

# Shopify Connector

## Feature Handbook

Everything the `baseup_shopify` module does — a bidirectional Shopify integration for Odoo 19 built on the GraphQL Admin API, with real-time webhooks, multi-warehouse inventory mapping, and direction-aware sync controls. Everything described here ships in the single module; there is nothing further to buy.

|                                             |                                          |                                          |
|---------------------------------------------|------------------------------------------|------------------------------------------|
| <div>MODULE</div> <div>baseup_shopify</div> | <div>VERSION</div> <div>19.0.2.0.0</div> | <div>ODOO</div> <div>19.0</div>          |
| <div>SHOPIFY API</div> <div>2026-04</div>   | <div>LICENSE</div> <div>OPL-1</div>      | <div>AUTHOR</div> <div>Baseup Labs</div> |

### Contents

|       |                              |
|-------|------------------------------|
| I     | What you have bought         |
| II    | How the connector is built   |
| III   | Connecting a store           |
| IV    | Webhooks & real-time updates |
| V     | Locations & warehouses       |
| VI    | Products: Shopify → Odoo     |
| VII   | Products: Odoo → Shopify     |
| VIII  | Variants, options & images   |
| IX    | Inventory & stock levels     |
| X     | Orders: Shopify → Odoo       |
| XI    | Draft orders & quotations    |
| XII   | Fulfillment & tracking       |
| XIII  | Refunds & cancellations      |
| XIV   | Customers & addresses        |
| XV    | Scheduled automation         |
| XVI   | Manual sync controls         |
| XVII  | Settings reference           |
| XVIII | Logs & troubleshooting       |
| XIX   | Guided tours                 |
| XX    | Access & security            |
| XXI   | Limits & licensing           |

Section I

What you have bought

The Shopify Connector links one or more Shopify stores to Odoo 19 and keeps products, inventory, orders, fulfillments and refunds aligned in both directions. It talks to Shopify's **GraphQL Admin API** (version `2026-04` ) for all business operations, falling back to REST only for the two endpoints that have no GraphQL equivalent: the connection test ( `/shop.json` ) and webhook registration ( `/webhooks.json` ).

Capability summary

| AREA          | SHOPIFY → ODOO                                     | ODOO → SHOPIFY                                     |
|---------------|----------------------------------------------------|----------------------------------------------------|
| Products      | Full catalogue import, incremental and full resync | Create & update, optional auto-push on save        |
| Variants      | Attributes, options, prices, images, SKUs          | Create missing, delete removed, price & image push |
| Inventory     | Pull on-hand levels (Shopify wins)                 | Push on-hand levels (Odoo wins), per-warehouse     |
| Orders        | Import with auto-confirm on payment                | Quotations exported as draft orders                |
| Draft orders  | Import as Odoo quotations, promoted on completion  | Push confirmed quotations                          |
| Fulfillment   | Tracking numbers written to Odoo pickings          | Fulfillment created on delivery validation         |
| Refunds       | Draft credit notes from refund webhooks            | —                                                  |
| Cancellations | Cancelled & voided orders cancel the Odoo SO       | —                                                  |
| Customers     | Partners & child addresses created on import       | Customer payload on exported draft orders          |
| Locations     | Shopify locations become Odoo warehouses           | Odoo warehouses become Shopify locations           |

Odoo dependencies

The module installs on top of these standard Odoo apps, all of which must be available:

| DEPENDENCY                   | WHY IT IS NEEDED                                                 |
|------------------------------|------------------------------------------------------------------|
| <code>sale_management</code> | Sale orders and quotations — the target of order import          |
| <code>website_sale</code>    | eCommerce product fields used by the catalogue mapping           |
| <code>stock</code>           | Warehouses, quants and pickings for inventory and fulfillment    |
| <code>product</code>         | Product templates, variants and attribute values                 |
| <code>account</code>         | Invoices and credit notes for the refund flow                    |
| <code>delivery</code>        | Carriers and <code>carrier_tracking_ref</code> for tracking sync |

## Where to find it in Odoo

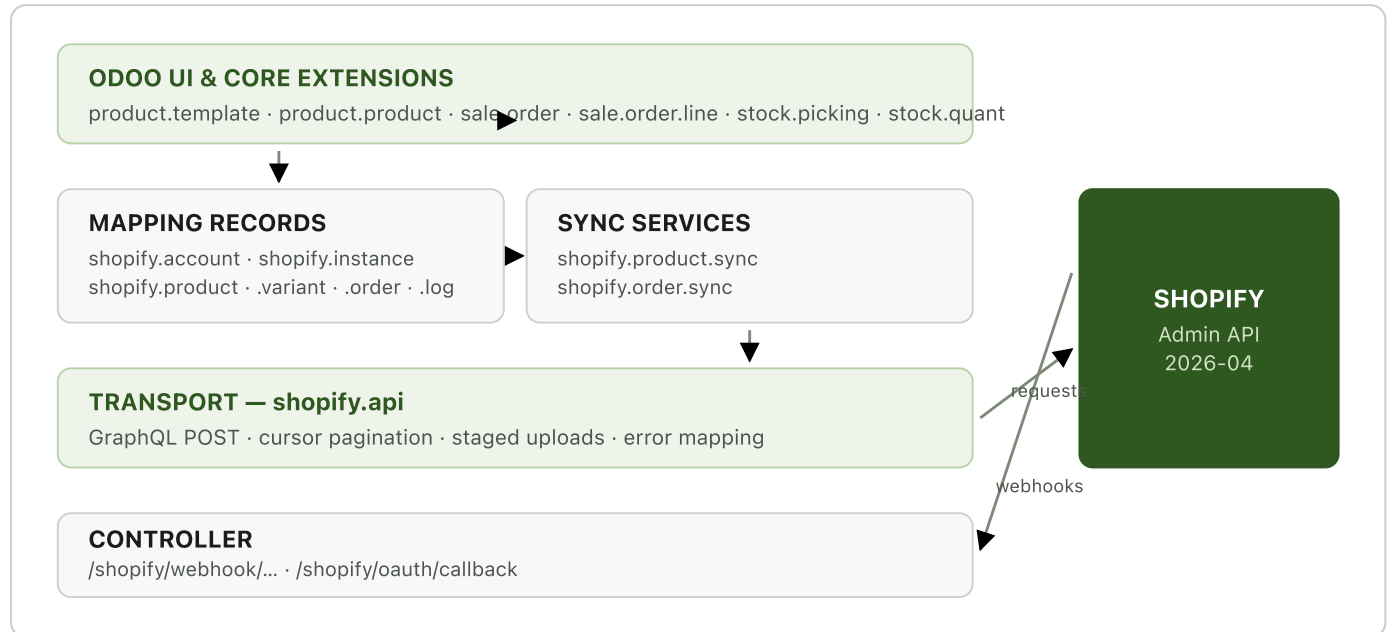
Installation adds a top-level **Shopify** app (visible to Sales Managers) with the following menu structure:

- **Shopify → Operations → Orders** — the `shopify.order` mapping records
- **Shopify → Operations → Sync Now** — the direction-aware sync wizard
- **Shopify → Operations → Logs** — the `shopify.log` audit trail
- **Shopify → Configuration → Shopify Accounts** — store credentials and per-store actions
- **Settings → Shopify Connector** — global sync defaults and tuning

## Section II

# How the connector is built

The module follows a strict separation between transport, business services and mapping records. No Odoo core addon is modified; all ERP behaviour is added through `_inherit` extensions inside this module.



Requests flow down through one transport model; Shopify pushes changes back into the webhook controller.

## Model responsibilities

| MODEL                                 | KIND      | RESPONSIBILITY                                                                                                              |
|---------------------------------------|-----------|-----------------------------------------------------------------------------------------------------------------------------|
| <code>shopify.account</code>          | Stored    | Store credentials, configuration, per-store action buttons, cron entry points                                               |
| <code>shopify.instance</code>         | Stored    | Runtime handle ( <code>_inherits</code> on the account) passed into every sync call; HMAC verification; location resolution |
| <code>shopify.api</code>              | Abstract  | GraphQL transport, cursor pagination, staged file uploads                                                                   |
| <code>shopify.product.sync</code>     | Abstract  | Product import/export, variant mapping, images, inventory push & pull                                                       |
| <code>shopify.order.sync</code>       | Abstract  | Order and draft-order import, fulfillment push, refund import                                                               |
| <code>shopify.product</code>          | Stored    | Shopify product GID ↔ <code>product.template</code> mapping, sync state                                                     |
| <code>shopify.product.variant</code>  | Stored    | Shopify variant & inventory-item GID ↔ <code>product.product</code> mapping                                                 |
| <code>shopify.order</code>            | Stored    | Shopify order GID ↔ <code>sale.order</code> mapping, financial & fulfillment status                                         |
| <code>shopify.location.mapping</code> | Stored    | Shopify location GID ↔ <code>stock.warehouse</code> pairs                                                                   |
| <code>shopify.log</code>              | Stored    | Operation audit trail                                                                                                       |
| <code>shopify.sync.wizard</code>      | Transient | Direction-aware manual sync                                                                                                 |
| <code>shopify.push.wizard</code>      | Transient | Product push with explicit store selection                                                                                  |

## Design rules the module follows

1. Odoo core addons ( `sale` , `stock` , `product` ) are never modified directly.
2. ERP behaviour is added through `_inherit` in this module only.
3. Mapping records stay separate from the sync service implementations.
4. Transport concerns live in `shopify.api` , never in a business service.

## Section III

# Connecting a store

Each Shopify store is one `shopify.account` record under **Shopify → Configuration → Shopify Accounts**. Multiple stores can be connected simultaneously; every mapping record is scoped to its instance.

## OAuth connection (recommended)

The connector implements Shopify's Authorization Code Grant flow for apps created in the Shopify Dev Dashboard. You supply the **Client ID** and **Client Secret**; the access token and webhook secret are filled in automatically.

### STEP 1

Enter Shop URL, Client ID and Client Secret, then save.

### STEP 2

Click **Connect to Shopify** — a random `state` token is generated and you are redirected to Shopify.

### STEP 3

Approve the requested scopes in Shopify.

### STEP 4

Shopify returns to `/shopify/oauth/callback`, which exchanges the code for an access token.

### STEP 5

The token and webhook secret are stored, status becomes **Connected**, and a `shopify.instance` is created.

The callback matches the returned `state` against the stored `oauth_state` and clears it on completion, so a stale or forged redirect cannot bind a token to an account.

### PUBLIC URL REQUIRED

The OAuth redirect and webhook callbacks must reach your Odoo instance over public HTTPS. Set `baseup_shopify.webhook_base_url` in **Settings → Technical → System Parameters** to your public domain. It takes precedence over `web.base.url`, so a development server can keep `web.base.url` pointed at a LAN address.

## Requested OAuth scopes

The connector requests exactly the scopes its features need:

| SCOPE PAIR                                                       | USED FOR                                      |
|------------------------------------------------------------------|-----------------------------------------------|
| <code>read_products</code> , <code>write_products</code>         | Catalogue import and export, variants, images |
| <code>read_orders</code> , <code>write_orders</code>             | Order import, cancellation detection          |
| <code>read_inventory</code> , <code>write_inventory</code>       | Inventory pull and push                       |
| <code>read_fulfillments</code> , <code>write_fulfillments</code> | Fulfillment creation and tracking updates     |
| <code>read_draft_orders</code> , <code>write_draft_orders</code> | Draft order import and quotation export       |
| <code>read_locations</code>                                      | Location discovery for warehouse mapping      |
| <code>write_webhooks</code>                                      | Webhook registration                          |

## Account fields

| FIELD                              | PURPOSE                                                                                                             |
|------------------------------------|---------------------------------------------------------------------------------------------------------------------|
| Shop Name                          | Display name; overwritten with the real shop name on a successful connection test                                   |
| Shop URL                           | <code>mystore.myshopify.com</code> — the <code>https://</code> prefix is added automatically                        |
| Client ID                          | OAuth client ID from the Shopify Dev Dashboard app                                                                  |
| Client Secret                      | OAuth client secret; also copied into the webhook secret on connect                                                 |
| Access Token                       | Filled by OAuth; can be entered manually for legacy custom apps                                                     |
| Webhook Secret                     | Used to verify the HMAC signature on every inbound webhook                                                          |
| Company                            | Odoo company that owns the imported records                                                                         |
| Warehouse                          | Default warehouse for imported orders and the legacy single-location fallback                                       |
| Pricelist, Sales Team              | Applied to imported sale orders                                                                                     |
| Sync Products / Orders / Inventory | Per-store master switches; defaults come from the global settings                                                   |
| Status                             | <div> <div>Disconnected</div> <div>Connected</div> <div>Error</div> </div> — set by connect, test and failure paths |

Credential fields ( `api_key` , `api_secret` , `webhook_secret` , `client_id` , `oauth_state` ) are restricted to the `base.group_system` group, so ordinary Sales Managers can operate the connector without seeing the secrets.

## Test Connection

**Test Connection** calls the REST `/shop.json` endpoint — the one check with no GraphQL equivalent. On success the account is marked **Connected**, the shop name is refreshed and a `shopify.instance` is created if one does not exist. On failure the account is marked **Error** and the underlying message is surfaced.

## Webhooks & real-time updates

Webhooks are what make the integration feel live: an order placed in Shopify appears in Odoo within seconds rather than waiting for the next cron tick.

### Registration

Click **Register Webhooks** on a connected account. The connector *deletes every existing webhook on the store first*, then registers the full topic list against your current base URL. This guarantees a clean slate and removes stale callbacks left by earlier instances.

Registration refuses to run unless the effective base URL is public HTTPS — `http://` , `localhost` and loopback addresses are rejected with an explanatory error rather than silently registering unreachable callbacks.

### Registered topics

| TOPIC                                   | EFFECT IN ODOO                                                      |
|-----------------------------------------|---------------------------------------------------------------------|
| <code>orders/create</code>              | Import the order, auto-confirming it when already paid              |
| <code>orders/updated</code>             | Refresh statuses, tracking, and cancel the SO when appropriate      |
| <code>orders/paid</code>                | Confirm the Odoo sale order                                         |
| <code>orders/cancelled</code>           | Cancel the linked Odoo sale order                                   |
| <code>orders/fulfilled</code>           | Record fulfillment status and tracking                              |
| <code>orders/partially_fulfilled</code> | Same handler as fulfilled                                           |
| <code>refunds/create</code>             | Create a draft credit note                                          |
| <code>products/create</code>            | Ignored unless the product is already mapped (prevents push echoes) |
| <code>products/update</code>            | Re-fetch the product via GraphQL and update Odoo                    |
| <code>products/delete</code>            | Handle removal of the Shopify product                               |
| <code>inventory_levels/update</code>    | Apply the new Shopify stock level in Odoo                           |
| <code>draft_orders/create</code>        | Create a draft Odoo quotation                                       |
| <code>draft_orders/update</code>        | Update the draft quotation                                          |

## Endpoints

| ROUTE                                                                  | NOTES                                                                       |
|------------------------------------------------------------------------|-----------------------------------------------------------------------------|
| <code>/shopify/webhook/account/&lt;account_id&gt;/&lt;topic&gt;</code> | Current endpoint — keyed on the account, so it survives instance recreation |
| <code>/shopify/webhook/&lt;instance_id&gt;/&lt;topic&gt;</code>        | Legacy endpoint, retained for backward compatibility                        |
| <code>/shopify/oauth/callback</code>                                   | OAuth code exchange (authenticated user route)                              |

Topic slugs in the URL use underscores. The controller maps them back to real Shopify topics by trying each underscore boundary, so multi-word left-hand topics such as `inventory_levels_update` → `inventory_levels/update` resolve correctly.

## Signature verification

Every request must carry a valid `X-Shopify-Hmac-Sha256` header. The connector computes an HMAC-SHA256 of the raw request body using the stored webhook secret and compares it with `hmac.compare_digest`. Requests with a missing secret, a bad signature or a malformed body are rejected with `401` or `400` and never reach a handler.

## Echo suppression

When Odoo pushes a product to Shopify, Shopify immediately fires `products/update` back. Left unchecked this causes write contention and pointless re-imports. The connector suppresses these echoes:

- `products/create` for an unmapped product is skipped outright — it is almost always the echo of an outbound push.
- `products/update` for an unmapped product is skipped, preventing duplicate Odoo products during push races.
- Within **20 seconds** of an outbound push, an update whose `updated_at` is not newer than our own `sync_date` is treated as an echo and skipped. A genuinely newer change still gets processed.
- The mapping row is probed with `FOR UPDATE SKIP LOCKED`; if our own outbound push holds the row, the webhook backs off instead of fighting for it.

## Concurrency handling

Webhook processing runs under PostgreSQL `REPEATABLE READ`, where a concurrent write causes a serialization failure. Each webhook is attempted up to **four times** with exponential backoff (0.25s, 0.5s, 1s, capped at 2s). Because a savepoint rollback keeps the stale snapshot, the connector performs a *full* transaction rollback between attempts so each retry gets a fresh snapshot. Non-serialization errors are logged to `shopify.log` immediately.

The endpoint returns `200 OK` even after a handled failure, so Shopify does not enter a retry storm; the failure is recorded in the Odoo log for you to act on.

## Section V

# Locations & warehouses

Shopify tracks stock per *location*; Odoo tracks it per *warehouse*. The `shopify.location.mapping` table pairs them, which is what makes true multi-warehouse inventory sync possible.

## Sync Locations Shopify → Odoo

For each active Shopify location, the button:

- skips it if a mapping with that GID already exists;
- links it to an existing Odoo warehouse when the names match (case-insensitive);
- otherwise creates a new warehouse, deriving a unique 5-character code from the location name;
- enables `fulfillsOnlineOrders` on every mapped location so it can serve online orders.

## Push Warehouses to Shopify Odoo → Shopify

Creates Shopify locations for Odoo warehouses that have no mapping yet, using the warehouse partner's address. Already-mapped warehouses are left alone. Use it after **Sync Locations** when Odoo has warehouses that Shopify does not know about.

## Mapping constraints

Two unique constraints keep the pairing unambiguous:

- each warehouse may be mapped only once per Shopify account;
- each Shopify location may be mapped only once per account.

## Legacy single-location fallback

When no mapping rows exist, the connector falls back to the account's **Warehouse** field paired with a cached `shopify_location_gid`. If no GID is cached it queries Shopify's active locations and stores the first one; if the store has no active location at all it creates one from the warehouse address. Configuring explicit mappings is strongly preferred — the fallback exists only for single-warehouse upgrades from earlier versions.

## Products: Shopify → Odoo

### Two import modes

Product import runs in one of two modes, chosen automatically per account.

| MODE               | WHEN IT RUNS                                            | HOW IT WALKS THE CATALOGUE                                                                                                                                      |
|--------------------|---------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Incremental</b> | Normal day-to-day operation                             | GraphQL filter <code>updated_at:&gt;=...</code> , starting a configurable lookback window before the last successful sync so nothing falls through the boundary |
| <b>Full resync</b> | First-ever sync, or after<br><b>Full Product Resync</b> | ID-cursor walk: products are processed in numeric-ID order and the last processed ID is persisted, so the run resumes exactly where it stopped                  |

### Why an ID cursor

A full catalogue import can exceed the Odoo cron time limit. Rather than restarting from scratch each tick, the account stores `full_resync_cursor`:

- `False` / unset — no resync in progress, incremental mode applies;
- `"0"` — a walk is starting from the beginning;
- a numeric ID — resume after that product.

Each cron tick advances the cursor. When the walk completes, the cursor is cleared and `last_product_sync_date` is set, returning the account to incremental mode. **Full Product Resync** therefore returns immediately and queues the work for the next auto-sync cycle instead of blocking the UI.

### What an import creates

For each Shopify product node the connector:

1. finds the existing `shopify.product` mapping by GID, or the Odoo product referenced by the `odoo-product-tmpl-...` tag Shopify carries from an earlier push;
2. creates or updates the `product.template`;
3. builds the full variant structure — attributes, values and variants — to match Shopify's options;
4. maps every variant into `shopify.product.variant`, storing variant and inventory-item GIDs;
5. pulls product and variant images unless images are skipped;
6. optionally pulls inventory levels for the mapped variants.

## Performance controls

| CONTROL                          | EFFECT                                                                                                                                                                                               |
|----------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Sync batch size                  | Records processed before a flush; also the batch ceiling in batch mode                                                                                                                               |
| Skip product images in bulk sync | Omits image download during bulk runs — by far the most expensive part of a large import                                                                                                             |
| Bulk context flags               | Bulk runs set <code>shopify_skip_auto_push</code> , <code>shopify_bulk_product_sync</code> and <code>shopify_skip_inventory_pull</code> so an import cannot trigger an outbound push back to Shopify |

## Error recovery

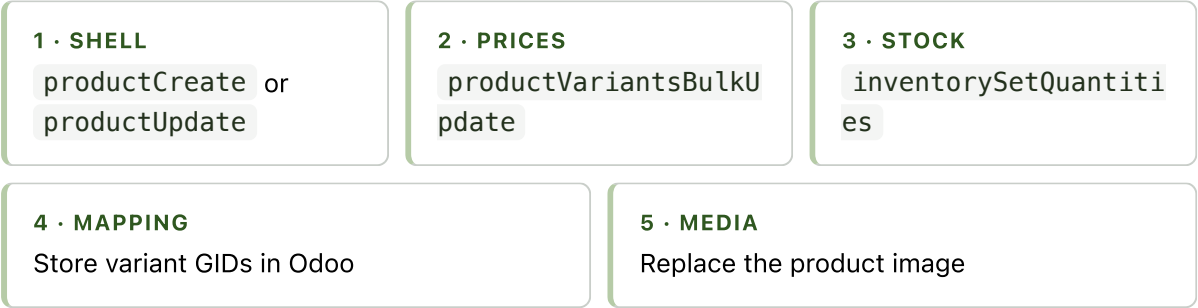
A product that fails mid-import is flagged `state = 'error'` on its mapping rather than aborting the run. Two mechanisms recover them:

| MECHANISM                            | BEHAVIOUR                                                                                                                                                                                                                                                                                                                    |
|--------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Retry Errored Products (hourly cron) | Re-attempts every errored mapping in a fresh transaction. Most failures are serialization conflicts, which a new snapshot resolves.                                                                                                                                                                                          |
| Clean Up Errored Products (button)   | Checks each errored mapping against Shopify. Where the Shopify product no longer exists, the Odoo product is archived and the orphaned mapping deleted. Mappings whose Shopify product still exists are left for the retry cron. The notification reports exactly how many were checked, archived, deleted and left pending. |

# Products: Odoo → Shopify

## The export sequence

Pushing one `product.template` runs a fixed five-step sequence, guarded by a per-product advisory lock so two workers cannot export the same product concurrently:



## Field mapping

| SHOPIFY FIELD   | ODOO SOURCE                                                                                                    |
|-----------------|----------------------------------------------------------------------------------------------------------------|
| title           | name                                                                                                           |
| descriptionHtml | description_sale                                                                                               |
| vendor          | Company name                                                                                                   |
| productType     | Product category name                                                                                          |
| tags            | On creation only: <code>odoo-product-tmpl-&lt;id&gt;</code> , used to re-identify the product on later imports |

## Self-healing on a missing product

If a mapped Shopify product has been deleted, `productUpdate` fails. The connector recognises that specific error, drops the stale mapping and recreates the product instead of leaving the mapping permanently broken.

## Automatic pushes

Two settings control whether saving in Odoo reaches Shopify:

| SETTING                          | TRIGGER                                                                                                                                                                                                                                                                                      |
|----------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Auto-push newly created products | Exports every newly created product template. Default off.                                                                                                                                                                                                                                   |
| Auto-update products on save     | Re-exports an already-linked product when one of these fields changes: <code>name</code> , <code>description_sale</code> , <code>categ_id</code> , <code>image_1920</code> , <code>active</code> , <code>default_code</code> , <code>barcode</code> , <code>list_price</code> . Default off. |

#### MULTI-STORE GUARD

Auto-push of *new* products is deliberately skipped when more than one store is connected — silently pushing a new product to every store is never the right default. Use **Push to Shopify** on the product to choose the target store.

### Push to Shopify wizard

Available on the product form and as a list action. It requires an explicit **Shopify Store** selection (pre-filled when exactly one store is connected) and reports how many products pushed and how many failed. Each product is pushed inside its own savepoint, so one failure cannot roll back the rest.

### Attribute changes trigger a re-export

Editing an attribute value's `name`, `price_extra`, `image_1920` or `sequence` re-exports every affected product template, so variant names, prices and images stay aligned. Adding, changing or removing a template attribute line does the same. Both paths respect the **Auto-update products on save** setting.

### Deleting a product

Deleting an Odoo product template **archives** its Shopify counterpart ( `status: ARCHIVED` ) rather than deleting it — order history in Shopify stays intact. The archive attempt is best-effort: a Shopify failure is logged but does not block the Odoo deletion.

### Two-way product sync from the list

**Sync Shopify Products** on the product list pushes each selected mapped product and then pulls it back, per connected store. Unmapped products are counted as skipped, and the notification breaks down pushed, pulled, skipped and failed counts.

## Variants, options & images

### How a Shopify variant is matched to an Odoo variant

Correct variant matching is the hardest part of any Shopify integration. The connector normalises both sides to a comparable key: option names and values are lowercased, sorted alphabetically and joined — for example `color=blue|size=xl`. The single-variant pseudo-option `Title / Default Title` that Shopify adds to products with no real options is filtered out, so such products normalise to an empty key.

Resolution is attempted in this order:

1. stored `shopify.product.variant` mapping by variant GID;
2. parent product mapping, then option-key match within that template;
3. on-demand product sync to build the missing variant structure, then retry the match once;
4. inventory-item GID mapping;
5. SKU match on `default_code` ;
6. product title match, creating a consumable template as a last resort.

### Self-healing stale mappings

When a stored mapping resolves to a variant whose option key does *not* match the Shopify line's `selectedOptions`, the connector treats the mapping as stale. It re-matches within the parent template, corrects the mapping in place and logs a warning naming both the old and new variant. This repairs the wrong-variant mappings that positional fallback can produce when SKUs are absent.

Successful fallback resolutions are written back as mappings, so the expensive chain runs once and later imports hit the direct lookup. A dedicated `reconcile_variant_mappings` routine can re-verify mappings in bulk.

### Variant lifecycle on export

- **Missing variants** present in Odoo but not Shopify are created, then have their inventory set.
- **Removed variants** present in Shopify but no longer in Odoo are deleted.
- **Prices** are pushed for all variants via `productVariantsBulkUpdate`, including attribute-value price extras.

### Images

Image upload uses Shopify's staged-upload flow: request a staged target, upload the bytes (multipart `POST`, or a raw `PUT` for signed targets), attach the media to the product, then poll until Shopify reports the media ready — up to 120 seconds. Replacing a product image deletes the previous media only after the new one is confirmed ready. Variant images are pushed separately, and image work can be skipped per-operation with `shopify_skip_image_sync` / `shopify_skip_image_pull` or globally with the **Skip product images in bulk sync** setting.

## Inventory & stock levels

### DIRECTION DECIDES THE WINNER

Inventory sync is destructive by design — whichever side is the source overwrites the other. **Pull** makes Shopify authoritative; **Push** makes Odoo authoritative. The scheduled cron defaults to **Pull**.

### Push: Odoo → Shopify

For every `(warehouse, Shopify location)` pair, the connector reads the Odoo available quantity from `stock.quant` and writes it to Shopify with `inventorySetQuantities`. Three details matter:

- **Compare before writing.** Current Shopify availability is read once per chunk and unchanged quantities are dropped, so a routine sync of a large catalogue sends almost no mutations.
- **Idempotency keys.** Every mutation carries a fresh idempotency key, and `changeFromQuantity` is supplied so Shopify can reject a write based on a stale read.
- **Deduplication.** Quantities are deduplicated per inventory item before sending, last write winning.

Bulk push deliberately skips per-item inventory *activation*: each activation call adds SSL round-trip latency and would breach the cron time limit on a large catalogue. Activation belongs to the product-push path, where it happens once per new variant.

### Cron safety guards

Inside a scheduled run, inventory push stops early once it exceeds **90 seconds** or **300 items** (both configurable). It logs what it processed; the remainder syncs on the next tick. This keeps the cron thread well inside Odoo's limit.

### Pull: Shopify → Odoo

Reads Shopify availability for every mapped variant that has an inventory-item GID and overwrites Odoo's on-hand quantity. This is the reverse of push and the default cron direction.

## What triggers an inventory push

| TRIGGER                         | BEHAVIOUR                                                                                                                                                               |
|---------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Validating a picking</b>     | After <code>button_validate</code> , affected products are pushed for the locations the moves touched — receipts, deliveries, adjustments and internal transfers alike. |
| <b>Editing a quant directly</b> | A change to <code>quantity</code> on an internal-location quant pushes that product.                                                                                    |
| <b>Product export</b>           | Step 3 of the export sequence sets quantities for the exported product.                                                                                                 |
| <b>Scheduled cron</b>           | Only when <b>Run inventory sync in scheduled cron</b> is enabled; direction follows the cron direction setting.                                                         |
| <b>Sync wizard</b>              | Explicit push or pull of inventory only.                                                                                                                                |

### RESERVATIONS ARE IGNORED ON PURPOSE

Changes to `reserved_quantity` never trigger a push. Reservation is internal Odoo mechanics — set when an order is confirmed or unreserved — and is not a real stock movement. Reacting to it would send Shopify the pre-deduction quantity, because the quant write fires mid-move before the move lines are fully applied.

For the same reason, picking validation sets `shopify_skip_quant_push` so the quant hook stays quiet and the picking pushes once, after the move is committed and the final quantity is readable.

## Section X

# Orders: Shopify → Odoo

## Selecting orders to import

Incremental runs filter on `updated_at:>=...` `status:any` , starting a configurable lookback window (default one hour) before the last successful order sync so orders modified around the boundary are not missed. A full resync uses `status:any` with no date filter. The first sync for an account is always a full resync.

`last_order_sync_date` advances to the newest `updatedAt` actually seen, not to "now", so an interrupted run cannot skip orders.

## What an imported order becomes

| ODOO FIELD                                                         | SHOPIFY SOURCE                               |
|--------------------------------------------------------------------|----------------------------------------------|
| <code>partner_id</code>                                            | Resolved or created customer                 |
| <code>partner_invoice_id</code> / <code>partner_shipping_id</code> | Billing / shipping address as child partners |
| <code>client_order_ref</code>                                      | Shopify order name, e.g. <code>#1149</code>  |
| <code>note</code>                                                  | Order note                                   |
| <code>date_order</code>                                            | <code>createdAt</code>                       |
| <code>warehouse_id</code> / <code>team_id</code>                   | From the account configuration               |
| <code>shopify_instance_id</code>                                   | The originating store                        |

Each line records its Shopify provenance on `sale.order.line` :

- `shopify_line_gid` , `shopify_variant_gid` ,
- `shopify_sku_snapshot` — the SKU as it was at import time,
- `shopify_instance_id` and `shopify_source_channel = 'shopify'` .

Line price comes from `originalUnitPriceSet.shopMoney.amount` . If Shopify sends no amount, the connector falls back to the Odoo list price plus attribute-value price extras.

## Auto-confirmation

An order arriving with financial status `PAID` is confirmed automatically. If confirmation fails because routes or procurement rules are not configured, the order is left in **Draft** and a warning is written to `shopify.log` naming the reason — the import itself still succeeds.

## Status tracking

The `shopify.order` mapping stores the Shopify order name, financial status, fulfillment status, fulfillment GID and whether the order came from a draft order. These surface read-only on the sale order as **Shopify Financial Status** and **Shopify Fulfillment Status**, and on the delivery picking as **Shopify Order** and its fulfillment status.

## Duplicate protection

Three layers stop a double import when a webhook and a cron run collide:

1. a PostgreSQL advisory lock per `(instance, order GID)` serialises concurrent imports of the same order;
2. `SELECT ... FOR UPDATE` inside a savepoint detects a concurrent update — if the row moved, the newer data already won and this attempt is skipped;
3. a unique constraint on the mapping catches the remaining race at `INSERT` time, handled gracefully as "skipped" rather than aborting the transaction.

An order whose `updatedAt` is not newer than the stored `sync_date` is skipped as unchanged — unless the Odoo sale order still needs cancelling, in which case it is reprocessed anyway. That exception exists so a missed cancellation webhook is repaired on the next sync.

### NOT IMPORTED

Order import maps **line items only**. Shopify shipping lines, per-line tax lines and discount allocations are *not* created as Odoo lines, so an imported order's total reflects line-item subtotals rather than the Shopify grand total. Taxes and discounts are applied by your Odoo fiscal position and pricelist configuration, and are carried into refund credit notes from the Odoo invoice. Plan for this if you reconcile Shopify payouts against Odoo order totals.

## Draft orders & quotations

### Shopify draft orders → Odoo quotations

Open Shopify draft orders ( `status:open` ) are imported as **draft** Odoo sale orders, with their mapping flagged `shopify_is_draft_order` . A computed order-type label distinguishes them from real orders in the list.

When a draft order is completed in Shopify it becomes a real order with a new GID and name. The connector *promotes* the existing mapping: it re-points it at the real order GID and updates the name from `D1234` to `#1149` , so no duplicate Odoo order appears.

### Odoo quotations → Shopify draft orders

A confirmed sale order is exported to Shopify as a draft order, provided:

- it is in **Sale** or **Locked** state;
- no confirmed (non-draft) Shopify mapping already exists;
- order sync is enabled globally;
- `shopify_instance_id` is set and that store is connected.

`shopify_instance_id` is set automatically on import; for an Odoo-originated order you choose the target store yourself. Export builds line items from the stored variant mappings, plus customer and address payloads derived from the Odoo partner. Export failures are logged as warnings and never block `action_confirm` .

**Sync to Shopify** is also available as a bulk action on the sale order list, and **Shopify Two-Way Order Sync** pushes all visible confirmed orders and then pulls from every connected store, reporting per-direction counts.

## Fulfillment & tracking

### Delivery validation creates the Shopify fulfillment

Validating an outgoing picking linked to a Shopify order triggers `fulfillmentCreateV2`. The connector first queries the order's fulfillment orders and fulfils only those in `OPEN` or `IN_PROGRESS` state; if none are open it logs and exits rather than erroring. The carrier tracking reference and carrier name are attached when present, and the returned fulfillment GID is stored on the mapping for later tracking updates.

Fulfillment is also pushed when a picking transitions to `done` through a write rather than the validate button. Mappings already marked `FULFILLED` are skipped, so no duplicate fulfillment is created.

### Tracking updates in both directions

| DIRECTION      | BEHAVIOUR                                                                                                                                                                                                                                          |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Odoo → Shopify | Setting or changing <code>carrier_tracking_ref</code> on an already-done outgoing picking calls <code>fulfillmentTrackingInfoUpdateV2</code> on the stored fulfillment GID. Customer notification is off by default.                               |
| Shopify → Odoo | An order update carrying tracking info writes the number to the done outgoing picking's <code>carrier_tracking_ref</code> , and matches the tracking company to an Odoo <code>delivery.carrier</code> by name when the picking has no carrier set. |

Each direction sets a context flag ( `shopify_skip_tracking_push` ) when writing, so an inbound tracking sync cannot bounce straight back out to Shopify.

Only the most recent non-cancelled fulfillment is considered when reading tracking from Shopify, and a write is skipped when the tracking number already matches.

## Refunds & cancellations

### Refunds become draft credit notes

A `refunds/create` webhook creates a **draft** credit note against the order's posted invoice — draft, so your finance team reviews before posting.

| SITUATION                                                            | RESULT                                                                                                                                 |
|----------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| Posted invoice exists, refund names line items                       | Credit note lines matched to the refunded invoice lines, preserving discount and tax configuration                                     |
| Posted invoice exists, no line items (shipping or adjustment refund) | A single amount line built from the refund transactions                                                                                |
| No posted invoice                                                    | A warning is logged for manual action; nothing is created                                                                              |
| Refund already processed                                             | Skipped — each credit note is tagged <code>shopify-refund-&lt;id&gt;</code> , making the handler idempotent against webhook redelivery |
| Order not mapped                                                     | Logged and ignored                                                                                                                     |

### Cancellations and voided orders

The linked Odoo sale order is cancelled when either condition holds:

- the Shopify order has a `cancelledAt` timestamp; or
- its financial status is `VOIDED` and its fulfillment status is `UNFULFILLED`.

The second rule catches voided payments that Shopify does not formally cancel — an order nobody paid for and nobody shipped should not stay open in Odoo. Sale orders already in **Cancelled** or **Done** are left alone, and a cancellation that Odoo refuses (for example because it is already invoiced) is logged as a warning instead of failing the sync.

Because the cancellation check also runs for orders that look unchanged, a cancellation missed while the connector was offline is repaired on the next sync.

## Customers & addresses

### Customer resolution order

Imported orders create real Odoo partners. To avoid duplicates, the connector resolves a customer in this order:

1. **Shopify customer GID** stored in the standard `res.partner.ref` field — exact and indexed;
2. **Email**, case-insensitive, restricted to non-company partners with no parent. When a match is found and `ref` is empty, the GID is back-filled so later syncs take the faster path;
3. **Create** a new partner from the Shopify name, email and phone.

A per-run cache keyed on GID (or email) prevents repeated lookups when one customer places several orders in the same batch. Where the installed localisation provides separate `first_name` / `last_name` fields, they are populated too.

### Guest orders

An order with neither a customer GID nor an email is attached to a shared **Shopify Customer** placeholder partner, created once per company. This keeps guest checkouts importable without generating a partner per anonymous order.

### Addresses

Billing and shipping addresses become **child partners** of type `invoice` and `delivery`. An existing child address matching on street, ZIP and city is reused rather than duplicated. Country is matched by ISO code, and state by name or code within that country. When Shopify sends no address, the parent partner is used for both roles.

## Scheduled automation

### The two scheduled actions

| SCHEDULED ACTION                             | INTERVAL   | WHAT IT DOES                                                                  |
|----------------------------------------------|------------|-------------------------------------------------------------------------------|
| <b>Shopify: Auto Sync Connected Accounts</b> | 30 minutes | Runs a full sync for every active, connected account                          |
| <b>Shopify: Retry Errored Products</b>       | 1 hour     | Re-attempts product mappings left in error state, each in a fresh transaction |

Both are standard `ir.cron` records — adjust or disable them in **Settings → Technical → Scheduled Actions**.

### What a scheduled sync does

For each connected account, honouring its per-store switches:

1. **Products** — incremental, or the next slice of an in-progress full resync;
2. **Orders** — incremental, or a full resync on the account's first sync, followed by draft orders;
3. **Inventory** — only when **Run inventory sync in scheduled cron** is enabled, in the configured direction.

Inventory is excluded from the scheduled run by default because it is the slowest phase; enable it once you have confirmed the run finishes comfortably.

### Resilience

- Each account syncs inside a savepoint and is retried up to **three times** on a concurrent-update conflict.
- Pending writes are flushed *inside* the savepoint, so a serialization failure is caught and retried where it happens rather than poisoning the transaction later.
- A failing account is rolled back safely, its error written to `shopify.log`, and the loop continues to the next account.
- Sync timestamps are flushed between phases through a raw `SAVEPOINT`, deliberately bypassing Odoo's flushing savepoint helper — which would pre-flush all pending writes and could poison the transaction before the error handler ran.
- GraphQL calls inside a cron run use a shorter timeout (5s connect, 15s read) than interactive calls (5s / 30s).

Order and product runs stop after one batch inside a cron or batch-mode run and resume from the persisted timestamp or cursor on the next tick, keeping every run inside Odoo's cron time limit.

## Manual sync controls

### The sync wizard

**Shopify → Operations → Sync Now** opens a direction-aware wizard — you choose the direction first, then the scope:

| FIELD                  | OPTIONS                                                            |
|------------------------|--------------------------------------------------------------------|
| <b>Shopify Account</b> | Any connected account                                              |
| <b>Direction</b>       | Pull from Shopify · Push to Shopify                                |
| <b>What to Sync</b>    | All · Products Only · Orders Only · Inventory Only                 |
| <b>Batch Mode</b>      | Process one batch and stop — run repeatedly to drain a large queue |

Orders cannot be pushed: they originate in Shopify. Selecting **Push** with **Orders Only** switches the scope back to **All**, and the combination is refused with a clear message if forced.

### Account buttons

| BUTTON                            | EFFECT                                                                                                 |
|-----------------------------------|--------------------------------------------------------------------------------------------------------|
| <b>Connect to Shopify</b>         | Start the OAuth flow (hidden once connected)                                                           |
| <b>Test Connection</b>            | Verify credentials and refresh the shop name                                                           |
| <b>Register Webhooks</b>          | Clear and re-register all webhook topics                                                               |
| <b>Sync Locations</b>             | Import Shopify locations as Odoo warehouses and map them                                               |
| <b>Push Warehouses to Shopify</b> | Create Shopify locations from unmapped Odoo warehouses                                                 |
| <b>Sync All</b>                   | Run products, orders and inventory now, honouring the per-store switches                               |
| <b>Full Product Resync</b>        | Queue a complete catalogue re-fetch for the next cron cycle (confirmation required)                    |
| <b>Clean Up Errored Products</b>  | Verify errored mappings against Shopify and archive genuinely deleted products (confirmation required) |

The form also carries statistic buttons for mapped **Products** and **Orders**, an **Error** ribbon when the account is in error state, and a per-row **Sync** button in the account list.

Record-level actions

| WHERE                   | ACTION                                                             |
|-------------------------|--------------------------------------------------------------------|
| Product form / list     | <b>Push to Shopify</b> — explicit store selection                  |
| Product form            | <b>Pull from Shopify</b> — re-import this product                  |
| Product list            | <b>Sync Shopify Products</b> — push then pull the selection        |
| Sale order list         | <b>Sync to Shopify</b> — export confirmed orders                   |
| Sale order list         | <b>Shopify Two-Way Order Sync</b> — push visible orders, then pull |
| Shopify product mapping | <b>Import from Shopify</b> — fetch by GID and reload the form      |

## Settings reference

**Settings → Shopify Connector** exposes the options below. All are stored as `ir.config_parameter` records under the `baseup_shopify.` prefix.

### Exposed in the settings UI

| SETTING                              | PARAMETER                              | DEFAULT           | EFFECT                                                          |
|--------------------------------------|----------------------------------------|-------------------|-----------------------------------------------------------------|
| Sync products by default             | <code>default_sync_products</code>     | On                | Default for new accounts                                        |
| Sync orders by default               | <code>default_sync_orders</code>       | On                | Default for new accounts; also gates outbound order export      |
| Sync inventory by default            | <code>default_sync_inventory</code>    | On                | Default for new accounts                                        |
| Run inventory sync in scheduled cron | <code>cron_sync_inventory</code>       | Off               | When off, the cron handles only products and orders             |
| Cron inventory direction             | <code>cron_inventory_direction</code>  | <code>pull</code> | <code>pull</code> = Shopify wins; <code>push</code> = Odoo wins |
| Auto-push newly created products     | <code>auto_push_new_products</code>    | Off               | Export new templates automatically (single-store only)          |
| Auto-update products on save         | <code>auto_push_product_updates</code> | Off               | Re-export linked products when watched fields change            |
| Sync batch size                      | <code>sync_batch_size</code>           | 100               | Shared batch size for product, order and inventory flows        |
| Skip product images in bulk sync     | <code>product_sync_skip_images</code>  | Off               | Large speed gain on big catalogues                              |
| Order sync lookback window           | <code>order_sync_lookback_hours</code> | 1                 | Hours of overlap before the last sync date                      |

## System parameters (no UI field)

| PARAMETER                                              | DEFAULT | EFFECT                                                                                                 |
|--------------------------------------------------------|---------|--------------------------------------------------------------------------------------------------------|
| <code>baseup_shopify.webhook_base_url</code>           | —       | Public HTTPS base URL for webhook and OAuth callbacks; takes precedence over <code>web.base.url</code> |
| <code>baseup_shopify.cron_inventory_max_seconds</code> | 90      | Time budget for inventory push inside a cron run                                                       |
| <code>baseup_shopify.cron_inventory_max_items</code>   | 300     | Item budget for inventory push inside a cron run                                                       |
| <code>baseup_shopify.order_sync_commit_every</code>    | 100     | Legacy order batch size; superseded by <code>sync_batch_size</code>                                    |
| <code>baseup_shopify.inventory_sync_chunk_size</code>  | 100     | Legacy inventory chunk size; superseded by <code>sync_batch_size</code>                                |

### WHY BOOLEANS ARE WRITTEN AS STRINGS

Odoo's default settings handler deletes a config parameter when its value is `False`. Combined with a field default of `True`, that would make an unchecked box silently re-check itself on the next form load. The connector writes boolean settings explicitly as `'True'` / `'False'` so *off* is stored as a real value and stays off.

## Logs & troubleshooting

### The Shopify log

**Shopify → Operations → Logs** lists `shopify.log` records, newest first. Each entry records the instance, type ( `Success` `Info` `Warning` `Error` ), the operation, the affected Odoo model and record, the Shopify ID or GID, and a message.

Log writing is itself defensive: if the transaction is already in a bad state the failure to log is caught and reported to the server log rather than masking the original error.

### Common situations

| SYMPTOM                                                        | CAUSE AND RESOLUTION                                                                                                                              |
|----------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| "Webhook URL must be a public HTTPS URL"                       | The effective base URL is not publicly reachable. Set <code>baseup_shopify.webhook_base_url</code> to your public HTTPS domain and retry.         |
| "No public base URL is configured"                             | Neither <code>baseup_shopify.webhook_base_url</code> nor <code>web.base.url</code> is set — OAuth cannot build a redirect URI.                    |
| "Shopify authentication failed" and the account flips to Error | The access token is invalid or revoked (HTTP 401). Reconnect through OAuth.                                                                       |
| "Shopify rate limit reached"                                   | HTTP 429. Reduce <b>Sync batch size</b> or lengthen the cron interval.                                                                            |
| Orders import but stay in Draft                                | Auto-confirm failed on routes or procurement rules; the reason is in the log. Fix the product's route configuration and confirm manually.         |
| Product mappings stuck in Error                                | Wait for the hourly retry cron, then use <b>Clean Up Errored Products</b> to archive any whose Shopify product was deleted.                       |
| Inventory sync never finishes in the cron                      | It is stopping at the 90-second / 300-item guard by design and resuming next tick. Raise the guards, or run inventory from the wizard instead.    |
| A new product did not auto-push                                | Either auto-push is off, or more than one store is connected — the multi-store guard skips it. Use <b>Push to Shopify</b> .                       |
| An order line resolved to the wrong variant                    | A stale mapping from positional fallback. The connector detects and corrects this on the next import, logging a warning with both variants.       |
| "Shopify SSL verification failed"                              | The server's CA bundle cannot verify Shopify. The connector pins <code>requests</code> 'CA bundle explicitly; check the host's certificate store. |

### Concurrency, in one place

Much of the module's complexity exists to survive webhooks and crons touching the same rows simultaneously under `REPEATABLE READ`. The techniques used:

| TECHNIQUE                               | WHERE                                                                                       |
|-----------------------------------------|---------------------------------------------------------------------------------------------|
| Advisory transaction locks              | Per order GID on import; per product on export                                              |
| FOR UPDATE SKIP LOCKED                  | Webhook marker writes — back off instead of blocking                                        |
| Raw SAVEPOINT instead of the ORM helper | Timestamp flushes, order locks, webhook markers                                             |
| Full rollback between retries           | Webhook dispatch — a savepoint rollback keeps the stale snapshot                            |
| Bounded retries with backoff            | Webhooks (4 attempts), cron account sync (3 attempts), product template writes (6 attempts) |
| Unique constraints as a final guard     | Order and product mappings, caught as "skipped"                                             |
| No-op write detection                   | Mapping updates skip writes when nothing changed                                            |

## Section XIX

# Guided tours

The module ships four interactive walkthroughs that drive the real UI. Start one from **Settings** → **Technical** → **Tours**, or let the onboarding tour run for a new user. The same definitions are reused by the automated test suite, so the tours stay accurate as the UI changes.

| TOUR                          | COVERS                                                                                                                          |
|-------------------------------|---------------------------------------------------------------------------------------------------------------------------------|
| shopify_onboarding            | Creating the account, entering credentials, choosing a warehouse, connecting via OAuth, registering webhooks, syncing locations |
| shopify_first_product_push    | Pushing a first product to Shopify                                                                                              |
| shopify_multi_warehouse_setup | Mapping several Shopify locations to Odoo warehouses                                                                            |
| shopify_first_order_review    | Reviewing an imported order and its Shopify status fields                                                                       |

# Access & security

## Group permissions

| MODEL GROUP                                    | SALES MANAGER               | INTERNAL USER                              |
|------------------------------------------------|-----------------------------|--------------------------------------------|
| Accounts, instances, mappings, locations, logs | Read, write, create, delete | Read only                                  |
| Sync wizard, push wizard                       | Full                        | Read, write, create <span>transient</span> |

The **Shopify** menu itself is restricted to `sales_team.group_sale_manager` . Internal users keep read access so Shopify status fields remain visible on products, orders and pickings they already work with.

## Credential handling

- Access token, API secret, webhook secret, client ID and OAuth state are limited to `base.group_system` .
- Secret inputs are rendered as password fields.
- The OAuth `state` token is 32 bytes of `secrets.token_urlsafe` entropy, matched on callback and cleared immediately after use.
- Webhook bodies are authenticated by HMAC-SHA256 with a constant-time comparison; an account with no webhook secret rejects all webhooks.
- The webhook route is public by necessity (Shopify cannot authenticate) but processes nothing before the signature check passes.

## Limits & licensing

### Known limits

- **Order totals.** Shipping lines, tax lines and discount allocations are not imported as order lines (see Section X).
- **Orders are inbound-only as orders.** Odoo quotations export as Shopify *draft* orders, never as completed orders.
- **Inventory sync overwrites.** There is no merge strategy — the configured direction wins outright.
- **Auto-push of new products is single-store.** With several stores connected, use the push wizard.
- **Multi-currency.** Line prices are read from `shopMoney`, the shop's own currency; presentment currencies are not converted.
- **Refunds require a posted invoice.** Without one, the refund is logged for manual handling.

### Requirements

| REQUIREMENT       | DETAIL                                                                                                                                                                     |
|-------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Odoo              | 19.0, with <code>sale_management</code> , <code>website_sale</code> , <code>stock</code> , <code>product</code> , <code>account</code> and <code>delivery</code> installed |
| Shopify           | A store plus a custom app from the Shopify Dev Dashboard with the scopes in Section III                                                                                    |
| Networking        | A public HTTPS URL for OAuth redirects and webhook delivery; outbound HTTPS to Shopify                                                                                     |
| Python            | <code>requests</code> (already an Odoo dependency)                                                                                                                         |
| Scheduled actions | Odoo cron enabled for automatic syncing                                                                                                                                    |

### Licensing & support

| ITEM    | DETAIL                                          |
|---------|-------------------------------------------------|
| Module  | <code>baseup_shopify</code> — Shopify Connector |
| Version | 19.0.2.0.0                                      |
| License | Odoo Proprietary License v1.0 (OPL-1)           |
| Price   | USD 450.00                                      |
| Author  | Baseup Labs — baseuplabs.com                    |
| Support | support@baseuplabs.com                          |

