Working with Images in Markdown Posts and Pages

When building Gatsby sites composed primarily of Markdown pages or posts, insertion of images can enhance the content. You can add images in multiple ways.

Featured images with Frontmatter metadata

In sites like a blog, you may want to include a featured image that appears at the top of a page. One way to do this is to grab the image filename from a frontmatter field and then transform it with `gatsby-plugin-sharp` in a GraphQL query.

This solution assumes you already have programmatically generated pages from Markdown with renderers like `gatsby-transformer-remark` or `gatsby-plugin-mdx`. If not, take a read through up to Part 7 of the Gatsby Tutorial. This will build upon the tutorial and as such, `gatsby-transformer-remark` will be used for this example.

> Note: This can be done similarly using MDX as well. Instead of the `markdownRemark` nodes in GraphQL, `Mdx` can be swapped in and should work.

To start out, download the plugins for Gatsby-image as mentioned in Using gatsby-image.

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

You will also want to have `gatsby-source-filesystem` installed. Then, configure the various plugins in the `gatsby-config` file.

Configuring for images and posts in the same directory

If your images are in the same directory as the Markdown files, sourcing and resolving the images can be done in one configuration. For example, if your Markdown pages and images are located together in a `/pages` directory, both content types will be automatically picked up by GraphQL as part of Gatsby's data layer.

```
module.exports = {
  plugins: [
    `gatsby-plugin-sharp`,
    `gatsby-transformer-sharp`,
    `gatsby-transformer-remark`,
    {
      resolve: `gatsby-source-filesystem`,
      options: {
        path: `${__dirname}/src/pages`, // highlight-line
      },
    },
  ],
}
```

Then, in an example Markdown file, add a field called `featuredImage`:

```
---
title: My Favorite Doggos
featuredImage: pupperino.png
---

Content goes here!
```

The next step will be to incorporate the data into a template with a GraphQL query, which can be found later in this guide.

Configuring for images and posts in different directories

There are also occasions when you may want to source images from a different directory than where your Markdown posts or pages are located, such as in an external `/images` folder. You can set this up by specifying two distinct sources, one for the pages and the other for images:

```
module.exports = {
  plugins: [
    `gatsby-plugin-sharp`,
    `gatsby-transformer-sharp`,
    `gatsby-transformer-remark`,
    {
      resolve: `gatsby-source-filesystem`,
      options: {
        path: `${__dirname}/src/pages`, // highlight-line
      },
    },
    {
      resolve: `gatsby-source-filesystem`,
      options: {
        path: `${__dirname}/src/images`, // highlight-line
```

```
      },
    },
  ],
}
```

Then, in a Markdown file, the path to a `featuredImage` would be relative to the page file (in this case, in an `/images` directory up a level):

```
---
title: About
featuredImage: ../images/team-cat.png
---

Content goes here!
```

Querying for images from Frontmatter

Now that you've sourced Markdown and image data, you can query for featured images in GraphQL. If a filepath points to an actual image, it will be transformed into a `File` node in GraphQL and then you can get the image data out of it by using the `childImageSharp` field.

This can be added to the GraphQL query in a Markdown template file. In this example, a Fluid query is used to make a responsive image.

```
export const query = graphql`
  query PostQuery($slug: String!) {
    markdownRemark(fields: { slug: { eq: $slug } }) {
      html
      frontmatter {
        title
        // highlight-start
        featuredImage {
          childImageSharp {
            fluid(maxWidth: 800) {
              ...GatsbyImageSharpFluid
            }
          }
        }
        // highlight-end
      }
    }
  }
`
```

Also in the Markdown post template, import the `gatsby-image` package and pass the results of the GraphQL query into an `<Img />` component.

```
import React from "react"
```

```
import { graphql } from "gatsby"
import Layout from "../components/layout"
// highlight-start
import Img from "gatsby-image"
// highlight-end

export default function BlogPost({ data }) {
  let post = data.markdownRemark

  // highlight-start
  let featuredImgFluid = post.frontmatter.featuredImage.childImageSharp.fluid
  // highlight-end

  return (
    <Layout>
      <div>
        <h1>{post.frontmatter.title}</h1>
        // highlight-start
        <Img fluid={featuredImgFluid} />
        // highlight-end
        <div dangerouslySetInnerHTML={{ __html: post.html }} />
      </div>
    </Layout>
  )
}

export const query = graphql`
  query PostQuery($slug: String!) {
    markdownRemark(fields: { slug: { eq: $slug } }) {
      html
      frontmatter {
        title
        featuredImage {
          childImageSharp {
            fluid(maxWidth: 800) {
              ...GatsbyImageSharpFluid
            }
          }
        }
      }
    }
  }
`
```

Your featured image should now appear on the generated page right below the main header. Tada!

Inline images with `gatsby-remark-images`

You may also include images in the Markdown body itself. The plugin gatsby-remark-images comes in handy for this.

Start out by installing `gatsby-remark-images` and `gatsby-plugin-sharp`.

```
npm install gatsby-remark-images gatsby-plugin-sharp
```

Also make sure that `gatsby-source-filesystem` is installed and points at the directory where your images are located.

Configure the plugins in your `gatsby-config` file. As with the previous example, either `Remark` or `MDX` can be used.

Using the MDX Plugin

The below example uses the `gatsby-plugin-mdx` plugin.

`gatsby-remark-images` needs to be both a sub-plugin of `gatsby-plugin-mdx`, included in the `options` field, and a string entry in the plugins array. `gatsby-plugin-sharp` can be included on its own.

`gatsby-source-filesystem` needs to be pointed at wherever you have your images on disk.

> Note: This example configuration assumes your images and Markdown pages are sourced from the same directory. Check out the section on configuring for different directories for additional help.

```
module.exports = {
  plugins: [
    `gatsby-plugin-sharp`,
    `gatsby-remark-images`,
    {
      resolve: `gatsby-plugin-mdx`,
      options: {
        gatsbyRemarkPlugins: [
          {
            resolve: `gatsby-remark-images`,
            options: {
              maxWidth: 1200,
            },
          },
        ],
      },
    },
    {
      resolve: `gatsby-source-filesystem`,
      options: {
```

```
        path: `${__dirname}/src/pages`,
      },
    },
  ],
}
```

Using the Transformer Remark Plugin

Here is a similar example using the `gatsby-transformer-remark` plugin instead of `gatsby-plugin-mdx`. Put the `gatsby-remark-images` plugin within the `plugins` option field of `gatsby-transformer-remark`.

```
module.exports = {
  plugins: [
    `gatsby-plugin-sharp`,
    {
      resolve: `gatsby-transformer-remark`,
      options: {
        plugins: [
          {
            resolve: `gatsby-remark-images`,
            options: {
              maxWidth: 800,
            },
          },
        ],
      },
    },
    {
      resolve: `gatsby-source-filesystem`,
      options: {
        path: `${__dirname}/src/posts`,
      },
    },
  ],
}
```

With the configurations above, you can use the default Markdown syntax for images. They will be processed by Sharp and appear as if you placed them in a `gatsby-image` component.

```
![Hopper The Rabbit](./rabbit-friend.png)
```