Shopify Flow Connector

Architecture & Usage Guide

Connector: Shopify (GraphQL Admin API, version 2026-04)

Platform: Salesforce Flow

Operations: 45

Document version: 1.0 — 26/06/2026

> *This guide is a high-level overview focused on how to use the connector. For per-operation request/response detail see the API Callouts document.*

Contents

1. What the Connector Does

The Shopify Flow Connector lets Salesforce Flow read from and write to a Shopify store. Once installed and connected, Shopify operations appear as native actions in Flow Builder — an admin or developer adds them to a flow the same way they add any other action, with no code and no manual API calls.

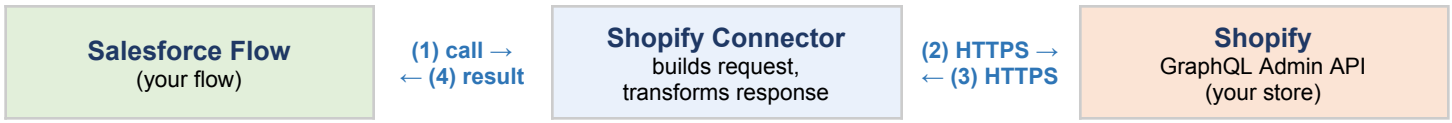
The connector covers the core Shopify commerce objects through 45 ready-to-use operations:

- **Products & variants** — list, fetch, create, update, delete, publish, and manage variants.
- **Customers & addresses** — list, search, fetch, create, delete, and manage addresses.
- **Orders, draft orders & fulfillment** — list, fetch, cancel, create/complete drafts, and fulfill.
- **Collections** — list, fetch, create, and add products.
- **Inventory & locations** — read levels, activate, adjust, set quantities, and list locations.
- **Discounts, metafields, webhooks, selling plans, shop info, and bulk export.**

> *Each operation has flat, typed inputs and outputs, so Flow Builder renders them as ordinary fields you can map to variables and records.*

2. How It Works

At a high level, the connector sits between Salesforce Flow and the Shopify GraphQL Admin API. When a flow runs an operation, the connector builds the request, sends it to Shopify over HTTPS, transforms the response into a clean structure, and hands it back to the flow.



Step by step:

1. Your flow calls a Shopify operation (e.g. Get Products) with simple input fields.
2. The connector builds the request and sends it over HTTPS to your store's Shopify Admin API.
3. Shopify processes the request and returns the data.
4. The connector cleans up the response and returns a typed result your flow can use directly.

Key things to know:

- **Outbound only.** All calls go from Salesforce to Shopify; the connector never receives inbound traffic.
- **Encrypted in transit.** Every call uses HTTPS/TLS.
- **Stateless.** The connector stores and logs nothing — it transforms data in memory and passes it straight to your flow. Where the data lands (e.g. a custom object) is decided by your flow.
- **Clean output.** Shopify's nested GraphQL structure is flattened into simple records and collections, so there are no awkward wrapper fields to dig through.

3. Installation & Deployment

The connector is delivered as a packaged artifact and deployed into your Salesforce org. After a successful deploy it is registered in the org and becomes available to Flow.

1. Deploy the connector to the org (as an IntegArtifactDef metadata component).
2. Confirm it appears under **Setup → Automation → Integrations.**
3. It is now selectable as an action inside Flow Builder.

> *Deployment is typically a one-time setup step performed by an administrator. Re-deploying a new version replaces the existing one; if Flow shows stale fields after an update, remove and re-add the connector to refresh its cached schema.*

4. Connecting to Shopify

Before using the connector in a flow, create a connection with your store's Admin API URL and an access token. The same connection is reused across all operations and flows.

4.1 What you need

Field	Value
URL	https://{your-store}.myshopify.com/admin/api/2026-04
API Key	A Shopify Admin API access token (shpat_…)

4.2 Getting an access token

1. In Shopify Admin, go to **Settings → Apps and sales channels → Develop apps.**
2. Create (or open) a custom app.
3. Grant the Admin API scopes the operations you plan to use require (e.g. read_products, write_products, read_customers, read_orders).
4. Install the app and copy the **Admin API access token** (shpat_…). It is permanent.

4.3 Creating the connection

1. **Authentication Protocol:** ApiKey.
2. **URL:** `https://{your-store}.myshopify.com/admin/api/2026-04`
3. **API Key:** your access token.
4. Save and run the **connection test** — it confirms both that the URL responds and that the token is valid.

> *The URL field is required and always shown. Enter the full Admin API URL including /admin/api/2026-04.*

5. Using the Connector in Salesforce Flow

5.1 Adding a Shopify action

1. Open **Setup → Flows** and create or edit a flow.
2. Add an **Action** element.

3. Search for the Shopify connector and pick the operation you need (e.g. Get Products, Create Customer).
4. Select your Shopify connection.

5.2 Providing inputs

Each operation shows a flat set of input fields. At least one input is always required, not all are required but it depends on Shopify. For example, Get Products:

- `first` — how many products to fetch (default 10).
- `query` — a Shopify search filter, e.g. status:active.
- `sortKey`, `reverse`, `after` — sorting and pagination.

5.3 Using the output

Operation results come back as clean, typed values you can use immediately downstream:

- **List operations** return a collection of items (under edges → node) plus pagination info and an errors list.
- **Single-item operations** return the record's fields directly.
- **Create/update/delete operations** return the affected record plus a userErrors list of any validation problems.

5.4 Saving results to Salesforce

A common pattern is to bring Shopify data into Salesforce records:

1. Call the operation (e.g. Get Products).
2. Loop over the returned collection.
3. Map each item's fields to a custom object (e.g. ShopifyProduct__c).
4. Use Create or Update Records to persist them.

5.5 Handling errors

Operations throw — they don’t return an error message within. If they do, it is a safe output, and can be used to debug.

5.6 Paging through large result sets

List operations return pageInfo with hasNextPage and endCursor. To fetch the next page, call the operation again and pass endCursor into the after input. Repeat while hasNextPage is true.

5.7 Safe retries for inventory

Inventory operations (inventoryActivate, inventoryAdjustQuantities) accept an optional idempotencyKey. Pass a stable key (one per business event) so that if a step retries, Shopify won't apply the change twice.

6. Input Reference — Data Types, Formats & Rules

Shopify's GraphQL Admin API is strict about input formats. Passing a value in the wrong shape is the most common cause of an operation that "runs but returns an error". This section documents each input data type the connector accepts, the exact format Shopify expects, and the business rules that apply. Formats below are from Shopify's official documentation for Admin API `2026-04`.

6.1 Quick reference

Input type	Format	Example
ID (GID)	gid://shopify/<Type>/<numericId>	gid://shopify/Product/108828309
Money / price	Decimal string, no symbol or code	"29.99"
Date / time	ISO 8601, UTC (Z suffix)	2026-01-31T15:50:00Z
Country	ISO 3166-1 alpha-2 code	US, AR, ES
Province / state	Region code (alphanumeric)	ON, NY, CA
Currency	ISO 4217 three-letter code	USD, EUR, ARS
Boolean	true / false	true
Integer	Whole number	10
Array / list	Salesforce Text Collection variable	(see 6.8)
Search filter	Shopify search syntax string	status:active
Phone	E.164 format	+12125551234

6.2 IDs are Global IDs (GIDs)

Every `id` input (and any `...Id` field such as `customerId`, `productId`, `locationId`, `inventoryItemId`) must be a Shopify **Global ID** (GID) — not a bare number. The format is:

```
gid://shopify/<Type>/<numericId>
```

Examples by entity:

Entity	GID example
Product	gid://shopify/Product/108828309
Product variant	gid://shopify/ProductVariant/43729076
Customer	gid://shopify/Customer/123456789
Order	gid://shopify/Order/1019283
Collection	gid://shopify/Collection/482093
Location	gid://shopify/Location/9302847
Inventory item	gid://shopify/InventoryItem/443289
Publication (channel)	gid://shopify/Publication/12345

> *Where to find a GID: in the Shopify admin, open the record (e.g. a product) and the numeric ID is in the page URL; wrap it as gid://shopify/Product/<id>. GIDs are also returned by every read operation (the id field), so a common pattern is to fetch a record first and feed its id into the next operation.*

Customer address IDs are special

A customer address ID is a `MailingAddress` GID with a `?model_name=CustomerAddress` suffix. Pass it exactly as returned, including the suffix:

```
gid://shopify/MailingAddress/10022462455996?model_name=CustomerAddress
```

This applies to `addressId` in customerAddressUpdate and customerAddressDelete.

6.3 Money & prices

Monetary inputs (e.g. a variant `price` or `compareAtPrice`) are **decimal strings with no currency symbol or code**. Shopify's Money scalar is serialized as a string.

Correct	Incorrect
"29.99"	$29.99
"100.57"	100,57
"1500.00"	1500 USD
"0.99"	.99

> *Use a period as the decimal separator, never a comma or thousands separator. The store's currency is determined by the store settings, so you supply only the amount.*

6.4 Dates & times

Date/time inputs (e.g. discount `startsAt` / `endsAt`) use the **ISO 8601** format in UTC, with a `Z` suffix:

```
YYYY-MM-DDTHH:MM:SSZ

2026-01-31T15:50:00Z    (3:50 pm UTC on 31 Jan 2026)
2026-12-25T00:00:00Z    (midnight UTC on 25 Dec 2026)
```

> *In Flow, format any date/time value to this exact string before passing it in. A date without a time, or in local format (e.g. 31/01/2026), will be rejected.*

6.5 Countries, provinces & addresses

Address inputs (customer addresses, draft-order shipping address) use a `MailingAddressInput` object. Shopify validates these against real geographic data — invalid combinations are rejected.

Country code

`countryCode` is an **ISO 3166-1 alpha-2** two-letter code, not a country name.

Country	Code
United States	US
Argentina	AR
Spain	ES
United Kingdom	GB
Canada	CA
Germany	DE

Province / state code

`provinceCode` is the region's code (e.g. `ON` for Ontario, `NY` for New York, `CA` for California) — and it must be a valid subdivision of the given country.

Full address example

```
{
  "firstName": "Ada",
  "lastName":  "Lovelace",
  "address1":  "101 Liberty Street",
  "city":      "New York",
  "provinceCode": "NY",
  "countryCode":  "US",
  "zip":       "10006",
  "phone":     "+1 212 555 0100"
}
```

> *Shopify validates that country, province, city, and zip are consistent. "New York, NY, US, 10006" is accepted; "New York, NY, AR" (Argentina) is rejected. Use the address fields exactly as they exist in the real world.*

6.6 Business rules & validation

Even with correct formats, Shopify enforces business rules. The most common ones surfaced in userErrors:

Field / area	Rule
Customer email	Must be unique in the store. Creating a customer with an email that already exists fails with "Email has already been taken".
Customer phone	Must be a valid phone number, typically in E.164 format (e.g. +12125550100). Invalid numbers are rejected.
Product handle	Must be unique. Reusing an existing handle fails; if omitted, Shopify generates one from the title.
Metafield key	Minimum 2 characters. Shorter keys are rejected.
Discount percentage	Provided as a whole number in the connector (e.g. 10 = 10% off), converted internally to the 0–1 value Shopify expects.
Order cancel reason	Must be one of Shopify's accepted reasons: CUSTOMER, DECLINED, FRAUD, INVENTORY, OTHER, STAFF.
Inventory name	available or on_hand (the quantity state being changed).
Webhook topic	A valid Shopify topic enum, e.g. ORDERS_CREATE, PRODUCTS_UPDATE, CUSTOMERS_CREATE.
Product status	DRAFT, ACTIVE, or ARCHIVED.

6.7 Search filters (the query input)

List operations accept an optional `query` string using **Shopify search syntax** — field:value terms combined with spaces (implicit AND).

Goal	query value
Active products only	status:active
Products tagged "sale"	tag:sale
Unfulfilled orders	fulfillment_status:unfulfilled
Orders created after a date	created_at:>'2026-01-01T00:00:00Z'
Customers with > 5 orders	orders_count:>5

Goal	query value
Combine filters (AND)	status:active product_type:Snowboard

> *Comparison operators (:>, :>=, :<, :<=) work on numbers and ISO 8601 dates. Quote date values with single quotes as shown.*

6.8 Array / list inputs (important)

Several operations take **array inputs** — for example `tags`, `productIds`, `topics`, and `collectionsToJoin`. These cannot be typed in as a plain comma-separated string. In Salesforce Flow you must supply a **Text Collection variable**.

How to pass an array from Flow

1. Create a **Collection variable** of Data Type Text (check "Allow multiple values (collection)").
2. Populate it — e.g. with an Assignment element, adding each value to the collection, or from a prior operation's output.
3. In the connector action, map this Text Collection variable into the array input.

> **A single text value will not satisfy an array input — Flow requires the collection variable type to match. This applies to every array input in the connector: tags, productIds, topics, options, collectionsToJoin, and the object-array inputs (variants, lineItems, sellingPlansToCreate).**

7. Operation Reference

All 45 operations. Inputs marked (req) are required; all others are optional. Outputs are the flattened structure returned to Flow (each also includes an errors list).

Products

Operation	Type	Inputs	Output (flattened)
`products`	Query	first, query, sortKey, reverse, after	edges[].node{id,title,vendor,productType,status,handle,descriptionHtml,tags,totalInventory,createdAt,updatedAt,publishedAt,onlineStoreUrl,featuredImage,priceRangeV2,variants}, pageInfo
`product`	Query	id (req)	id,title,vendor,productType,status,handle,descriptionHtml,tags,totalInventory,onlineStoreUrl,seo,options,variants,media,collections,metafields
`productCreate`	Mutation	title (req), vendor, productType, status, descriptionHtml, tags, handle, seo, productOptions, metafields, collectionsToJoin	product{...}, userErrors
`productUpdate`	Mutation	id (req), title, descriptionHtml, vendor, productType, tags, status, handle, seo	product{...}, userErrors
`productDelete`	Mutation	id (req)	deletedProductId, userErrors
`publishablePublish`	Mutation	id (req), publicationId (req)	publishable{...}, userErrors
`productVariants`	Query	productId (req), first, after	edges[].node{id,title,price,compareAtPrice,sku,barcode,inventoryQuantity,selectedOptions,inventoryItem}, pageInfo
`productVariantsBulkCreate`	Mutation	productId (req), variants (req), strategy	productVariants[]{...}, userErrors
`productVariantsBulkUpdate`	Mutation	productId (req), variants (req)	productVariants[]{...}, userErrors

Customers

Operation	Type	Inputs	Output (flattened)
`customers`	Query	first, query, sortKey, reverse, after	edges[].node{id,firstName,lastName,email,phone,state,createdAt,updatedAt,numberOfOrders,amountSpent,tags,note,verifiedEmail,taxExempt,defaultAddress}, pageInfo
`customer`	Query	id (req)	id,firstName,lastName,email,phone,state,createdAt,updatedAt,numberOfOrders,amountSpent,tags,note,defaultEmailAddress,addresses,lastOrder,metafields
`customersByEmail`	Query	email (req), first	edges[].node{id,firstName,lastName,email,phone,state,numberOfOrders,amountSpent,createdAt,tags,verifiedEmail}
`customerCreate`	Mutation	firstName, lastName, email, phone, note, tags, taxExempt, emailMarketingConsent, smsMarketingConsent	customer{...}, userErrors
`customerDelete`	Mutation	id (req)	deletedCustomerId, userErrors
`customerAddressCreate`	Mutation	customerId (req), address (req), setAsDefault	address{...}, userErrors
`customerAddressUpdate`	Mutation	customerId (req), addressId (req), address (req), setAsDefault	address{...}, userErrors
`customerAddressDelete`	Mutation	customerId (req), addressId (req)	deletedAddressId, userErrors

Orders & Draft Orders

Operation	Type	Inputs	Output (flattened)
`orders`	Query	first, query, sortKey, reverse, after	edges[].node{id,name,email,tags,createdAt,displayFinancialStatus,displayFulfillmentStatus,totalPriceSet,subtotalPriceSet,totalTaxSet,customer,lineItems}, pageInfo
`order`	Query	id (req)	id,name,email,phone,note,tags,createdAt,financial/fulfillment status, price sets, shipping/billing address, cancel info, customer, lineItems
`ordersByCustomer`	Query	customerId (req), first, sortKey, reverse, after	edges[].node{...same as orders...}, pageInfo
`orderCancel`	Mutation	orderId (req), reason (req), refund, restock, notifyCustomer	job{id}, userErrors
`draftOrders`	Query	first, query, after	edges[].node{id,name,status,totalPrice,customer,createdAt,invoiceUrl}, pageInfo
`draftOrderCreate`	Mutation	lineItems (req), customerId, shippingAddress, note, tags, email	draftOrder{id,invoiceUrl,totalPrice,status}, userErrors
`draftOrderComplete`	Mutation	id (req), paymentPending	draftOrder{id,status,order}, userErrors
`fulfillmentOrders`	Query	orderId (req), first	edges[].node{id,status,assignedLocation,lineItems}
`fulfillmentCreateV2`	Mutation	fulfillmentOrderId (req), trackingInfo, notifyCustomer	fulfillment{id,status,trackingInfo}, userErrors

Collections

Operation	Type	Inputs	Output (flattened)
`collections`	Query	first, query, after	edges[].node{id,title,handle,productsCount,updatedAt}, pageInfo
`collection`	Query	id (req), productsFirst	id,title,handle,descriptionHtml,updatedAt,products
`collectionCreate`	Mutation	title (req), descriptionHtml, ruleSet	collection{id,title,handle}, userErrors
`collectionAddProductsV2`	Mutation	collectionId (req), productIds (req)	job{id,done}, userErrors

Inventory & Locations

Operation	Type	Inputs	Output (flattened)
`inventoryLevels`	Query	inventoryItemId (req), first	edges[].node{id,updatedAt,quantities,location}, pageInfo
`inventoryActivate`	Mutation	inventoryItemId (req), locationId (req), onHand, idempotencyKey	inventoryLevel{...}, userErrors
`inventoryAdjustQuantities`	Mutation	inventoryItemId (req), locationId (req), delta (req), name (req), reason, idempotencyKey	inventoryAdjustmentGroup{...}, userErrors
`inventorySetQuantities`	Mutation	inventoryItemId (req), locationId (req), quantity (req), name (req), reason	inventoryAdjustmentGroup{...}, userErrors
`locations`	Query	first	edges[].node{id,name,isActive,address}, pageInfo

Discounts

Operation	Type	Inputs	Output (flattened)
`codeDiscountNode`	Query	id (req)	id, codeDiscount{title,status,summary,startsAt,endsAt,usageLimit,appliesOncePerCustomer,asyncUsageCount,codes,codesCount}
`discountCodeBasicCreate`	Mutation	title (req), code (req), startsAt (req), endsAt, discountPercentage, customerGets, customerSelection, minimumRequirement, usageLimit, appliesOncePerCustomer	codeDiscountNode{...}, userErrors
`discountCodeDelete`	Mutation	id (req)	deletedCodeDiscountId, userErrors

Metafields, Webhooks, Shop, Bulk, Selling Plans

Operation	Type	Inputs	Output (flattened)
`metafields`	Query	ownerId (req), ownerType (req), first, namespace	edges[].node{id,namespace,key,value,type}, pageInfo
`metafieldsSet`	Mutation	ownerId (req), namespace (req), key (req), type (req), value (req)	metafields[]{...}, userErrors
`webhookSubscriptions`	Query	first, topics	edges[].node{id,topic,endpoint,createdAt,format}
`webhookSubscriptionCreate`	Mutation	topic (req), uri (req), format	webhookSubscription{...}, userErrors
`shop`	Query	(none)	id,name,email,myshopifyDomain,currencyCode,plan,ianaTimezone,billingAddress
`bulkOperationRunQuery`	Mutation	query (req)	bulkOperation{id,status,url,errorCode}, userErrors

Operation	Type	Inputs	Output (flattened)
`sellingPlanGroupCreate`	Mutation	name (req), merchantCode (req), options (req), description, productId, sellingPlansToCreate (req)	sellingPlanGroup{...}, userErrors

8. Troubleshooting

Symptom	Likely cause	Resolution
Connection test fails with an HTML page	URL points at the storefront, or the store is password-protected	Use the full Admin API URL (…/admin/api/2026-04); disable the storefront password
Connection test fails with 401	Invalid/expired token or missing scopes	Re-check the access token and the scopes granted to the Shopify custom app
Operation returns errors but the flow keeps running	Shopify reported a problem in the result	This is expected — read the errors / userErrors list and branch on it
Flow shows old fields after an update	Cached connector schema	Remove and re-add the connector to refresh the schema