

Developer Setup Guide

How to go from a purchased `baseup_shopify` package to a connected, syncing Shopify store — where the module goes on disk, how to load it, how to make Odoo reachable for OAuth and webhooks, and how to run the test suite.

MODULE baseup_shopify	VERSION 19.0.2.0.0	ODOO 19.0
SHOPIFY API 2026-04	EXTRA PIP DEPS None	TIME NEEDED ~30 min

Contents

I	Before you start
II	Get the module
III	Where the module goes
IV	Install it in Odoo
V	Make Odoo publicly reachable
VI	Create the Shopify app
VII	Connect & first sync
VIII	Verify the install
IX	Development workflow
X	Running the tests
XI	Upgrade & uninstall
XII	Setup troubleshooting

Section I

Before you start

Check these before you touch the filesystem — two of them rule out an install entirely.

REQUIREMENT	DETAIL
Odoo 19.0	The module version is <code>19.0.2.0.0</code> . It will not load on 17 or 18 — the manifest, view syntax and ORM calls target 19.
A hosting model that allows custom modules	Self-hosted (source, package or Docker) or Odoo.sh. See the warning below.
Standard apps available	<code>sale_management</code> , <code>website_sale</code> , <code>stock</code> , <code>product</code> , <code>account</code> , <code>delivery</code> — all ship with Odoo and are pulled in automatically as dependencies.
A public HTTPS URL	Required for the OAuth redirect and webhook delivery. Section V covers the options, including tunnels for local development.
Shopify store + Dev Dashboard access	You need to create a custom app to obtain a Client ID and Secret.
Python packages	None to install. The module imports only the standard library plus <code>requests</code> and <code>psycpg2</code> , both already Odoo dependencies.
Shell access to the Odoo host	To place files and restart the service.

ODOO ONLINE CANNOT RUN THIS MODULE

Odoo Online (the `*.odoo.com` SaaS tier) does not permit custom or third-party code. If that is your current hosting, you must move to **Odoo.sh** or a **self-hosted** deployment before installing. There is no workaround.

Get the module

From the Odoo App Store

1 Purchase the module

Buy **Shopify Connector** (USD 450.00, OPL-1) from the Odoo Apps store with the Odoo account that will own the licence.

2 Download the package

Open **My Apps / Purchases** in your Odoo account and download the module for **19.0**. You get a `.zip`.

3 Unzip it

The archive contains a single top-level folder named `baseup_shopify`. That folder is the module.

THE FOLDER NAME IS THE MODULE NAME

Odoo identifies a module by its directory name. The folder must stay exactly `baseup_shopify` — no version suffix, no `-main`, no rename. A folder called `baseup_shopify-19.0` simply will not be found.

From source control

Licensed customers and Baseup developers who have been granted access to the private source repository can clone it instead of unzipping the App Store package. Contact support@baseuplabs.com for access; the repository location is provided with it and is not published here.

Whichever way you obtain it, the layout is the same: a checkout keeps `baseup_shopify/` as a direct child of its root, so you can point `addons_path` at that root and Odoo will find the module. That is the most convenient layout for development, because pulling updates the module in place.

What is inside

PATH	CONTENTS
<code>__manifest__.py</code>	Module metadata, dependencies, data files, assets
<code>models/</code>	Mapping models, sync services, GraphQL transport and queries
<code>controllers/</code>	Webhook endpoints and the OAuth callback
<code>wizard/</code>	Sync wizard and push wizard
<code>views/</code>	Form, list and settings views; the Shopify menu
<code>data/</code>	Two <code>ir.cron</code> records and four tour registrations
<code>security/</code>	<code>ir.model.access.csv</code> access rules
<code>static/src/tours/</code>	Onboarding tours loaded into the backend bundle
<code>static/description/</code>	App Store listing page and images
<code>tests/</code>	Python unit tests and a JS tour test
<code>ARCHITECTURE.md</code>	Layering and design rules for contributors

Section III

Where the module goes

Place the `baseup_shopify` folder in a directory that is listed in Odoo's `addons_path` . Pick the section matching your deployment.

Self-hosted from source

A conventional layout keeps third-party modules out of the Odoo checkout:

SHELL

```
sudo mkdir -p /opt/odoo/custom-addons
sudo cp -r ~/Downloads/baseup_shopify /opt/odoo/custom-addons/
sudo chown -R odoo:odoo /opt/odoo/custom-addons/baseup_shopify
sudo find /opt/odoo/custom-addons/baseup_shopify -type d -exec chmod 755 {} \;
sudo find /opt/odoo/custom-addons/baseup_shopify -type f -exec chmod 644 {} \;
```

Then add that directory to `addons_path` in your config file:

/ETC/ODOO/ODOO.CONF

```
[options]
addons_path = /opt/odoo/odoo/addons,/opt/odoo/custom-addons
; the connector needs a cron worker for scheduled sync
max_cron_threads = 2
; large catalogue imports can outlast the default 120s
limit_time_real = 600
limit_time_real_cron = 900
```

OWNERSHIP MATTERS

The files must be readable by the system user that runs Odoo (usually `odoo`). A module copied as `root` with restrictive permissions is a common cause of "module not found" — Odoo cannot read the directory to scan it.

Docker

Mount the module into the image's addons directory, or mount a folder of custom addons:

DOCKER-COMPOSE.YML

```
services:
  odoo:
    image: odoo:19
    depends_on: [db]
    ports: ["8069:8069"]
    volumes:
      - ./custom-addons:/mnt/extra-addons # contains baseup_shopify/
      - ./odoo.conf:/etc/odoo/odoo.conf
      - odoo-web-data:/var/lib/odoo
    environment:
      - HOST=db
      - USER=odoo
      - PASSWORD=odoo
```

The official image already includes `/mnt/extra-addons` in its `addons_path`, so placing `baseup_shopify/` inside `./custom-addons` is enough. Restart the container after adding it:

SHELL

```
docker compose restart odoo
```

Odoo.sh

Odoo.sh builds from your Git repository — there is no filesystem to copy into. Commit the module to the branch you deploy:

1 Add the module to your Odoo.sh repo

Commit the `baseup_shopify/` folder at the repository root (or inside a directory already on the addons path).

2 Push to the branch

Push to your **development** or **staging** branch first. Odoo.sh triggers a build automatically.

3 Install on the build

Once the build is green, install the module from the Apps menu, or add it to the branch's install list.

4 Promote to production

Merge to **production** only after the staging build installs cleanly and a test sync works.

Odoo.sh gives you a public HTTPS hostname out of the box, which satisfies Section V with no extra work.

Install it in Odoo

Option A — from the Apps menu

1 Restart Odoo

A new directory on the addons path is only picked up at startup.

SHELL

```
sudo systemctl restart odoo
```

2 Enable developer mode

Settings → **General Settings**, scroll to **Developer Tools** and click **Activate the developer mode**. Without it, the Apps menu hides the list-update action.

3 Update the apps list

Go to **Apps** and click **Update Apps List**, then confirm. This rescans the addons path.

4 Install

Clear the default *Apps* filter, search **Shopify Connector**, and click **Activate**. Odoo installs the six dependencies first if they are not present.

Option B — from the command line

Faster and scriptable; the standard choice for a fresh environment or CI:

SHELL

```
# install into an existing database
odoo -c /etc/odoo/odoo.conf -d mydb -i baseup_shopify --stop-after-init

# upgrade after changing code or pulling a new version
odoo -c /etc/odoo/odoo.conf -d mydb -u baseup_shopify --stop-after-init
```

WEBSITE_SALE PULLS IN THE WEBSITE STACK

`website_sale` is a hard dependency, so installing the connector also installs **Website** and **eCommerce**. On a database that has never had them, expect a noticeably longer install and new website menus. This is expected, not a fault — plan the install window accordingly.

What installation creates

- A top-level **Shopify** app menu, visible to Sales Managers.
- Two scheduled actions: **Auto Sync Connected Accounts** (every 30 minutes) and **Retry Errored Products** (hourly), both active immediately.
- A **Shopify Connector** block in **Settings**.
- Four onboarding tours registered under **Settings** → **Technical** → **Tours**.

- Access rules granting Sales Managers full control and internal users read-only visibility.

THE CRONS START EMPTY-HANDED

Both scheduled actions only act on accounts whose state is *connected*. Until you finish Section VII they run and do nothing, which is harmless.

Section V

Make Odoo publicly reachable

Two features need Shopify to reach your Odoo over public HTTPS: the **OAuth redirect** and **webhook delivery**. Webhook registration actively refuses to run against `http://`, `localhost` or a loopback address, so this is not optional.

Set the base URL parameter

Go to **Settings → Technical → Parameters → System Parameters** and create:

KEY	VALUE
<code>baseup_shopify.webhook_base_url</code>	<code>https://odoo.example.com</code> — no trailing slash

The connector prefers this parameter over `web.base.url`. That split is deliberate: your development server can keep `web.base.url` pointed at a LAN address while webhooks and OAuth use the public tunnel.

Production — reverse proxy

Terminate TLS at nginx or another proxy in front of Odoo and forward to port 8069. The proxy must preserve the `X-Shopify-Hmac-Sha256` header and the **raw, unmodified request body** — the signature is computed over the exact bytes, so any rewriting or re-encoding breaks verification. Set `proxy_mode = True` in `odoo.conf`.

Local development — a tunnel

For a laptop install, expose port 8069 with a tunnelling service and use the HTTPS hostname it hands you:

SHELL
<pre>ngrok http 8069 # then set baseup_shopify.webhook_base_url to the https://... forwarding URL</pre>

Remember that a free tunnel hostname changes on every restart. When it changes you must update the system parameter, update the redirect URI in the Shopify app, and click **Register Webhooks** again — the previously registered callbacks now point at a dead host.

Create the Shopify app

The connector authenticates as a custom app you own. Create it in the Shopify Dev Dashboard for the store you are integrating.

- 1

Create a custom app

In the Shopify Dev Dashboard, create an app for your store.

- 2

Add the redirect URI

Register this exact callback URL, built from the base URL you set in Section V:

REDIRECT URI

https://odoo.example.com/shopify/oauth/callback

Shopify rejects the authorisation request if the redirect URI does not match a registered one character for character.

- 3

Grant the required scopes

The connector requests all of the following; the app must be allowed to grant them.

- 4

Copy the Client ID and Client Secret

From the app's settings. You will paste both into Odoo in the next section.

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

LEGACY ACCESS TOKENS STILL WORK

If you already have an Admin API access token from an older custom app, you can paste it straight into the **Access Token** field and skip OAuth. You must then also set the **Webhook Secret** by hand, because nothing fills it in for you — and without it every inbound webhook is rejected.

Connect & first sync

Do these in order. The sequence matters: locations must be mapped before inventory can sync, and products must exist before orders can resolve their lines cleanly.

1 Create the account record

Shopify → Configuration → Shopify Accounts → New. Fill in a name, the **Shop URL** (`mystore.myshopify.com`), the **Client ID** and **Client Secret**, then pick the **Company** and default **Warehouse**. Save.

2 Connect to Shopify

Click **Connect to Shopify**. You are redirected to Shopify to approve the scopes and returned to the account form. Status should now read **Connected**, with the Access Token and Webhook Secret filled in automatically.

3 Test the connection

Click **Test Connection**. A success notification naming your shop confirms the token works. The shop name is refreshed from Shopify at this point.

4 Register webhooks

Click **Register Webhooks**. This clears every existing webhook on the store and registers all 13 topics against your base URL. Expect a "13 webhook(s) registered" notification.

5 Map locations to warehouses

Click **Sync Locations** to pull Shopify's active locations in as Odoo warehouses and create the mapping rows. If Odoo has warehouses that Shopify does not know about, follow up with **Push Warehouses to Shopify**. Confirm the **Location Mappings** table is populated before continuing.

6 Run the first sync

Click **Sync All**. Because no sync has run yet, this performs a full product walk and a full order resync. On a large catalogue it will take several cron cycles to finish — that is by design; the ID cursor resumes where each run stops.

7 Choose your inventory direction

In **Settings → Shopify Connector**, decide whether the scheduled cron should sync inventory at all, and in which direction. **Pull** makes Shopify authoritative; **Push** makes Odoo authoritative. It is off by default because it is the slowest phase.

DECIDE THE INVENTORY DIRECTION BEFORE ENABLING IT

Inventory sync overwrites — there is no merge. Enabling the wrong direction on a live store will overwrite real stock levels on one side. Confirm which system is your source of truth first, and test on a staging database.

Section VIII

Verify the install

Work down this list; each item exercises a different part of the integration.

- ✓ The **Shopify** app appears in the main menu with **Operations** and **Configuration** submenus.
- ✓ The account's status badge reads **Connected** and **Test Connection** succeeds.
- ✓ **Settings** → **Technical** → **Scheduled Actions** lists both Shopify crons as active.
- ✓ The account's **Location Mappings** table has at least one row with a Shopify location GID.
- ✓ The account form's **Products** and **Orders** statistic buttons show non-zero counts after the first sync.
- ✓ **Shopify** → **Operations** → **Logs** contains entries and no unexplained errors.
- ✓ Placing a test order in Shopify creates a sale order in Odoo within seconds — this proves webhooks are being delivered *and* verified.
- ✓ Validating that order's delivery marks the Shopify order fulfilled.
- ✓ The last-sync timestamps on the account form are advancing after a cron tick.

THE WEBHOOK TEST IS THE IMPORTANT ONE

A successful **Register Webhooks** only proves Shopify accepted your URL. Only a real inbound event proves the URL is reachable, the HMAC secret matches and the proxy is passing the raw body through. Always place a test order.

Development workflow

Reloading changes

WHAT CHANGED	WHAT TO DO
Python code	Restart Odoo. With <code>--dev=reload</code> the server restarts itself on save.
XML views, data, security	Upgrade the module: <code>-u baseup_shopify</code> . With <code>--dev=xml</code> views are re-read from file on each request, so no upgrade is needed while iterating.
JS tours / assets	Hard-refresh the browser. With <code>--dev=assets</code> bundles are rebuilt per request.
Manifest (new data file, new dependency)	Upgrade the module; a new dependency may require a restart first.

SHELL – DEVELOPMENT SERVER

```
odoo -c /etc/odoo/odoo.conf -d mydb --dev=all
```

Watching the connector's logs

Every module file logs under the `odoo.addons.baseup_shopify` namespace, so you can raise its verbosity without drowning in unrelated Odoo output:

SHELL

```
odoo -c /etc/odoo/odoo.conf -d mydb \  
--log-handler odoo.addons.baseup_shopify:DEBUG
```

At `DEBUG` you additionally get GraphQL pagination progress, variant resolution decisions, echo-suppression skips and per-order import detail.

Useful context flags

The sync services honour context keys that suppress specific behaviour. They are invaluable when scripting a migration from the Odoo shell, because they stop an import from bouncing straight back out to Shopify.

CONTEXT KEY	EFFECT
<code>shopify_skip_auto_push</code>	Suppress outbound push on product create/write
<code>shopify_skip_order_export</code>	Suppress outbound order export on confirm
<code>shopify_skip_image_sync</code> / <code>shopify_skip_image_pull</code>	Skip image push / pull for this operation
<code>shopify_skip_inventory_pull</code>	Skip pulling inventory during a product import
<code>shopify_skip_quant_push</code>	Stop the <code>stock.quant</code> hook from pushing
<code>shopify_skip_tracking_push</code>	Mark a tracking write as inbound, preventing an echo
<code>shopify_bulk_product_sync</code>	Signal a bulk run so per-record side effects are skipped
<code>shopify_batch_mode</code>	Stop after one batch
<code>shopify_cron_run</code>	Apply cron timeouts and time/item guards

ODOO SHELL – EXAMPLE

```
instance = env['shopify.instance'].search([], limit=1)
env['shopify.product.sync'].with_context(
    shopify_skip_auto_push=True,
    shopify_skip_image_pull=True,
).import_all_products(instance)
```

Extending the module

Follow the layering described in `ARCHITECTURE.md` and in Section II of the Feature Handbook: never edit Odoo core addons, add behaviour through `_inherit`, keep mapping models separate from sync services, and keep transport concerns inside `shopify.api`. Adding a GraphQL call means adding a query constant and calling `env['shopify.api'].graphql(instance, QUERY, vars)` — not issuing your own HTTP request.

Section X

Running the tests

The module ships Python unit tests and one JS tour test. Odoo discovers everything in `tests/` automatically when tests are enabled.

Test tags

TAG	COVERS
<code>baseup_shopify</code>	Every test in the module
<code>shopify_order_sync</code>	Order sync unit tests — draft-order promotion, unchanged-order skipping
<code>shopify_tour</code>	The sync wizard JS tour (tagged <code>post_install</code> , <code>-at_install</code>)

Commands

SHELL

```
# everything in the module, on a fresh database
odoo -c /etc/odoo/odoo.conf -d test_db -i baseup_shopify \
  --test-enable --test-tags baseup_shopify --stop-after-init

# just the order sync unit tests, against an existing database
odoo -c /etc/odoo/odoo.conf -d test_db -u baseup_shopify \
  --test-enable --test-tags shopify_order_sync --stop-after-init

# just the JS tour
odoo -c /etc/odoo/odoo.conf -d test_db -u baseup_shopify \
  --test-enable --test-tags shopify_tour --stop-after-init
```

THE TOUR TEST NEEDS A BROWSER

`shopify_tour` is an `HttpCase` that drives the real UI, so the Odoo host needs Chrome or Chromium on its `PATH` . Without it the test is skipped or fails on browser startup rather than on a genuine regression.

Writing tests against Shopify

`tests/common.py` provides `ShopifyTestCase` , which creates a connected account and matching instance in `setUpClass` , plus a `mock_graphql()` context manager that swaps the transport for a FIFO queue of canned responses. Subclass it rather than rebuilding the fixtures:

TESTS/TEST_EXAMPLE.PY

```
from odoo.tests import tagged
from .common import ShopifyTestCase

@tagged('baseup_shopify', 'shopify_example')
class TestExample(ShopifyTestCase):

    def test_something(self):
        with self.mock_graphql([{'data': {'products': {'edges': [], 'pageInfo': {}}}}]):
            self.env['shopify.product.sync'].import_all_products(self.instance)
```

`mock_graphql` raises an assertion if the code under test makes more calls than you queued responses for, so a missing fixture fails loudly instead of silently returning `None`. No test ever contacts Shopify.

Running the onboarding tours by hand

The four user-facing tours are useful for smoke-testing a fresh install. Start one from **Settings** → **Technical** → **Tours**: `shopify_onboarding`, `shopify_first_product_push`, `shopify_multi_warehouse_setup` or `shopify_first_order_review`.

Upgrade & uninstall

Upgrading to a new release

1 Back up

Take a database and filestore backup. Always test on staging first.

2 Replace the folder

Swap in the new `baseup_shopify` directory, or `git pull` if you installed from the repository.

3 Restart and upgrade

SHELL

```
sudo systemctl restart odoo
odoo -c /etc/odoo/odoo.conf -d mydb -u baseup_shopify --stop-after-init
```

4 Re-register webhooks if the base URL changed

Registered callbacks embed the base URL, so a host change requires clicking **Register Webhooks** again.

5 Re-run the verification checklist

Section VIII, in particular the test-order step.

Uninstalling

Uninstall from the Apps list. Odoo drops the module's models, which deletes every mapping record — `shopify.product`, `shopify.product.variant`, `shopify.order`, `shopify.log` and the location mappings.

UNINSTALLING IS DESTRUCTIVE AND ASYMMETRIC

Products, sale orders and partners created by the connector **stay** in Odoo, but the links back to Shopify are gone. Reinstalling will not restore them: the connector has to rediscover each product, either from the `odoo-product-tmpl-<id>` tag it wrote on Shopify at creation time, or by a fresh import. Webhooks registered on the Shopify side are also left behind — delete them in Shopify, or re-register after reinstalling.

Setup troubleshooting

SYMPTOM	CAUSE AND FIX
Module does not appear after Update Apps List	The folder is not on the addons path, is misnamed, or is unreadable by the Odoo user. Confirm the directory is exactly <code>baseup_shopify</code> , that its parent is in <code>addons_path</code> , and that Odoo was restarted after the change. Check ownership per Section III.
No Update Apps List button	Developer mode is off. Enable it in Settings → Developer Tools .
Install fails on a missing dependency	One of the six required apps is unavailable in your Odoo edition or addons path. Verify <code>website_sale</code> and <code>delivery</code> in particular are present.
"No public base URL is configured"	Neither <code>baseup_shopify.webhook_base_url</code> nor <code>web.base.url</code> is set, so no OAuth redirect URI can be built. Set the parameter per Section V.
"Webhook URL must be a public HTTPS URL"	The effective base URL is <code>http://</code> , <code>localhost</code> or a loopback address. Use a real HTTPS hostname or a tunnel.
OAuth returns to Odoo but the account stays Disconnected	The token exchange failed — usually a wrong Client Secret or a redirect URI that does not match the one registered in the Shopify app. The server log records the exchange response.
"Invalid or expired OAuth state"	The stored state no longer matches — the flow was restarted, or another connect attempt overwrote it. Click Connect to Shopify again and complete it in one pass.
Webhooks register but nothing arrives	Shopify cannot reach the URL. Verify it from outside your network, and check the proxy forwards to Odoo.
Webhooks arrive but are rejected with 401	The HMAC check failed: the Webhook Secret does not match the app's Client Secret, or a proxy is altering the request body. The signature is computed over the raw bytes.
Scheduled sync never runs	<code>max_cron_threads</code> is 0, or the crons are inactive. Confirm both in <code>odoo.conf</code> and under Scheduled Actions .
Cron sync starts then stops mid-catalogue	Working as designed — the run hit a batch or time guard and persisted its cursor. It resumes next tick. Raise <code>limit_time_real_cron</code> and the inventory guards if you want longer runs.
First sync seems stuck on products	A full ID-cursor resync spans multiple cron cycles on a large catalogue. Watch <code>full_resync_cursor</code> on the account advance; enable Skip product images in bulk sync to speed it up substantially.
Inventory sync does nothing	No location mappings exist. Run Sync Locations — without a warehouse-to-location pair there is nowhere to write quantities.

For behaviour questions rather than setup problems — what a given sync actually does, what each setting changes, the full webhook topic list — see the Feature Handbook, in particular Section XVIII, Logs & troubleshooting.

