This guide will walk you through the process of using Gatsby with WordPress and WPGraphQL.

WordPress is a free and open-source content management system (CMS). Let's say you have a site built with WordPress and you want to pull the existing data into your static Gatsby site. You can do that with gatsby-source-wordpress. Let's begin!

Note: this guide uses the `gatsby-starter-default` to provide you with the knowledge necessary to start working with WordPress but if you get stuck at some point of the guide feel free to use this example to gain extra insights.

Setup

Quick start

This guide assumes that you have a Gatsby project set up along with a WordPress instance with the appropriate plugins. If you need to set up a Gatsby project, head to the Quick Start guide, then come back. For information on configuring your WordPress instance, checkout the docs before continuing.

gatsby-config.js

Essentially the Gatsby home base. The two things defined here initially (in the starter) are `siteMetadata` and `plugins` you can add to enable new functionalities on your site.

```
module.exports = {
  siteMetadata: {
    title: "Gatsby Default Starter",
  },
  plugins: ["gatsby-plugin-react-helmet"],
  ...
}
```

Gatsby Plugin: gatsby-source-wordpress

Now that you have some understanding of project structure let's add fetching WordPress data functionality. There's a plugin for that. `gatsby-source-wordpress` is Gatsby's plugin for sourcing data from WordPress sites using the WPGraphQL API. You can install it by running the following command:

```
npm install gatsby-source-wordpress
```

Configuring the plugin

In `gatsby-config.js`, add your WordPress site's url, all other config is optional but recommended.

```
module.exports = {
  ...
  plugins: [
    ...,
    {
      resolve: `gatsby-source-wordpress`,
      options: {
        url:
        // allows a fallback url if WPGRAPHQL_URL is not set in the env, this may be
```

```
a local or remote WP instance.
          process.env.WPGRAPHQL_URL ||
          `https://localhost/graphql`,
        schema: {
          //Prefixes all WP Types with "Wp" so "Post and allPost" become "WpPost and
allWpPost".
          typePrefix: `Wp`,
        },
        develop: {
          //caches media files outside of Gatsby's default cache an thus allows them
to persist through a cache reset.
          hardCacheMediaFiles: true,
        },
        type: {
          Post: {
            limit:
              process.env.NODE_ENV === `development`
                ? // Lets just pull 50 posts in development to make it easy on
ourselves (aka. faster).
                  50
                : // and we don't actually need more than 5000 in production for
this particular site
                  5000,
          },
        },
      },
    },
  ]
}
```

Note: If your config varies from what is shown above, for instance, if you are securing your WordPress instance with Basic Auth, please refer to the plugin docs for more information on how to set up other options required for your use case.

Using WordPress data

Once your source plugin is pulling data, you can construct your site pages by implementing the `createPages` API in `gatsby-node.js`. When this is called, your data has already been fetched and is available to query with GraphQL. Gatsby uses GraphQL at build time; Your source plugin (in this case, `gatsby-source-wordpress`) fetches your data, and Gatsby uses that data to "automatically *infer* a GraphQL schema" that you can query against.

The `createPages` API exposes the `graphql` function:

> *The GraphQL function allows us to run arbitrary queries against the local WordPress GraphQL schema... like the site has a built-in database constructed from the fetched data that you can run queries against. (Source)*

You can use the `gatsby-node.js` from the plugin demo to get started. For the purpose of this guide, the code to construct posts works out of the box. It queries your local WordPress GraphQL schema for all Posts, iterates through each Post node and constructs a static page for each, based on the defined template.

For example:

```
const path = require(`path`)
const { slash } = require(`gatsby-core-utils`)

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

  // query content for WordPress posts
  const {
    data: {
      allWpPost: { nodes: allPosts },
    },
  } = await graphql(`
    query {
      allWpPost {
        nodes {
          id
          uri
        }
      }
    }
  `)

  const postTemplate = path.resolve(`./src/templates/post.js`)

  allPosts.forEach(post => {
    createPage({
      // will be the url for the page
      path: post.uri,
      // specify the component template of your choice
      component: slash(postTemplate),
      // In the ^template's GraphQL query, 'id' will be available
      // as a GraphQL variable to query for this post's data.
      context: {
        id: post.id,
      },
    })
  })
}
```

After fetching data from WordPress via the query, all posts are iterated over, calling `createPage` for each one.

A Gatsby page is defined as "a site page with a pathname, a template component, and an *optional* GraphQL query and Layout component."

When you restart your server with the `gatsby develop` command, you'll be able to navigate to the new pages created for each of your posts at their respective paths.

In the GraphiQL IDE at `http://localhost:8000/__graphql` you should now see queryable fields for `allWpPosts` in the docs or explorer sidebar.

Wrapping up

This was a very basic example meant to help you understand how you can fetch data from WordPress and use it with Gatsby. As the guide mentioned already, if you got stuck, you can have a look at example repo, which is a working example created to support this guide.

Other resources

- WordPress integration announcement
- Video Demo of WordPress integration
- More Gatsby blog posts about using Gatsby + WordPress