This guide demonstrates how to fetch data at both *build time* and *runtime* in Gatsby. Most of the techniques outlined are for custom data handling. Be sure to check out Gatsby's plugin library to see if there's an off-the-shelf solution for your data requirements, such as sourcing from a CMS or other third-party integration.

The benefits of the hybrid nature of Gatsby apps

Because Gatsby is capable of generating content at build time as well as making calls to external services at runtime, you can make hybrid pages that take advantage of the benefits of static content as well as dynamic content. You can gather data ahead of time while the site builds so that when a user loads your page the data is already available. Then, for data that is of a more dynamic nature, you can request data from another service like an API with `fetch` or a library like `axios`.

This combination of static/dynamic is possible through React hydration, which means that Gatsby (through React.js) builds static files to render server-side. When Gatsby's script bundle downloads and executes in the browser, it preserves the HTML markup built by Gatsby and turns the site into a full React web application that can manipulate the DOM. The result of this process creates fast loading pages and a nice user experience.

> *To understand how statically generated content can turn into a React app, refer to the Understanding React Hydration guide*

Compiling pages at build time is useful when your website content won't change often, or when triggering a build process to recompile works fine. However, some websites with more dynamic needs require a client-side runtime to handle constantly changing content after the page loads, like a chat widget, user upvotes, or an email client web application.

Combining build time and client runtime data

To illustrate a combination of build time and client runtime data, this guide uses code from a small example site. It uses the `gatsby-source-graphql` plugin to fetch data from GitHub's GraphQL API at build time for static content like the name and URL to a repository, and the `fetch` API to retrieve more dynamic data from the GitHub API on the client-side like star counts when the page loads in the browser. Reasons to fetch certain data at build time vs. client runtime will vary, but in this example the repo's name and URL are much less likely to change between builds of the site. The repo's star counts, on the other hand, are likely to change often and would benefit from a client-side request to the GitHub API to stay current between static site builds.

> *Check out the code from the full example here.*

Fetching data at build time

In order to fetch data at build time, you can use a source plugin or source data yourself. To source data yourself you can create an integration with a third-party system by creating nodes for the GraphQL layer in your `gatsby-node` file from retrieved data that becomes queryable in pages. This is the same method that source plugins implement to source data while the site builds. You can read about that process in the Creating a Source Plugin guide.

> *This process of fetching data at build time and creating pages from the data is covered in more depth in the tutorial as well as the docs for creating pages from data programmatically.*

Source data to be queried at build time

To source data using an existing source plugin you need to install a plugin and add it to your Gatsby config file. To use `gatsby-source-graphql`, first install it:

```
npm install gatsby-source-graphql
```

Then, add the plugin to your `gatsby-config.js` file:

```
module.exports = {
  plugins: [
    {
      resolve: `gatsby-source-graphql`,
      options: {
        typeName: `GitHub`,
        fieldName: `github`,
        url: `https://api.github.com/graphql`, //highlight-line
        headers: {
          Authorization: `Bearer your-github-token`,
        },
      },
    },
  ],
}
```

> *Because the GitHub GraphQL API requires you to be authenticated to make requests, you need to create a token and replace the "your-github-token" text in the header for Authorization with that token. You can secure your key using environment variables if you're pushing code to a public repository.*

Alternately, if you want to source data yourself you can use APIs Gatsby provides. Source plugins take advantage of the `sourceNodes` API and the `createNode` action provided by Gatsby to make your data queryable during the build process. If you want to source data yourself you can add a section of code like this using the `createNode` API to add a node to your data layer manually:

```
const fetch = (...args) =>
  import(`node-fetch`).then(({ default: fetch }) => fetch(...args))

exports.sourceNodes = async ({
  actions: { createNode },
  createContentDigest,
}) => {
  // get data from GitHub API at build time
  const result = await fetch(`https://api.github.com/repos/gatsbyjs/gatsby`)
  const resultData = await result.json()
  // create node for build time data example in the docs
  createNode({
    // nameWithOwner and url are arbitrary fields from the data
    nameWithOwner: resultData.full_name,
    url: resultData.html_url,
    // required fields
    id: `example-build-time-data`,
    parent: null,
    children: [],
    internal: {
      type: `Example`,
      contentDigest: createContentDigest(resultData),
    },
```

```
  })
}
```

It is important to note that node fields can indeed be arbitrary. The `nameWithOwner` and `url` fields from above are examples of this.

This node created manually could be retrieved with a query like this:

```
query {
  example {
    nameWithOwner
    url
  }
}
```

Writing a query to gather the static data needed for a page

With the source plugin installed and added to your config, you can write a static query that will retrieve the necessary data from Gatsby's data layer while building the site.

```
import React from "react"
import { graphql, useStaticQuery } from "gatsby" // highlight-line

const IndexPage = () => {
  // highlight-start
  const gatsbyRepoData = useStaticQuery(graphql`
    query {
      github {
        repository(name: "gatsby", owner: "gatsbyjs") {
          id
          nameWithOwner
          url
        }
      }
    }
  `)
  // highlight-end

  return (
    <section>
      <p>
        Build Time Data: Gatsby repo{` `}
        <a href={gatsbyRepoData.github.repository.url}>
          {gatsbyRepoData.github.repository.nameWithOwner} // highlight-line
        </a>
      </p>
    </section>
  )
}

export default IndexPage
```

> *This data is gathered at build time and written to a JSON file. As the build continues, the code is rewritten behind the scenes to dynamically import the JSON file and set* `gatsbyRepoData` *equal to the contents of the JSON file instead of the call to* `useStaticQuery`*. You can read more about this process in the Gatsby internals section on Normal vs Static Queries*

In the example site, the build data is shown in the line:

> *Gatsby repo: gatsbyjs/gatsby*

The linked URL and repository name are fetched at build time; if the name of the repository changed and the site were rebuilt, it would update.

Fetching data at client-side runtime

For fetching data at runtime in the browser, you can use any method to retrieve data that you would use in a regular React app.

Retrieving data with the `fetch` API

The `fetch` API is a modern implementation of the older, well-supported `XMLHttpRequest` (also known as AJAX).

With the `useState` and `useEffect` hooks from React, you can query for the data once when the page loads, and save the data returned to a variable called `starsCount`. Every time the page is refreshed, `fetch` will go to the GitHub API to retrieve the most up-to-date version of the data.

```
import React, { useState, useEffect } from "react" // highlight-line
import { graphql, useStaticQuery } from "gatsby"

const IndexPage = () => {
  // Build Time Data Fetching
  const gatsbyRepoData = useStaticQuery(graphql`
    query {
      github {
        repository(name: "gatsby", owner: "gatsbyjs") {
          id
          nameWithOwner
          url
        }
      }
    }
  `)
  // Client-side Runtime Data Fetching
  // highlight-start
  const [starsCount, setStarsCount] = useState(0)
  useEffect(() => {
    // get data from GitHub api
    fetch(`https://api.github.com/repos/gatsbyjs/gatsby`)
      .then(response => response.json()) // parse JSON from request
      .then(resultData => {
        setStarsCount(resultData.stargazers_count)
      }) // set data for the number of stars
  }, [])
```

```
  // highlight-end

  return (
    <section>
      <p>
        Build Time Data: Gatsby repo{` `}
        <a href={gatsbyRepoData.github.repository.url}>
          {gatsbyRepoData.github.repository.nameWithOwner}
        </a>
      </p>
      <p>Runtime Data: Star count for the Gatsby repo {starsCount}</p> // highlight-
line
    </section>
  )
}

export default IndexPage
```

In the code above, both the build time and runtime data are rendered from the same page component. The build time data has the advantage of being loaded before the user ever gets to the page. When the site loads in the browser, the runtime section in the `useEffect` hook will gather its data and render it as well.

In the example site, runtime data is shown in the line:

> *Star count for the Gatsby repo: [current star count]*

The repo's star count is fetched at runtime; if you refresh the page, this number will likely update.

Other resources

You may be interested in other projects (both used in production and proof-of-concepts) that illustrate this usage:

- Live example of the code used in this guide
- Gatsby store: with static product pages at build time and client-side interactions for e-commerce features
- Gatsby mail: a client-side email application
- Example repo fetching data using Apollo