Migrating from Gatsby

This guide will help you understand how to transition from an existing Gatsby project to Next.js. Migrating to Next.js will allow you to:

- Choose which data fetching strategy you want on a per-page basis.
- Use Incremental Static Regeneration to update *existing* pages by re-rendering them in the background as traffic comes in.
- Use API Routes.

And more! Let's walk through a series of steps to complete the migration.

Updating `package.json` and dependencies

The first step towards migrating to Next.js is to update `package.json` and dependencies. You should:

- Remove all Gatsby-related packages (but keep `react` and `react-dom`).
- Install `next` .
- Add Next.js related commands to `scripts` . One is `next dev` , which runs a development server at `localhost:3000` . You should also add `next build` and `next start` for creating and starting a production build.

Here's an example `package.json` (view diff):

```
{
  "scripts": {
    "dev": "next dev",
    "build": "next build",
    "start": "next start"
  },
  "dependencies": {
    "next": "latest",
    "react": "latest",
    "react-dom": "latest"
  }
}
```

Static Assets and Compiled Output

Gatsby uses the `public` directory for the compiled output, whereas Next.js uses it for static assets. Here are the steps for migration (view diff):

- Remove `.cache/` and `public` from `.gitignore` and delete both directories.
- Rename Gatsby's `static` directory as `public` .
- Add `.next` to `.gitignore` .

Creating Routes

Both Gatsby and Next support a `pages` directory, which uses file-system based routing. Gatsby's directory is `src/pages` , which is also supported by Next.js.

Gatsby creates dynamic routes using the `createPages` API inside of `gatsby-node.js`. With Next, we can use Dynamic Routes inside of `pages` to achieve the same effect. Rather than having a `template` directory, you can use the React component inside your dynamic route file. For example:

- **Gatsby:** `createPages` API inside `gatsby-node.js` for each blog post, then have a template file at `src/templates/blog-post.js`.
- **Next:** Create `pages/blog/[slug].js` which contains the blog post template. The value of `slug` is accessible through a query parameter. For example, the route `/blog/first-post` would forward the query object `{ 'slug': 'first-post' }` to `pages/blog/[slug].js` (learn more here).

Styling

With Gatsby, global CSS imports are included in `gatsby-browser.js`. With Next, you should create a custom `_app.js` for global CSS. When migrating, you can copy over your CSS imports directly and update the relative file path, if necessary. Next.js has built-in CSS support.

Links

The Gatsby `Link` and Next.js `Link` component have a slightly different API.

```
// Gatsby

import { Link } from 'gatsby'

export default function Home() {
  return <Link to="/blog">blog</Link>
}
```

```
// Next.js

import Link from 'next/link'

export default function Home() {
  return (
    <Link href="/blog">
      <a>blog</a>
    </Link>
  )
}
```

Update any import statements, switch `to` to `href`, and add an `<a>` tag as a child of the element.

Data Fetching

The largest difference between Gatsby and Next.js is how data fetching is implemented. Gatsby is opinionated with GraphQL being the default strategy for retrieving data across your application. With Next.js, you get to choose which strategy you want (GraphQL is one supported option).

Gatsby uses the `graphql` tag to query data in the pages of your site. This may include local data, remote data, or information about your site configuration. Gatsby only allows the creation of static pages. With Next.js, you can choose on a per-page basis which data fetching strategy you want. For example, `getServerSideProps` allows you to do server-side rendering. If you wanted to generate a static page, you'd export `getStaticProps` / `getStaticPaths` inside the page, rather than using `pageQuery`. For example:

```
// src/pages/[slug].js

// Install remark and remark-html
import { remark } from 'remark'
import html from 'remark-html'
import { getPostBySlug, getAllPosts } from '../lib/blog'

export async function getStaticProps({ params }) {
  const post = getPostBySlug(params.slug)
  const markdown = await remark()
    .use(html)
    .process(post.content || '')
  const content = markdown.toString()

  return {
    props: {
      ...post,
      content,
    },
  }
}

export async function getStaticPaths() {
  const posts = getAllPosts()

  return {
    paths: posts.map((post) => {
      return {
        params: {
          slug: post.slug,
        },
      }
    }),
    fallback: false,
  }
}
```

You'll commonly see Gatsby plugins used for reading the file system (`gatsby-source-filesystem`), handling markdown files (`gatsby-transformer-remark`), and so on. For example, the popular starter blog example has 15 Gatsby specific packages. Next takes a different approach. It includes common features directly inside the framework, and gives the user full control over integrations with external packages. For example, rather than abstracting reading from the file system to a plugin, you can use the native Node.js `fs` package inside `getStaticProps` / `getStaticPaths` to read from the file system.

```
// src/lib/blog.js
// Install gray-matter and date-fns
import matter from 'gray-matter'
import { parseISO, format } from 'date-fns'
import fs from 'fs'
import { join } from 'path'

// Add markdown files in `src/content/blog`
const postsDirectory = join(process.cwd(), 'src', 'content', 'blog')

export function getPostBySlug(slug) {
  const realSlug = slug.replace(/\.md$/, '')
  const fullPath = join(postsDirectory, `${realSlug}.md`)
  const fileContents = fs.readFileSync(fullPath, 'utf8')
  const { data, content } = matter(fileContents)
  const date = format(parseISO(data.date), 'MMMM dd, yyyy')

  return { slug: realSlug, frontmatter: { ...data, date }, content }
}

export function getAllPosts() {
  const slugs = fs.readdirSync(postsDirectory)
  const posts = slugs.map((slug) => getPostBySlug(slug))

  return posts
}
```

Image Component and Image Optimization

Next.js has a built-in Image Component and Automatic Image Optimization.

The Next.js Image Component, `next/image` , is an extension of the HTML `<img>` element, evolved for the modern web.

The Automatic Image Optimization allows for resizing, optimizing, and serving images in modern formats like WebP when the browser supports it. This avoids shipping large images to devices with a smaller viewport. It also allows Next.js to automatically adopt future image formats and serve them to browsers that support those formats.

Migrating from Gatsby Image

Instead of optimizing images at build time, Next.js optimizes images on-demand, as users request them. Unlike static site generators and static-only solutions, your build times aren't increased, whether shipping 10 images or 10 million images.

This means you can remove common Gatsby plugins like:

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

Instead, use the built-in `next/image` component and Automatic Image Optimization.

> *The `next/image` component's default loader is not supported when using `next export` . However, other loader options will work.*

```
import Image from 'next/image'
import profilePic from '../public/me.png'

export default function Home() {
  return (
    <>
      <h1>My Homepage</h1>
      <Image
        src={profilePic}
        alt="Picture of the author"
        // When "responsive", similar to "fluid" from Gatsby
        // When "intrinsic", similar to "fluid" with maxWidth from Gatsby
        // When "fixed", similar to "fixed" from Gatsby
        layout="responsive"
        // Optional, similar to "blur-up" from Gatsby
        placeholder="blur"
        // Optional, similar to "width" in Gatsby GraphQL
        width={500}
        // Optional, similar to "height" in Gatsby GraphQL
        height={500}
      />
      <p>Welcome to my homepage!</p>
    </>
  )
}
```

Site Configuration

With Gatsby, your site's metadata (name, description, etc.) is located inside `gatsby-config.js`. This is then exposed through the GraphQL API and consumed through a `pageQuery` or a static query inside a component.

With Next.js, we recommend creating a config file similar to below. You can then import this file anywhere without having to use GraphQL to access your site's metadata.

```
// src/config.js

export default {
  title: 'Starter Blog',
  author: {
    name: 'Lee Robinson',
    summary: 'who loves Next.js.',
  },
  description: 'A starter blog converting Gatsby -> Next.',
  social: {
    twitter: 'leeerob',
  },
}
```

Search Engine Optimization

Most Gatsby examples use `react-helmet` to assist with adding `meta` tags for proper SEO. With Next.js, we use [`next/head`]() to add `meta` tags to your `<head />` element. For example, here's an SEO component with Gatsby:

```
// src/components/seo.js

import { Helmet } from 'react-helmet'

export default function SEO({ description, title, siteTitle }) {
  return (
    <Helmet
      title={title}
      titleTemplate={siteTitle ? `%s | ${siteTitle}` : null}
      meta={[
        {
          name: `description`,
          content: description,
        },
        {
          property: `og:title`,
          content: title,
        },
        {
          property: `og:description`,
          content: description,
        },
        {
          property: `og:type`,
          content: `website`,
        },
        {
          name: `twitter:card`,
          content: `summary`,
        },
        {
          name: `twitter:creator`,
          content: twitter,
        },
        {
          name: `twitter:title`,
          content: title,
        },
        {
          name: `twitter:description`,
          content: description,
        },
      ]}
    />
  )
}
```

And here's the same example using Next.js, including reading from a site config file.

```
// src/components/seo.js

import Head from 'next/head'
import config from '../config'

export default function SEO({ description, title }) {
  const siteTitle = config.title

  return (
    <Head>
      <title>{`${title} | ${siteTitle}`}</title>
      <meta name="description" content={description} />
      <meta property="og:type" content="website" />
      <meta property="og:title" content={title} />
      <meta property="og:description" content={description} />
      <meta property="og:site_name" content={siteTitle} />
      <meta property="twitter:card" content="summary" />
      <meta property="twitter:creator" content={config.social.twitter} />
      <meta property="twitter:title" content={title} />
      <meta property="twitter:description" content={description} />
    </Head>
  )
}
```

Learn more

Take a look at this pull request for more details on how an app can be migrated from Gatsby to Next.js. If you have questions or if this guide didn't work for you, feel free to reach out to our community on GitHub Discussions.