Working with Images in Gatsby

Juan Villela

> Ensuring that users get the best experience with images on the web can be a tedious task. However, with Gatsby, we can leverage the power of Sharp to get the best performance with a little setup and a rich toolset.

In this article, we'll take a look at the gatsby-image component and how it can simplify the process of using images in various scenarios. The steps in this guide assume you have a working Gatsby project. So if you haven't already, you can get started with Gatsby by following along the Gatsby First Steps article.

Alligator.io recommends ↴

React for Beginners by Wes Bos

ⓘ About this affiliate link

stay in the loop with our monthly web dev newsletter:

you@awesome.com

Subscribe!

follow us @alligatorio

site sponsors

SNIPCART

Add custom e-commerce to any site in minutes.

learn more →

about sponsors

Getting Started

Let's get started by installing the necessary plugins. Depending on the Gatsby starter you're using, these might already be installed. Check the `package.json` to see if that's the case.

Install

We're installing a few things here. Each play a different role in our image setup. We'll go into further detail in a bit.

```
$ npm install --save gatsby-image gatsby-transformer-sharp gatsby-p
```

Configuration

Now we'll add these to our `gatsby-config.js`.

↓ gatsby-config.js

```
const path = require('path');

module.exports = {
  plugins: [
    ...
    'gatsby-plugin-sharp',
    'gatsby-transformer-sharp',
    {
      resolve: 'gatsby-source-filesystem',
      options: {
        name: 'images',
        path: path.join(__dirname, `src`, `images`),
      },
    },
  ],
}
```

Notice that we are configuring gatsby-source-filesystem as well. This is to create file nodes from our images so they're available through `graphql` queries. For this guide, we're placing our images in the `src/images` directory.

As for our other plugins:

- gatsby-plugin-sharp is a low-level helper that powers the connections between **Sharp** and **gatsby-image**. It also exposes several image processing functions.
- gatsby-transformer-sharp facilitates the creation of multiples images of the right sizes and resolutions.

Working With Images

Now that we're set up, we can start working with images in our site. Let's create a `hero.js` component to use with our images.

↓ src/components/hero.js

```
import React from 'react';

export default ({ data }) => (
  <section>
    <div>
      <h1>Hi, I like websites.</h1>
      <p>Sometimes I make them.</p>
    </div>
  </section>
)
```

Querying Images

The **gatsby-transformer-sharp** plugin creates nodes of type `ImageSharp` for us to query. They can be either `fixed` or `fluid`.

- **Fixed** takes the `width` and `heigh` arguments in our queries and provides fixed-size images.
- **Fluid** takes `maxWidth` and `maxHeight` as arguments in a query and provides responsive image sizes.

Both of these will have a number of varying file sizes that utilize the <picture> element to load the right file size based on media breakpoints.

↓ src/components/hero.js

```
export const query = graphql`
  query {
    fileName: file(relativePath: { eq: "images/heroImage.jpg" }) {
      childImageSharp {
        fluid(maxWidth: 400, maxHeight: 250) {
          ...GatsbyImageSharpFluid
        }
      }
    }
  }
`
```

Our query includes the `...GatsbyImageSharpFluid` fragment, which essentially imports a few predetermined properties. You can read more about the available fragments in the gatsby-image Readme.

You can run this query in GraphiQL to browse the several useful properties at your disposal.

Using the Gatsby Image Component

Now that we have our query, let's use it in the `hero.js` component we made earlier. We'll need to import `graphql` from `gatsby` and `Img` from `gatsby-image`.

↓ src/components/hero.js

```
import React from 'react';
import { graphql } from 'gatsby';
import Img from 'gatsby-image';

export default ({ data }) => (
  <section>
    <Img
      fluid={data.file.childImageSharp.fluid}
      alt="This is a picture of my face."
    />
    <div>
      <h1>Hi, I like websites.</h1>
      <p>Sometimes I make them.</p>
    </div>
  </section>
)

export const query = graphql`
  query {
    fileName: file(relativePath: { eq: "images/heroImage.jpg" }) {
      childImageSharp {
        fluid(maxWidth: 400, maxHeight: 250) {
          ...GatsbyImageSharpFluid
        }
      }
    }
  }
`
```

Aside from taking the `alt` prop, the `Img` component will also accept `style` and `imgStyle` as objects for adding custom styling to the parent container and `img` element, respectively. For a complete list, check out the component documentation.

Gatsby will render a responsive and lazy-loaded set of images. These will be compressed, have any unnecessary metadata stripped, and include a "blur-up" effect on load. Not bad!

Polyfill

There's also a polyfill available for the object-fit/object-position CSS properties. You can instead import from `gatsby-image/withIEPolyfill`. The component will tell Gatsby to automatically apply the `object-fit-images` polyfill to your image.

↓ src/components/hero.js

```
import React from 'react';
import { graphql } from 'gatsby';
import Img from 'gatsby-image/withIEPolyfill';

export default ({ data }) => (
  <section>
    <Img
      fluid={data.file.childImageSharp.fluid}
      objectFit="cover"
      objectPosition="50% 50%"
      alt="This is a picture of my face."
    />
    ...
  </section>
)

...
```

Image Compression

By default, `gatsby-plugin-sharp` uses various compression libraries. But there are a few options we can set if we'd like to modify the default behavior.

↓ gatsby-config.js

```
module.exports = {
  plugins: [
    ...
    {
      resolve: 'gatsby-plugin-sharp',
      options: {
        useMozJpeg: false,
        stripMetadata: true,
        defaultQuality: 75,
      },
    },
  ],
}
```

We can optionally use MozJPEG for even better JPEG compression. However, keep in mind that this will likely significantly increase your build time.

Check out the plugin's documentation for all the available methods to modify and optimize your images.

Conclusion

This is only scratching the surface of what you can do with images in Gatsby. Whether you're looking to squeeze out as much performance as possible or create a quality experience for your users, Gatsby's rich toolset has you covered. I encourage you to read all the linked resources and play around with the plugins to find what best suits your needs.

Published: July 30, 2019

Fullstack Advanced React & GraphQL

Learn React + GraphQL by building an online store

ⓘ About this affiliate link

Catch up with the latest in front-end web dev with our monthly newsletter:

you@unicorn.com

Subscribe