Access images as static resources, or automate the process of optimizing them through powerful plugins.

Import an image into a component with webpack

Images can be imported right into a JavaScript module with webpack. This process automatically minifies and copies the image to your site's `public` folder, providing a dynamic image URL for you to pass to an HTML `<img>` element like a regular file path.

Prerequisites

- A Gatsby Site with a `.js` file exporting a React component
- an image (`.jpg` , `.png` , `.gif` , `.svg` , etc.) in the `src` folder

Directions

1. Import your file from its path in the `src` folder:

```
import React from "react"
// Tell webpack this JS file uses this image
import FiestaImg from "../assets/fiesta.jpg" // highlight-line
```

2. In `index.js` , add an `<img>` tag with the `src` as the name of the import you used from webpack (in this case `FiestaImg`), and add an `alt` attribute describing the image:

```
import React from "react"
import FiestaImg from "../assets/fiesta.jpg"

export default function Home() {
  return (
    // The import result is the URL of your image
    <img src={FiestaImg} alt="A dog smiling in a party hat" /> // highlight-line
  )
}
```

3. Run `gatsby develop` to start the development server.
4. View your image in the browser: `http://localhost:8000/`

Additional resources

- Example repo importing an image with webpack
- More on all image techniques in Gatsby

Reference an image from the `static` folder

As an alternative to importing assets with webpack, the `static` folder allows access to content that gets automatically copied into the `public` folder when built.

This is an **escape route** for specific use cases, and other methods like importing with webpack are recommended to leverage optimizations made by Gatsby.

Prerequisites

- A Gatsby Site with a `.js` file exporting a React component
- An image (`.jpg` , `.png` , `.gif` , `.svg` , etc.) in the `static` folder

Directions

1. Ensure that the image is in your `static` folder at the root of the project. Your project structure might look something like this:

```
├── gatsby-config.js
├── src
│   └── pages
│       └── index.js
├── static
│       └── fiesta.jpg
```

2. In `index.js` , add an `<img>` tag with the `src` as the relative path of the file from the `static` folder, and add an `alt` attribute describing the image:

```
import React from "react"

export default function Home() {
  return (
    <img src={`fiesta.jpg`} alt="A dog smiling in a party hat" /> // highlight-line
  )
}
```

3. Run `gatsby develop` to start the development server.
4. View your image in the browser: `http://localhost:8000/`

Additional resources

- Example repo referencing an image from the static folder
- Using the Static Folder
- More on all image techniques in Gatsby

Optimizing and querying local images with gatsby-image

The `gatsby-image` plugin can relieve much of the pain associated with optimizing images in your site.

Gatsby will generate optimized resources which can be queried with GraphQL and passed into Gatsby's image component. This takes care of the heavy lifting including creating several image sizes and loading them at the right time.

Prerequisites

- The `gatsby-image` , `gatsby-transformer-sharp` , and `gatsby-plugin-sharp` packages installed and added to the plugins array in `gatsby-config`
- Images sourced in your `gatsby-config` using a plugin like `gatsby-source-filesystem`

Directions

1. First, import `Img` from `gatsby-image` , as well as `graphql` and `useStaticQuery` from `gatsby`

```
import { useStaticQuery, graphql } from "gatsby" // to query for image data
import Img from "gatsby-image" // to take image data and render it
```

2. Write a query to get image data, and pass the data into the `<Img />` component:

Choose any of the following options or a combination of them.

a. a single image queried by its file path (Example: `images/corgi.jpg`)

```
const data = useStaticQuery(graphql`
  query {
    file(relativePath: { eq: "corgi.jpg" }) { // highlight-line
      childImageSharp {
        fluid {
          base64
          aspectRatio
          src
          srcSet
          sizes
        }
      }
    }
  }
`)

return (
  <Img fluid={data.file.childImageSharp.fluid} alt="A corgi smiling happily" />
)
```

b. using a GraphQL fragment, to query for the necessary fields more tersely

```
const data = useStaticQuery(graphql`
  query {
    file(relativePath: { eq: "corgi.jpg" }) {
      childImageSharp {
        fluid {
          ...GatsbyImageSharpFluid // highlight-line
        }
      }
    }
  }
`)

return (
  <Img fluid={data.file.childImageSharp.fluid} alt="A corgi smiling happily" />
)
```

c. several images from a directory (Example: `images/dogs`) filtered by the `extension` and `relativeDirectory` fields, and then mapped into `Img` components

```
const data = useStaticQuery(graphql`
  query {
    allFile(
      // highlight-start
      filter: {
        extension: { regex: "/(jpg)|(png)|(jpeg)/" }
        relativeDirectory: { eq: "dogs" }
      }
      // highlight-end
    ) {
      edges {
        node {
          base
          childImageSharp {
            fluid {
              ...GatsbyImageSharpFluid
            }
          }
        }
      }
    }
  }
`)

return (
  <div>
    // highlight-start
    {data.allFile.edges.map(image => (
      <Img
        fluid={image.node.childImageSharp.fluid}
        alt={image.node.base.split(".")[0]} // only use section of the file
extension with the filename
      />
    ))}
    // highlight-end
  </div>
)
```

Note: This method can make it difficult to match images with `alt` text for accessibility. This example uses images with `alt` text included in the filename, like `dog in a party hat.jpg`.

d. an image of a fixed size using the `fixed` field instead of `fluid`

```
const data = useStaticQuery(graphql`
  query {
    file(relativePath: { eq: "corgi.jpg" }) {
      childImageSharp {
        fixed(width: 250, height: 250) { // highlight-line
          ...GatsbyImageSharpFixed
        }
      }
```

```
    }
  }
`)
return (
  <Img fixed={data.file.childImageSharp.fixed} alt="A corgi smiling happily" />
)
```

e. an image of a fixed size with a `maxWidth`

```
const data = useStaticQuery(graphql`
  query {
    file(relativePath: { eq: "corgi.jpg" }) {
      childImageSharp {
        fixed(maxWidth: 250) { // highlight-line
          ...GatsbyImageSharpFixed
        }
      }
    }
  }
`)
return (
  <Img fixed={data.file.childImageSharp.fixed} alt="A corgi smiling happily" /> //
highlight-line
)
```

f. an image filling a fluid container with a max width (in pixels) and a higher quality (the default value is 50 i.e. 50%)

```
const data = useStaticQuery(graphql`
  query {
    file(relativePath: { eq: "corgi.jpg" }) {
      childImageSharp {
        fluid(maxWidth: 800, quality: 75) { // highlight-line
          ...GatsbyImageSharpFluid
        }
      }
    }
  }
`)

return (
  <Img fluid={data.file.childImageSharp.fluid} alt="A corgi smiling happily" />
)
```

3. (Optional) Add inline styles to the `<Img />` like you would to other components

```
<Img
  fluid={data.file.childImageSharp.fluid}
  alt="A corgi smiling happily"
  style={{ border: "2px solid rebeccapurple", borderRadius: 5, height: 250 }} //
```

```
highlight-line
/>
```

4. (Optional) Force an image into a desired aspect ratio by overriding the `aspectRatio` field returned by the GraphQL query before it is passed into the `<Img />` component

```
<Img
  fluid={{
    ...data.file.childImageSharp.fluid,
    aspectRatio: 1.6, // 1280 / 800 = 1.6
  }}
  alt="A corgi smiling happily"
/>
```

5. Run `gatsby develop`, to generate images from files in the filesystem (if not done already) and cache them

Additional resources

- Example repository illustrating these examples
- Gatsby Image API
- Using Gatsby Image
- More on working with images in Gatsby
- Free egghead.io videos explaining these steps

Optimizing and querying images in post frontmatter with gatsby-image

For use cases like a featured image in a blog post, you can *still* use `gatsby-image`. The `Img` component needs processed image data, which can come from a local (or remote) file, including from a URL in the frontmatter of a `.md` or `.mdx` file.

To inline images in markdown (using the `![]()` syntax), consider using a plugin like `gatsby-remark-images`

Prerequisites

- The `gatsby-image`, `gatsby-transformer-sharp`, and `gatsby-plugin-sharp` packages installed and added to the plugins array in `gatsby-config`
- Images sourced in your `gatsby-config` using a plugin like `gatsby-source-filesystem`
- Markdown files sourced in your `gatsby-config` with image URLs in frontmatter
- Pages created from Markdown using `createPages`

Directions

1. Verify that the Markdown file has an image URL with a valid path to an image file in your project

```
---
title: My First Post
featuredImage: ./corgi.png // highlight-line
---

Post content...
```

2. Verify that a unique identifier (a slug in this example) is passed in context when `createPages` is called in `gatsby-node.js`, which will later be passed into a GraphQL query in the Layout component

```
exports.createPages = async ({ graphql, actions }) => {
  const { createPage } = actions

  // query for all markdown

  result.data.allMdx.edges.forEach(({ node }) => {
    createPage({
      path: node.fields.slug,
      component: path.resolve(`./src/components/markdown-layout.js`),
      // highlight-start
      context: {
        slug: node.fields.slug,
      },
      // highlight-end
    })
  })
}
```

3. Now, import `Img` from `gatsby-image`, and `graphql` from `gatsby` into the template component, write a pageQuery to get image data based on the passed in `slug` and pass that data to the `<Img />` component:

```
import React from "react"
import { graphql } from "gatsby" // highlight-line
import Img from "gatsby-image" // highlight-line

export default function Layout({ children, data }) {
  return (
    <main>
      // highlight-start
      <Img
        fluid={data.markdown.frontmatter.image.childImageSharp.fluid}
        alt="A corgi smiling happily"
      />
      // highlight-end
      {children}
    </main>
  )
}

// highlight-start
export const pageQuery = graphql`
  query PostQuery($slug: String) {
    markdown: mdx(fields: { slug: { eq: $slug } }) {
      id
      frontmatter {
        image {
```

```
            childImageSharp {
              fluid {
                ...GatsbyImageSharpFluid
              }
            }
          }
        }
      }
    }
`
// highlight-end
```

4. Run `gatsby develop`, which will generate images for files sourced in the filesystem

Additional resources

- Example repository using this recipe
- Featured images with frontmatter
- Gatsby Image API
- Using Gatsby Image
- More on working with images in Gatsby