GraphQL Admin API reference

The Admin API lets you build apps and integrations that extend and enhance the Shopify admin.

This page will help you get up and running with Shopify's GraphQL API.

Client libraries

Use Shopify's officially supported libraries to build fast, reliable apps with the programming languages and frameworks you already know.

cURL −

Use the cURL utility to make API queries directly from the command line.

Remix +

Node.js +

Ruby +

Other +

cURL Remix Node.js Ruby

```
npm install --save @shopify/shopify-api
# or
yarn add @shopify/shopify-api
```

Authentication

All GraphQL Admin API queries require a valid Shopify access token.

Public and custom apps created in the Partner Dashboard generate tokens using OAuth, and custom apps made in the Shopify admin are authenticated in the Shopify admin.

Include your token as a `X-Shopify-Access-Token` header on all API queries. Using Shopify's supported client libraries can simplify this process.

To keep the platform secure, apps need to request specific access scopes during the install process. Only request as much data access as your app needs to work.

Learn more about getting started with authentication and building apps.

cURL Remix Node.js Ruby

```
const client = new shopify.clients.Graphql({session});
const response = await client.query({data: '{your_query}'});
```

Endpoint and queries

GraphQL queries are executed by sending POST HTTP requests to the endpoint:

`POST` `https://{store_name}.myshopify.com/admin/api/2025-01/graphql.json`

Queries begin with one of the objects listed under QueryRoot. The QueryRoot is the schema's entry-point for queries.

Queries are equivalent to making a GET request in REST. The example shown is a query to get the ID and title of the first three products.

Learn more about API usage.

Explore and learn Shopify's Admin API using GraphiQL Explorer. To build queries and mutations with shop data, install Shopify's GraphiQL app.

`POST` `https://{store_name}.myshopify.com/admin/api/2025-01/graphql.json`

cURL Remix Node.js Ruby

```
const queryString = `{
  products (first: 3) {
    edges {
      node {
        id
        title
      }
    }
  }
}`

// `session` is built as part of the OAuth process
const client = new shopify.clients.Graphql({session});
const products = await client.query({
  data: queryString,
});
```

Rate limits

The GraphQL Admin API is rate-limited using calculated query costs, measured in cost points. Each field returned by a query costs a set number of points. The total cost of a query is the maximum of possible fields selected, so more complex queries cost more to run.

Learn more about rate limits.

{} Request

```
{
  products(first: 1) {
    edges {
      node {
        title
      }
    }
  }
}
```

{} Response

```
{
  "data": {
    "products": {
      "edges": [
        {
          "node": {
            "title": "Hiking backpack"
          }
        }
      ]
    }
  },
  "extensions": {
    "cost": {
      "requestedQueryCost": 3,
      "actualQueryCost": 3,
      "throttleStatus": {
        "maximumAvailable": 1000.0,
        "currentlyAvailable": 997,
        "restoreRate": 50.0
      }
    }
  }
}
```

Status and error codes

All API queries return HTTP status codes that contain more information about the response.

200 OK

GraphQL HTTP status codes are different from REST API status codes. Most importantly, the GraphQL API can return a `200 OK` response code in cases that would typically produce 4xx or 5xx errors in REST.

Error handling

The response for the errors object contains additional detail to help you debug your operation.

The response for mutations contains additional detail to help debug your query. To access this, you must request `userErrors`.

Properties

`errors` • `array`

A list of all errors returned

+ Show error item properties

{} Sample 200 error responses

Throttled Internal

```
{
  "errors": [
    {
      "message": "Query cost is 2003, which exceeds the single query
max cost limit (1000).


See https://shopify.dev/concepts/about-apis/rate-limits for more
information on how the
cost of a query is calculated.

To query larger amounts of data with fewer limits, bulk operations
should be used instead.
See https://shopify.dev/tutorials/perform-bulk-operations-with-admin-
api for usage details.
",
      "extensions": {
        "code": "MAX_COST_EXCEEDED",
        "cost": 2003,
        "maxCost": 1000,
        "documentation": "https://shopify.dev/api/usage/rate-limits"
      }
    }
  ]
```

4xx and 5xx status codes

The 4xx and 5xx errors occur infrequently. They are often related to network communications, your account, or an issue with Shopify's services.

Many errors that would typically return a 4xx or 5xx status code, return an HTTP 200 errors response instead. Refer to the 200 OK section above for details.

400 Bad Request

The server will not process the request.

402 Payment Required

The shop is frozen. The shop owner will need to pay the outstanding balance to unfreeze the shop.

403 Forbidden

The shop is forbidden. Returned if the store has been marked as fraudulent.

404 Not Found

The resource isn't available. This is often caused by querying for something that's been deleted.

423 Locked

The shop isn't available. This can happen when stores repeatedly exceed API rate limits or due to fraud risk.

5xx Errors

An internal error occurred in Shopify. Check out the Shopify status page for more information.

Didn't find the status code you're looking for? View the complete list of API status response and error codes.

{} Sample error codes

400 402 403 404 423 500

```

HTTP/1.1 400 Bad Request
{
  "errors": {
    "query": "Required parameter missing or invalid"
  }
}
```