gatsby-plugin-image

Adding responsive images to your site while maintaining high performance scores can be difficult to do manually. The Gatsby Image plugin handles the hard parts of producing images in multiple sizes and formats for you!

For full documentation on all configuration options, see the Gatsby Image Plugin reference guide

Contents

Installation

1. Install `gatsby-plugin-image` and `gatsby-plugin-sharp`. Additionally install `gatsby-source-filesystem` if you are using static images, and `gatsby-transformer-sharp` if you are using dynamic images.

```
npm install gatsby-plugin-image gatsby-plugin-sharp gatsby-source-filesystem gatsby-transformer-sharp
```

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

```
module.exports = {
  plugins: [
    `gatsby-plugin-image`,
    `gatsby-plugin-sharp`,
    `gatsby-transformer-sharp`, // Needed for dynamic images
  ],
}
```

Using the Gatsby Image components

Deciding which component to use

The Gatsby Image plugin includes two image components: one for static and one for dynamic images. An effective way to decide which you need is to ask yourself: *"will this image be the same every time the component or template is used?"*. If it will always be the same, then use `StaticImage`. If it will change, whether through data coming from a CMS or different values passed to a component each time you use it, then it is a dynamic image and you should use the `GatsbyImage` component.

Static images

If you are using an image that will be the same each time the component is used, such as a logo or front page hero image, you can use the `StaticImage` component. The image can be a local file in your project or an image hosted on a remote server. Any remote images are downloaded and resized at build time.

1. **Add the image to your project.**

   If you are using a local image, copy it into the project. A folder such as `src/images` is a good choice.

2. **Add the `StaticImage` component to your template.**

   Import the component, then set the `src` prop to point to the image you added earlier. The path is relative to the source file itself. If your component file was `src/components/dino.js`, then you would load the image like this:

   ```
   import { StaticImage } from "gatsby-plugin-image"

   export function Dino() {
     return <StaticImage src="../images/dino.png" alt="A dinosaur" />
   }
   ```

   If you are using a remote image, pass the image URL in the `src` prop:

   ```
   import { StaticImage } from "gatsby-plugin-image"

   export function Kitten() {
     return <StaticImage src="https://placekitten.com/800/600" alt="A kitten" />
   }
   ```

   When you build your site, the `StaticImage` component will load the image from your filesystem or from the remote URL, and it will generate all the sizes and formats that you need to support a responsive image.

   Because the image is loaded at build time, you cannot pass the filename in as a prop, or otherwise generate it outside of the component. It should either be a static string, or a local variable in the component's scope.

   Important: Remote images are downloaded and resized at build time. If the image is changed on the other server, it will not be updated on your site until you rebuild.

3. **Configure the image.**

   You configure the image by passing props to the `<StaticImage />` component. You can change the size and layout, as well as settings such as the type of placeholder used when lazy loading. There are also advanced image processing options available. You can find the full list of options in the API docs.

   ```
   import { StaticImage } from "gatsby-plugin-image"

   export function Dino() {
     return (
       <StaticImage
         src="../images/dino.png"
         alt="A dinosaur"
         placeholder="blurred"
         layout="fixed"
         width={200}
         height={200}
   ```

```
    />
  )
}
```

This component renders a 200px by 200px image of a dinosaur. Before loading it will have a blurred, low-resolution placeholder. It uses the `"fixed"` layout, which means the image does not resize with its container.

Restrictions on using `StaticImage`

There are a few technical restrictions to the way you can pass props into `StaticImage`. Most importantly, you can't use any of the parent component's props. For more information, refer to the Gatsby Image plugin reference guide. If you find yourself wishing you could use a prop passed from a parent for the image `src` then it's likely that you should be using a dynamic image.

Dynamic images

If you need to have dynamic images (such as if they are coming from a CMS), you can load them via GraphQL and display them using the `GatsbyImage` component.

1. **Add the image to your page query.**

   Any GraphQL File object that includes an image will have a `childImageSharp` field that you can use to query the image data. The exact data structure will vary according to your data source, but the syntax is like this:

   ```
   query {
     blogPost(id: { eq: $Id }) {
       title
       body
       avatar {
         childImageSharp {
           gatsbyImageData(width: 200)
         }
       }
     }
   }
   ```

2. **Configure your image.**

   For all the configuration options, see the Gatsby Image plugin reference guide.

   You configure the image by passing arguments to the `gatsbyImageData` resolver. You can change the size and layout, as well as settings such as the type of placeholder used when lazy loading. There are also advanced image processing options available. You can find the full list of options in the API docs.

   ```
   query {
     blogPost(id: { eq: $Id }) {
       title
       body
       author
       avatar {
   ```

```
        childImageSharp {
          gatsbyImageData(
            width: 200
            placeholder: BLURRED
            formats: [AUTO, WEBP, AVIF]
          )
        }
      }
    }
  }
```

3. **Display the image.**

   You can then use the `GatsbyImage` component to display the image on the page. The `getImage()` function is an optional helper to make your code easier to read. It takes a `File` and returns `file.childImageSharp.gatsbyImageData`, which can be passed to the `GatsbyImage` component.

```
import { graphql } from "gatsby"
import { GatsbyImage, getImage } from "gatsby-plugin-image"

function BlogPost({ data }) {
  const image = getImage(data.blogPost.avatar)
  return (
    <section>
      <h2>{data.blogPost.title}</h2>
      <GatsbyImage image={image} alt={data.blogPost.author} />
      <p>{data.blogPost.body}</p>
    </section>
  )
}

export const pageQuery = graphql`
  query {
    blogPost(id: { eq: $Id }) {
      title
      body
      author
      avatar {
        childImageSharp {
          gatsbyImageData(
            width: 200
            placeholder: BLURRED
            formats: [AUTO, WEBP, AVIF]
          )
        }
      }
    }
  }
`
```

For full APIs, see Gatsby Image plugin reference guide.

Customizing the default options

You might find yourself using the same options (like `placeholder`, `formats` etc.) with most of your `GatsbyImage` and `StaticImage` instances. You can customize the default options with `gatsby-plugin-sharp`.

The following configuration describes the options that can be customized along with their default values:

```
module.exports = {
  plugins: [
    {
      resolve: `gatsby-plugin-sharp`,
      options: {
        defaults: {
          formats: [`auto`, `webp`],
          placeholder: `dominantColor`,
          quality: 50,
          breakpoints: [750, 1080, 1366, 1920],
          backgroundColor: `transparent`,
          tracedSVGOptions: {},
          blurredOptions: {},
          jpgOptions: {},
          pngOptions: {},
          webpOptions: {},
          avifOptions: {},
        }
      }
    },
    `gatsby-transformer-sharp`,
    `gatsby-plugin-image`,
  ],
}
```

Migrating

*Main article: **Migrating from gatsby-image to gatsby-plugin-image***

If your site uses the old `gatsby-image` component, you can use a codemod to help you migrate to the new Gatsby Image components. This can update the code for most sites. To use the codemod, run this command in the root of your site:

```
npx gatsby-codemods gatsby-plugin-image
```

This will convert all GraphQL queries and components to use the new plugin. For more details, see the migration guide.