The file `gatsby-config.js` / `gatsby-config.ts` defines your site's metadata, plugins, and other general configuration. This file should be in the root of your Gatsby site. You can author the file in JavaScript or TypeScript.

If you created a Gatsby site with the `npm init gatsby` command, there should already be a sample configuration file in your site's directory. *Note: There are many sample configs which may be helpful to reference in the different Gatsby Example Websites.*

Set up the configuration file

The configuration file should export a JavaScript object. Within this object, you can define several different configuration options.

```
module.exports = {
  //configuration object
}
```

An example `gatsby-config.js` file could look like this:

```
module.exports = {
  siteMetadata: {
    title: `Gatsby`,
  },
  plugins: [
    `gatsby-transform-plugin`,
    {
      resolve: `gatsby-plugin-name`,
      options: {
        optionA: true,
        optionB: `Another option`,
      },
    },
  ],
}
```

The TypeScript and Gatsby documentation shows how to set up a configuration file in TypeScript.

Configuration options

Options available to set within `gatsby-config.js` include:

1. siteMetadata (object)
2. plugins (array)
3. flags (object)
4. pathPrefix (string)
5. trailingSlash (string)
6. polyfill (boolean)
7. mapping (object)
8. proxy (object)
9. developMiddleware (function)
10. jsxRuntime (string)

11. jsxImportSource (string)

siteMetadata

When you want to reuse common pieces of data across the site (for example, your site title), you can store that data in `siteMetadata`:

```
module.exports = {
  siteMetadata: {
    title: `Gatsby`,
    siteUrl: `https://www.gatsbyjs.com`,
    description: `Blazing fast modern site generator for React`,
  },
}
```

This way you can store it in one place, and pull it whenever you need it. If you ever need to update the info, you only have to change it here.

See a full description and sample usage in Gatsby Tutorial Part Four.

plugins

Plugins are Node.js packages that implement Gatsby APIs. The config file accepts an array of plugins. Some plugins may need only to be listed by name, while others may take options (see the docs for individual plugins).

Installing a plugin using a package manager like `npm` **does not** enable it in your Gatsby site. To finish adding a plugin, make sure your `gatsby-config.js` file has a `plugins` array so you can include a space for the plugins needed to build your site:

```
module.exports = {
  plugins: [
    //plugins go here
  ],
}
```

When adding multiple plugins, they should be separated by commas in the `plugins` array to support valid JavaScript syntax.

Plugins without options

If a plugin does not require any options, you can add its name as a string to the `plugins` array:

```
module.exports = {
  plugins: [`gatsby-plugin-name`],
}
```

Plugins with options

Many plugins have optional or required options to configure them. Instead of adding a name string to the `plugins` array, add an object with its name and options. Most plugins show examples in their `README` file or page in the Gatsby plugin library.

Here's an example showing how to write an object with keys to `resolve` the plugin name and an `options` object with any applicable settings:

```
module.exports = {
  plugins: [
    {
      resolve: `gatsby-plugin-name`,
      options: {
        optionA: true,
        optionB: `Another option`,
      },
    },
  ],
}
```

Mixed plugins

You can add plugins with and without options in the same array. Your site's config file could look like this:

```
module.exports = {
  plugins: [
    `gatsby-transform-plugin`,
    {
      resolve: `gatsby-plugin-name`,
      options: {
        optionA: true,
        optionB: `Another option`,
      },
    },
  ],
}
```

See more about Plugins for more on utilizing plugins, and to see available official and community plugins.

flags

Flags let sites enable experimental or upcoming changes that are still in testing or waiting for the next major release.

Go here to see a list of the current flags.

```
module.exports = {
  flags: {
    DEV_SSR: true,
  },
}
```

pathPrefix

It's common for sites to be hosted somewhere other than the root of their domain. Say you have a Gatsby site at `example.com/blog/`. In this case, you would need a prefix (`/blog`) added to all paths on the site.

```
module.exports = {
  pathPrefix: `/blog`,
}
```

See more about Adding a Path Prefix.

trailingSlash

Configures the creation of URLs for pages, and whether to remove, append, or ignore trailing slashes.

- `always` : Always add trailing slashes to each URL, e.g. `/x` to `/x/` .
- `never` : Remove all trailing slashes on each URL, e.g. `/x/` to `/x` .
- `ignore` : Don't automatically modify the URL

The default setting for this option is `legacy` in order to preserve existing behavior for current users. In Gatsby v5 the default mode will be `always` . Gatsby Cloud automatically handles and supports the `trailingSlash` option. Alternate hosting providers (or if you're managing this on your own) should follow the "Redirects, and expected behavior from the hosting provider" section on the initial RFC.

polyfill

Gatsby uses the ES6 Promise API. Because some browsers don't support this, Gatsby includes a Promise polyfill by default.

If you'd like to provide your own Promise polyfill, you can set `polyfill` to false.

```
module.exports = {
  polyfill: false,
}
```

See more about Browser Support in Gatsby.

Mapping node types

Please note: We strongly recommend using the `@link` GraphQL directive instead. It supports more use cases and will be the preferred method for foreign-keys in the future.

Gatsby includes a feature that lets you create "mappings" between node types.

For instance, imagine you have a multi-author markdown blog where you want to "link" from each blog post to the author information stored in a YAML file named `author.yaml` :

```
---
title: A blog post
author: Kyle Mathews
---

A treatise on the efficacy of bezoar for treating agricultural pesticide poisoning.
```

```
- name: Kyle Mathews
  bio: Founder @ GatsbyJS. Likes tech, reading/writing, founding things. Blogs at
bricolage.io.
  twitter: "@kylemathews"
```

You can map between the `author` field in `frontmatter` to the `name` in the `author.yaml` objects by adding to your `gatsby-config.js`:

```
module.exports = {
  plugins: [...],
  mapping: {
    "MarkdownRemark.frontmatter.author": `AuthorYaml.name`,
  },
}
```

You may need to install the appropriate file transformer (in this case YAML) and set up gatsby-source-filesystem properly for Gatsby to pick up the mapping files. This applies to other file types later mentioned in this segment as well.

Gatsby then uses this mapping when creating the GraphQL schema to enable you to query data from both sources:

```
query ($slug: String!) {
  markdownRemark(fields: { slug: { eq: $slug } }) {
    html
    fields {
      slug
    }
    frontmatter {
      title
      author {
        # This now links to the author object
        name
        bio
        twitter
      }
    }
  }
}
```

Mapping can also be used to map an array of ids to any other collection of data. For example, if you have two JSON files `experience.json` and `tech.json` as follows:

```
[
  {
    "id": "companyA",
    "company": "Company A",
    "position": "Unicorn Developer",
    "from": "Dec 2016",
    "to": "Present",
```

```
    "items": [
      {
        "label": "Responsibility",
        "description": "Being an unicorn"
      },
      {
        "label": "Hands on",
        "tech": ["REACT", "NODE"]
      }
    ]
  }
]
```

```
[
  {
    "name": "REACT",
    "icon": "facebook",
    "color": "teal",
    "label": "React"
  },
  {
    "name": "NODE",
    "icon": "server",
    "color": "green",
    "label": "NodeJS"
  }
]
```

And then add the following rule to your `gatsby-config.js`:

```
module.exports = {
  plugins: [...],
  mapping: {
    'ExperienceJson.items.tech': `TechJson.name`
  },
}
```

You can query the `tech` object via the referred items in `experience`:

```
query {
  allExperienceJson {
    edges {
      node {
        company
        position
        from
        to
        items {
          label
          description
```

```
          link
          tech {
            label
            color
            icon
          }
        }
      }
    }
  }
}
```

Mapping also works between Markdown files. For example, instead of having all authors in a YAML file, you could have info about each author in a separate Markdown file:

```
---
author_id: Kyle Mathews
twitter: "@kylemathews"
---

Founder @ GatsbyJS. Likes tech, reading/writing, founding things. Blogs at
bricolage.io.
```

And then add the following rule to your `gatsby-config.js`:

```
module.exports = {
  plugins: [...],
  mapping: {
    'MarkdownRemark.frontmatter.author': `MarkdownRemark.frontmatter.author_id`
  },
}
```

Similarly to YAML and JSON files, mapping between Markdown files can also be used to map an array of ids.

Proxy

Setting the proxy config option will tell the develop server to proxy any unknown requests to your specified server. For example:

```
module.exports = {
  proxy: {
    prefix: "/api",
    url: "http://examplesite.com/api/",
  },
}
```

See more about Proxying API Requests in Develop.

Advanced proxying with `developMiddleware`

See more about adding develop middleware.

jsxRuntime

Setting to `automatic` allows the use of JSX without having to import React. More information can be found on the Introducing the new JSX Transform blog post.

```
module.exports = {
  jsxRuntime: "automatic",
}
```

jsxImportSource

When `jsxRuntime` is set you can choose which package React should use as underlying JSX transformer with `jsxImportSource`. For example you can set it to `@emotion/react` so by default `@emotion/react` is used instead of the `react` package.

```
module.exports = {
  jsxRuntime: "automatic",
  jsxImportSource: "@emotion/react",
}
```

Please note: For now you'll also need to set this configuration inside `babel-preset-gatsby`, see its jsxImportSource documentation.