`, any future expiry, any CVC, any postal code. The session completes within ~2 seconds and emits two webhooks:

- `checkout.session.completed`
- `order.created`

See those land:

```
zephyr webhooks tail --event 'order.created'
```

What just happened

A new order resource exists at `GET /v1/orders/{order_id}`. In test mode it's flagged `livemode=false` and is invisible to the production dashboard. Test orders are pruned after 30 days; keep the order id somewhere if you want to refer to it later.

You're done. Read How a request flows next for the mental model behind the API.