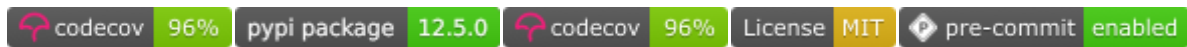

Shopify API



The Shopify Admin API Python Library

Usage

Requirements

You should be signed up as a partner on the Shopify Partners Dashboard so that you can create and manage shopify applications.

Installation

To easily install or upgrade to the latest release, use pip.

```
1 pip install --upgrade ShopifyAPI
```

Table of Contents

- Getting Started
 - Public and Custom Apps
 - Private Apps
- Billing
- Session Tokens
- Handling Access Scope Operations
- Advanced Usage
- Prefix Options
- Console
- GraphQL
- Using Development Version
- Relative Cursor Pagination
- Limitations
- Additional Resources

Getting Started

Public and Custom Apps

1. First create a new application in the Partners Dashboard, and retrieve your API Key and API Secret Key.
2. We then need to supply these keys to the Shopify Session Class so that it knows how to authenticate.

```
1 import shopify
2
3 shopify.Session.setup(api_key=API_KEY, secret=API_SECRET)
```

3. In order to access a shop's data, apps need an access token from that specific shop. We need to authenticate with that shop using OAuth, which we can start in the following way:

```
1 shop_url = "SHOP_NAME.myshopify.com"
2 api_version = '2024-01'
3 state = binascii.b2a_hex(os.urandom(15)).decode("utf-8")
4 redirect_uri = "http://myapp.com/auth/shopify/callback"
5 scopes = ['read_products', 'read_orders']
6
7 newSession = shopify.Session(shop_url, api_version)
8 auth_url = newSession.create_permission_url(scopes, redirect_uri,
9 state)
9 # redirect to auth_url
```

4. Once the merchant accepts, the shop redirects the owner to the `redirect_uri` of your application with a parameter named 'code'. This is a temporary token that the app can exchange for a permanent access token. You should compare the state you provided above with the one you received back to ensure the request is correct. Now we can exchange the code for an `access_token` when you get the request from shopify in your callback handler:

```
1 session = shopify.Session(shop_url, api_version)
2 access_token = session.request_token(request_params) #
    request_token will validate hmac and timing attacks
3 # you should save the access token now for future use.
```

5. Now you're ready to make authorized API requests to your shop!:

```
1 session = shopify.Session(shop_url, api_version, access_token)
2 shopify.ShopifyResource.activate_session(session)
3
4 shop = shopify.Shop.current() # Get the current shop
5 product = shopify.Product.find(179761209) # Get a specific product
6
7 # execute a graphql call
```

```
8 shopify.GraphQL().execute("{ shop { name id } }")
```

Alternatively, you can use `temp` to initialize a Session and execute a command:

```
1 with shopify.Session.temp(shop_url, api_version, token):
2     product = shopify.Product.find()
```

6. It is best practice to clear your session when you're done. A temporary session does this automatically:

```
1 shopify.ShopifyResource.clear_session()
```

Private Apps Private apps are a bit quicker to use because OAuth is not needed. You can create the private app in the Shopify Merchant Admin. You can use the Private App password as your `access_token`:

With full session

```
1 session = shopify.Session(shop_url, api_version, private_app_password)
2 shopify.ShopifyResource.activate_session(session)
3 # ...
4 shopify.ShopifyResource.clear_session()
```

With temporary session

```
1 with shopify.Session.temp(shop_url, api_version, private_app_password):
2     shopify.GraphQL().execute("{ shop { name id } }")
```

Billing

Note: Your application must be public to test the billing process. To test on a development store use the `'test': True` flag

1. Create charge after session has been activated `python application_charge = shopify.ApplicationCharge.create({ 'name': 'My public app', 'price': 123, 'test': True, 'return_url': 'https://domain.com/approve'})` # Redirect user to `application_charge.confirmation_url` so they can approve the charge
2. After approving the charge, the user is redirected to `return_url` with `charge_id` parameter (Note: This action is no longer necessary if the charge is created with API version 2021-01 or later)
`python charge = shopify.ApplicationCharge.find(charge_id)`
`shopify.ApplicationCharge.activate(charge)`

-
3. Check that `activated_charge` status is `active` python `activated_charge = shopify.ApplicationCharge.find(charge_id)has_been_billed = activated_charge.status == 'active'`

Advanced Usage

It is recommended to have at least a basic grasp on the principles of the `pyactiveresource` library, which is a port of rails/ActiveResource to Python and upon which this package relies heavily.

Instances of `pyactiveresource` resources map to RESTful resources in the Shopify API.

`pyactiveresource` exposes life cycle methods for creating, finding, updating, and deleting resources which are equivalent to the `POST`, `GET`, `PUT`, and `DELETE` HTTP verbs.

```
1 product = shopify.Product()
2 product.title = "Shopify Logo T-Shirt"
3 product.id # => 292082188312
4 product.save() # => True
5 shopify.Product.exists(product.id) # => True
6 product = shopify.Product.find(292082188312)
7 # Resource holding our newly created Product object
8 # Inspect attributes with product.attributes
9 product.price = 19.99
10 product.save() # => True
11 product.destroy()
12 # Delete the resource from the remote server (i.e. Shopify)
```

Here is another example to retrieve a list of open orders using certain parameters:

```
1 new_orders = shopify.Order.find(status="open", limit="50")
```

Prefix options

Some resources such as `Fulfillment` are prefixed by a parent resource in the Shopify API (e.g. `orders/450789469/fulfillments/255858046`). In order to interact with these resources, you must specify the identifier of the parent resource in your request.

```
1 shopify.Fulfillment.find(255858046, order_id=450789469)
```

Console

This package also includes the `shopify_api.py` script to make it easy to open an interactive console to use the API with a shop. 1. Obtain a private API key and password to use with your shop (step

2 in “Getting Started”) 1. Save your default credentials: `shopify_api.py add yourshopname`
1. Start the console for the connection: `shopify_api.py console` 1. To see the full list of commands, type: `shopify_api.py help`

GraphQL

This library also supports Shopify’s new GraphQL API. The authentication process is identical. Once your session is activated, simply construct a new graphql client and use `execute` to execute the query.

```
1 result = shopify.GraphQL().execute('{ shop { name id } }')
```

You can perform more complex operations using the `variables` and `operation_name` parameters of `execute`.

For example, this GraphQL document uses a fragment to construct two named queries - one for a single order, and one for multiple orders:

```
1 # ./order_queries.graphql
2
3 fragment OrderInfo on Order {
4     id
5     name
6     createdAt
7 }
8
9 query GetOneOrder($order_id: ID!){
10     node(id: $order_id){
11         ...OrderInfo
12     }
13 }
14
15 query GetManyOrders($order_ids: [ID]!){
16     nodes(ids: $order_ids){
17         ...OrderInfo
18     }
19 }
```

Now you can choose which operation to execute:

```
1 # Load the document with both queries
2 document = Path("./order_queries.graphql").read_text()
3
4 # Specify the named operation to execute, and the parameters for the
  query
5 result = shopify.GraphQL().execute(
6     query=document,
```

```
7     variables={"order_id": "gid://shopify/Order/12345"},
8     operation_name="GetOneOrder",
9 )
```

Using Development Version

Building and installing dev version

```
1 python setup.py sdist
2 pip install --upgrade dist/ShopifyAPI-*.tar.gz
```

Note Use the `bin/shopify_api.py` script when running from the source tree. It will add the lib directory to start of `sys.path`, so the installed version won't be used.

Running Tests

```
1 pip install setuptools --upgrade
2 python setup.py test
```

Relative Cursor Pagination

Cursor based pagination support has been added in 6.0.0.

```
1 import shopify
2
3 page1 = shopify.Product.find()
4 if page1.has_next_page():
5     page2 = page1.next_page()
6
7 # to persist across requests you can use next_page_url and
  previous_page_url
8 next_url = page1.next_page_url
9 page2 = shopify.Product.find(from_=next_url)
```

Set up pre-commit locally [OPTIONAL]

Pre-commit is set up as a GitHub action that runs on pull requests and pushes to the `master` branch. If you want to run pre-commit locally, install it and set up the git hook scripts

```
1 pip install -r requirements.txt
2 pre-commit install
```

Limitations

Currently there is no support for:

-
- asynchronous requests
 - persistent connections

Additional Resources

- Partners Dashboard
- developers.shopify.com
- [Shopify.dev](https://shopify.dev)
- Ask questions on the Shopify forums

Sample apps built using this library

- Sample Django app