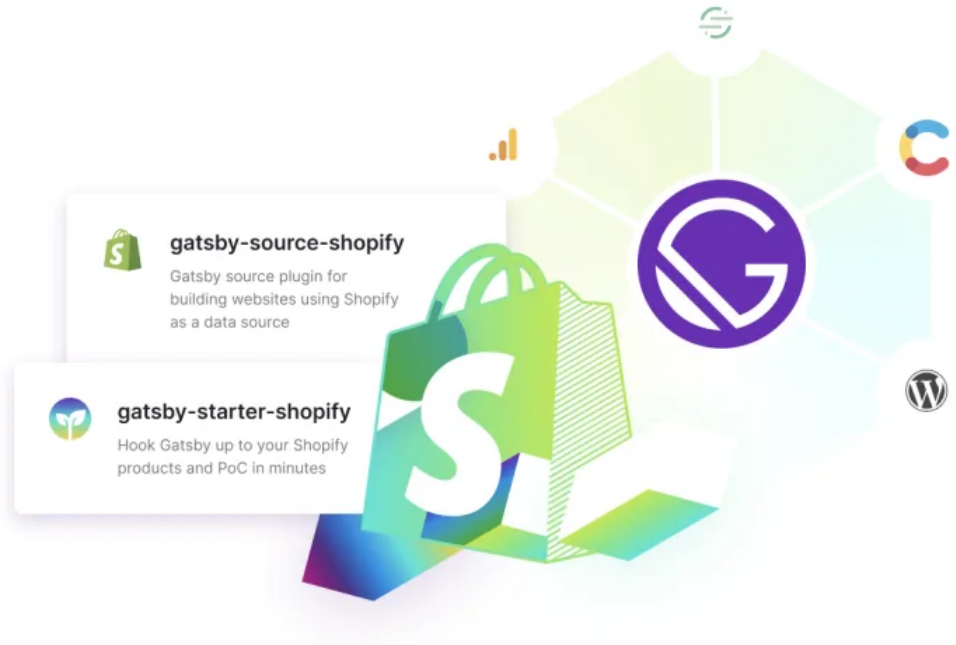

license MIT npm v7.11.0 downloads 25k/month downloads 510k PRs welcome

Follow @gatsbyjs 74k

- Getting started
  - Install
  - Configure
  - Retrieving API Information from Shopify
    - Enabling Cart and Checkout features
  - Fire it up
  - Priority builds
- Plugin options
- Images
  - Use Shopify CDN
  - Use downloaded images
  - Use run-time images
  - Displaying images
- Limitations
- V6 to V7 Migration Guide

gatsby-source-shopify

A scalable solution for sourcing data from Shopify.

This plugin works by leveraging Shopify's bulk operations API, which allows it to process large amounts of data at once. This gives it a more resilient and reliable build process. It also enables incremental builds so that your site can

build quickly when you change your data in Shopify.

Getting started

This takes you through the minimal steps to see your Shopify data in your Gatsby site's GraphiQL explorer.

Install

Install this plugin and its required peer dependency, `gatsby-plugin-image`, to your Gatsby site:

```
npm install gatsby-source-shopify gatsby-plugin-image
```

Configure

Add the plugins to your `gatsby-config.js`:

```
require("dotenv").config()

module.exports = {
  plugins: [
    {
      resolve: "gatsby-source-shopify",
      options: {
        password: process.env.SHOPIFY_APP_PASSWORD,
        storeUrl: process.env.GATSBY_MYSHOPIFY_URL,
        salesChannel: process.env.SHOPIFY_APP_ID, // Optional but recommended
      },
    },
    "gatsby-plugin-image",
  ],
}
```

Retrieving API Information from Shopify

`GATSBY_MYSHOPIFY_URL` is the Store address you enter when logging into your Shopify account. This is in the format of `my-unique-store-name.myshopify.com`.

Once logged into Shopify admin, navigate to the `Apps` page and click the link at the top to `Develop apps`. If you haven't yet, an admin on the Shopify store will need to enable private app development. This will allow you to create an app that Gatsby will use to access Shopify's Admin API.

For the Private app name enter `Gatsby` (the name does not really matter). Add the following under the `Active Permissions for this App` section:

- `Read access` for `Files`
- `Read access` for `Products`
- `Read access` for `Product listings` if you enable `collections` in the plugin options
- `Read access` for `Orders` if you enable `orders` in the plugin options
- `Read access` for `Inventory` and `Locations` if you enable `locations` in the plugin options

Enabling Cart and Checkout features

If you are planning on managing your cart within Gatsby you will also need to check the box next to `Allow this app to access your storefront data using the Storefront API` and make sure to check `Read and modify checkouts`. This source plugin does not require Shopify Storefront API access to work, however, this is needed to add items to a Shopify checkout before passing the user to Shopify's managed checkout workflow. See Gatsby Starter Shopify for an example.

Fire it up

Run your site with `gatsby develop`. When the site builds successfully, you should see output like this:

```
You can now view test-site in the browser.

  http://localhost:8000/

View GraphiQL, an in-browser IDE, to explore your site's data and schema

  http://localhost:8000/___graphql

Note that the development build is not optimized.
To create a production build, use gatsby build
```

Now follow the second link (`http://localhost:8000/___graphql`) to explore your Shopify data!

Priority builds

Because of the limitations of the Shopify Bulk API, the plugin includes logic to determine which builds are high priority for a given Shopify site. This allows the plugin to pause non-priority builds while priority builds are running while using the same Shopify App. The following logic determines whether a build is priority or not:

```
const isGatsbyCloudPriorityBuild =
  CI === `true` && GATSBY_CLOUD === `true` && GATSBY_IS_PR_BUILD !== `true`

const isNetlifyPriorityBuild =
  CI === `true` && NETLIFY === `true` && CONTEXT === `production`

return pluginOptions.prioritize !== undefined
  ? pluginOptions.prioritize
  : isGatsbyCloudPriorityBuild || isNetlifyPriorityBuild
```

This logic allows the plugin to determine whether it's running a production build on either Gatsby Cloud or Netlify using environment variables, but you also have the option to override the logic by setting the `prioritize` option in `gatsby-config`.

Plugin options

`password: string`

The admin password for the Shopify store + app you're using

`storeUrl: string`

Your Shopify store URL, e.g. my-unique-store-name.myshopify.com

`shopifyConnections: string[]`

An optional array of additional data types to source. Accepted values: `'orders'`, `'collections'`, `'locations'`

`downloadImages: bool`

Not set by default. If set to `true`, this plugin will download and process images during the build.

The plugin's default behavior is to fall back to Shopify's CDN.

`typePrefix: string`

Not set by default. If set to a string (example `MyStore`) node names will be `allMyStoreShopifyProducts` instead of `allShopifyProducts`.

`prioritize: boolean`

Not set by default. Allows you to override the priority status of a build. If set to undefined, the environment variables will determine priority status. If set to true or false, it will override the environment variables and set the priority status as such.

`salesChannel: string`

Not set by default. If set to a string (example `My Sales Channel`), only products, variants, collections, and locations that are published to that channel will be sourced. If you want to filter products by a Private App instead of a Public App or default sales channel, you have to provide the App ID instead of sales channel name. You can find this in the same place as the Shopify App Password.

💡 Note: The `salesChannel` plugin option defaults to the value of `process.env.GATBSY_SHOPIFY_SALES_CHANNEL`. If that value is not set the plugin will source only objects that are published to the `online store` sales channel.

Images

You have two options for displaying Shopify images in your Gatsby site. The default option is to use the Shopify CDN along with gatsby-plugin-image, but you can also opt-in to downloading the images as part of the build process. Your choice will result in differences to the schema. Both options are explained below.

Use Shopify CDN

This is the default behavior and is intended to be used in conjunction with gatsby-plugin-image.

Product Featured Media

This query is commonly used on collection pages to only load necessary image data.

```
query {
  products: allShopifyProduct {
    nodes {
      featuredMedia {
        preview {
          image {
```

```
            gatsbyImageData
          }
        }
      }
    }
  }
}
```

Product Media Previews

This query is commonly used on product pages to display images for all media types.

```
query {
  products: allShopifyProduct {
    nodes {
      media {
        preview {
          image {
            gatsbyImageData
          }
        }
      }
    }
  }
}
```

💡 Note: This query will return images for all media types including videos.

Product Media Previews and Videos

This query is commonly used on product pages to display images alongside videos.

```
query {
  products: allShopifyProduct {
    nodes {
      media {
        preview {
          image {
            gatsbyImageData
          }
        }
        ... on ShopifyExternalVideo {
          embeddedUrl
          host
        }
        ... on ShopifyVideo {
          sources {
            format
            height
            url
            width
          }
```

```
        }
      }
    }
  }
}
```

Use downloaded images

If you wish to download your images during the build, you can specify `downloadImages: true` as a plugin option:

```
require("dotenv").config()

module.exports = {
  plugins: [
    {
      resolve: "gatsby-source-shopify",
      options: {
        password: process.env.SHOPIFY_APP_PASSWORD,
        storeUrl: process.env.GATSBY_MYSHOPIFY_URL,
        downloadImages: true,
      },
    },
    "gatsby-plugin-image",
    "gatsby-plugin-sharp", // Required when downloadImages is true
    "gatsby-transformer-sharp", // Required when downloadImages is true
  ],
}
```

💡 Note: This will increase your build time duration with the added benefit of faster images at runtime as they are served from the same origin and not Shopify's CDN.

Media with local files

The following fragment will work with any of the `preview` fields in the runtime images section.

```
fragment MediaImageLocalFile on ShopifyMediaPreviewImage {
  image {
    localFile {
      childImageSharp {
        gatsbyImageData
      }
    }
  }
}
```

Use run-time images

If you get Shopify images at run-time that don't have the `gatsbyImageData` resolver, for example from the cart or Storefront API, you can use the `getShopifyImage` function to create an image-data object to use with `<GatsbyImage>`.

It expects an `image` object that contains the properties `width`, `height` and `originalSrc`, such as a Storefront API `Image` object.

```
import { getShopifyImage } from "gatsby-source-shopify"

function getCartImage(storefrontProduct) {
  const image = storefrontProduct.images.edges[0].node
  const imageData = getShopifyImage({
    image,
    width: 200,
    height: 200,
    layout: "fixed",
  })

  return imageData
}
```

Displaying images

```
import { GatsbyImage } from "gatsby-plugin-image"
import { getShopifyImage } from "gatsby-source-shopify"

const ShopifyProductImage = ({ product }) => (
  <GatsbyImage src={product.featuredMedia.preview.image.gatsbyImageData} />
)

const DownloadedProductImage = ({ product }) => (
  <GatsbyImage
    src={
      product.featuredMedia.preview.image.localFile.childImageSharp
        .gatsbyImageData
    }
  />
)

const RuntimeProductImage = ({ storefrontProduct }) => {
  const gatsbyImageData = getShopifyImage({
    image: storefrontProduct.images.edges[0],
    width: 800,
    height: 800,
    layout: "fixed",
  })

  return <GatsbyImage src={gatsbyImageData} />
}

const RuntimeLineItemImage = ({ storefrontLineItem }) => {
  const gatsbyImageData = getShopifyImage({
    image: storefrontLineItem.variant.image,
    width: 800,
```

```
    height: 800,
    layout: "fixed",
  })

  return <GatsbyImage src={gatsbyImageData} />
}
```

Please refer to the gatsby-plugin-image docs for more information on how to display images on your Gatsby site.

Limitations

The bulk API was chosen for resiliency, but it comes with some limitations, the most important of which is that a given Shopify App can only have one bulk operation running at a time. Because of this we recommend that you have at least two Shopify Apps for each Shopify Store, one for production and another for local development, in order to avoid potential build issues.

V6 to V7 Migration Guide

Need help upgrading this source plugin from V6 to V7? We want this guide to be as useful as possible. Please open an issue and let us know if you see anything wrong here or find something missing from this guide 🙏

Schema Changes

All `id` fields that come back from the Shopify API have now been mapped to `shopifyId` so that `id` is always intrinsic to Gatsby.

Additionally, the schema is now fully statically typed and matches the Shopify GraphQL API as closely as possible.

The following breaking schema changes must be updated in your site in order to upgrade:

`ShopifyProduct` Images/Media

Previous versions of this plugin exposed the `ShopifyProduct.images` field on products. Although it made the plugin easier to interact with, it made it impossible to add videos or 3D renderings to your products. The new version of the plugin exposes the `ShopifyProduct.media` field directly, allowing you to query for all of the images, videos and 3D renderings that Shopify supports.

It was previoulsy supported to query for videos or 3D models. In order to add support for these, the `ShopifyProduct` `images` field has been replaced by the `media` field.

Instead of doing this:

```
shopifyProduct {
  images {
    gatsbyImageData
  }
}
```

You'll now need to do this to get image data:

```
shopifyProduct {
  nodes {
```

```
    media {
      ... on ShopifyMediaImage {
        image {
          gatsbyImageData
        }
      }
    }
  }
}
```

💡 The shape of the data returned from `media` field is different than that returned from `images` which will require changes to the component code that consumes these queries in most cases.

`ShopifyProduct` Options

`ShopifyProductOption` is the type returned from `ShopifyProduct.options`.

`ShopifyProductOption.id` has been renamed to `ShopifyProductOption.shopifyId`.

Before:

```
shopifyProduct {
  options {
    id
  }
}
```

After:

```
shopifyProduct {
  options { # each of these options are of type "ShopifyProductOption"
    shopifyId
  }
}
```

Metafields

Previously, the following metafield types used to exist:

- `ShopifyProductMetafield`
- `ShopifyCollectionMetafield`
- `ShopifyProductVariantMetafield`

These have now been combined into a single `ShopifyMetafield` type.

Additionally, `Metafield.ownerType` has been changed from `string` to an `enum` type that matches the Shopify API enum for the metafield `ownerType` field

This means that any queries for metafields on a specific Shopify Owner Resource, need to be replaced like so:

```
allShopifyProductMetafield {
  nodes {
```

```
    id
    value
    description
    ownerType
  }
}
```

This will produce an equivalent to the previous example:

```
allShopifyMetafield(filter: {ownerType: {eq: PRODUCT}}) {
  nodes {
    id
    value
    description
    value
  }
}
```

Locations

Due to a bug with the Shopify API legacy locations throw an error internally in the Shopify API, `ShopifyLocation.fulfillmentService.callbackUrl` has been removed. This field will be re-added once the bug has been fixed on the Shopify side.

```
allShopifyLocation {
  edges {
    nodes {
      fulfillmentService {
        callbackUrl
      }
    }
  }
}
```