Using the Gatsby Image plugin

If you're looking for a guide on using the deprecated `gatsby-image` package, it can be found in the How to use Gatsby Image doc.

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!

Want to learn more about image optimization challenges? Read the Conceptual Guide: Why Gatsby's Automatic Image Optimizations Matter. For full documentation on all configuration options, see the reference guide.

Getting started

1. Install the following packages:

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

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

```
module.exports = {
  plugins: [
    `gatsby-plugin-image`,
    `gatsby-plugin-sharp`,
    `gatsby-transformer-sharp`,
  ],
}
```

If you already have some of these plugins installed, please check that they're updated to the latest version.

Note that `gatsby-source-filesystem` is not included in this config. If you are sourcing from your local filesystem to use `GatsbyImage` please configure accordingly. Otherwise, downloading the dependency without configuration is sufficient.

Using the Gatsby Image components

Decide which component to use

The Gatsby Image plugin includes two image components: one for static and one for dynamic images. The simplest 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 either 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.

   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.

```
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}
    />
  )
}
```

Note: There are a few technical restrictions to the way you can pass props into StaticImage. For more information, refer to the Reference Guide: Gatsby Image plugin. If you find yourself wishing you could use a prop 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. Many CMSs support `gatsby-plugin-image` without needing to download and process images locally. For these, you should see the individual plugin documentation for details on query syntax. See the CMS images section for a list of supported CMSs. For other data sources, images are downloaded and processed locally at build time. This includes local data sources where e.g. a

JSON file references an image by relative path. This section shows how to use gatsby-transformer-sharp to query for these images.

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
    # highlight-start
    avatar {
      childImageSharp {
        gatsbyImageData(width: 200)
      }
    }
    # highlight-end
  }
}
```

2. **Configure your image.**

   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 {
      # highlight-start
      childImageSharp {
        gatsbyImageData(
          width: 200
          placeholder: BLURRED
          formats: [AUTO, WEBP, AVIF]
        )
      }
      # highlight-end
    }
```

```
  }
}
```

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"
// highlight-next-line
import { GatsbyImage, getImage } from "gatsby-plugin-image"

function BlogPost({ data }) {
  // highlight-next-line
  const image = getImage(data.blogPost.avatar)
  return (
    <section>
      <h2>{data.blogPost.title}</h2>
      {/* highlight-next-line */}
      <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]
          )
        }
      }
    }
  }
`
```

Customizing the defaults

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`,
  ],
}
```

Using images from a CMS or CDN

Many source plugins have native support for `gatsby-plugin-image`, with images served directly from a content delivery network (CDN). This means that builds are faster, because there is no need to download images and process them locally. The query syntax varies according to the plugin, as do the supported transformation features and image formats. Make sure you update to the latest version of the source plugin to ensure there is support. For plugins that are not in this list you can use dynamic images from `gatsby-transformer-sharp`.

Source plugins

These source plugins support using `gatsby-plugin-image` with images served from their CDN.

- AgilityCMS
- Contentful
- DatoCMS
- GraphCMS
- Prismic
- Sanity
- Shopify

Image CDNs

A dedicated image CDN can be used with sources that don't have their own CDN, or where you need more transforms or formats than the CDN offers.

- imgix

Plugin authors

If you maintain a source plugin or image CDN, there is a toolkit to help you add support for `gatsby-plugin-image`. See Adding Gatsby Image support to your plugin for more details. You can then open a PR to add your plugin to this list.

Background images

Using CSS to display background images has more limited support for responsive image handling than the `<picture>` element. Most importantly, it does not handle fallback for next-gen image formats such as AVIF and WebP. You can get the benefits of `gatsby-plugin-image` for background images without any extra components.

This is an example of a hero image component with text overlaying an image background. It uses CSS grid to stack the elements on top of each other.

```
import * as React from "react"
import { StaticImage } from "gatsby-plugin-image"

export function Hero() {
  return (
    <div style={{ display: "grid" }}>
      {/* You can use a GatsbyImage component if the image is dynamic */}
      <StaticImage
        style={{
          gridArea: "1/1",
          // You can set a maximum height for the image, if you wish.
          // maxHeight: 600,
        }}
        layout="fullWidth"
        // You can optionally force an aspect ratio for the generated image
        aspectRatio={3 / 1}
```

```
        // This is a presentational image, so the alt should be an empty string
        alt=""
        // Assisi, Perúgia, Itália by Bernardo Ferrari, via Unsplash
        src={
          "https://images.unsplash.com/photo-1604975999044-188783d54fb3?w=2589"
        }
        formats={["auto", "webp", "avif"]}
      />
      <div
        style={{
          // By using the same grid area for both, they are stacked on top of each other
          gridArea: "1/1",
          position: "relative",
          // This centers the other elements inside the hero component
          placeItems: "center",
          display: "grid",
        }}
      >
        {/* Any content here will be centered in the component */}
        <h1>Hero text</h1>
      </div>
    </div>
  )
}
```

Migrating

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 information see the full migration guide.

Troubleshooting

If you're running into issues getting everything to work together we recommend following these steps.

1. Are your dependencies installed?

Check your package.json file for the following entries:

- `gatsby-plugin-image`
- `gatsby-plugin-sharp`

- `gatsby-transformer-sharp` (If you're querying for dynamic images)
- `gatsby-source-filesystem` (If you're using the StaticImage component)

If not, install them.

2. Have you added the necessary information to your `gatsby-config.js` file?

All of these plugins should be in your plugins array. Reminder that the `gatsby-image` package did not get included, so this is a change.

- `gatsby-plugin-image`
- `gatsby-plugin-sharp`
- `gatsby-transformer-sharp` (If you're querying for dynamic images)

If not, add them.

3. Do your queries in GraphiQL return the data you expect?

There are two paths here.

If you're using StaticImage:

Try running

```
query MyQuery {
  allFile {
    nodes {
      relativePath
    }
  }
}
```

Do you see results with `.cache/caches/gatsby-plugin-image` in the path?

If not, check steps 1 and 2 above.

If you're using GatsbyImage:

Run the query you're using in your site. Does it return a `gatsbyImageData` object?

If not, check steps 1 and 2 above.

4. Do the images render when you run your site?

Do you see errors in your `gatsby develop` logs? Do you see errors in your browser console?

For StaticImage:

Inspect the element on your page. Does it include a `div` with `gatsby-image-wrapper`? How about an internal `source` element?

If not, double-check the component usage. Did you use `src`? Is the relative path correct?

For GatsbyImage:

Trying using `console.log` to check the object you're passing as the `image` prop. Is it there?

If not, make sure you're using the same object you saw return from your query.

5. Are the images working yet?

If you still see problems and think it's a bug, please file an issue and let us know how far in these steps you progressed.