gatsby-source-contentful

> Source plugin for pulling content types, entries, and assets into Gatsby from Contentful spaces. It creates links between entry types and asset so they can be queried in Gatsby using GraphQL.
>
> An example site for using this plugin is at https://using-contentful.gatsbyjs.org/

Table of contents

Install

```
npm install gatsby-source-contentful
```

Setup Instructions

To get setup quickly with a new site and have Gatsby Cloud do the heavy lifting, deploy a new Gatsby Contentful site with just a few clicks on gatsbyjs.com.

For more detailed instructions on manually configuring your Gatsby Contentful site for production builds and Preview builds visit the Gatsby Cloud knowledge-base.

How to use

First, you need a way to pass environment variables to the build process, so secrets and other secured data aren't committed to source control. We recommend using `dotenv` which will then expose environment variables. Read more about `dotenv` and using environment variables here. Then we can *use* these environment variables and configure our plugin.

Restrictions and limitations

This plugin has several limitations, please be aware of these:

1. At the moment, fields that do not have at least one populated instance will not be created in the GraphQL schema. This can break your site when field values get removed. You may workaround with an extra content entry with all fields filled out.
2. When using reference fields, be aware that this source plugin will automatically create the reverse reference. You do not need to create references on both content types.
3. When working with environments, your access token has to have access to your desired enviornment and the `master` environment.
4. Using the preview functionallity might result in broken content over time, as syncing data on preview is not officially supported by Contentful. Make sure to regulary clean your cache when using Contentfuls preview API.
5. The following content type names are not allowed: `entity`, `reference`
6. The following field names are restricted and will be prefixed: `children`, `contentful_id`, `fields`, `id`, `internal`, `parent`,

Using Delivery API

```
// In your gatsby-config.js
module.exports = {
  plugins: [
    {
      resolve: `gatsby-source-contentful`,
      options: {
        spaceId: `your_space_id`,
        // Learn about environment variables: https://gatsby.dev/env-vars
        accessToken: process.env.CONTENTFUL_ACCESS_TOKEN,
      },
```

```
    },
  ],
}
```

Using Preview API

```
// In your gatsby-config.js
module.exports = {
  plugins: [
    {
      resolve: `gatsby-source-contentful`,
      options: {
        spaceId: `your_space_id`,
        // Learn about environment variables: https://gatsby.dev/env-vars
        accessToken: process.env.CONTENTFUL_ACCESS_TOKEN,
        host: `preview.contentful.com`,
      },
    },
  ],
}
```

Offline

If you don't have internet connection you can add `export GATSBY_CONTENTFUL_OFFLINE=true` to tell the plugin to fallback to the cached data, if there is any.

Configuration options

`spaceId` [string][required]

Contentful space ID

`accessToken` [string][required]

Contentful delivery API key, when using the Preview API use your Preview API key

`host` [string][optional] [default: `'cdn.contentful.com'`]

The base host for all the API requests, by default it's `'cdn.contentful.com'`, if you want to use the Preview API set it to `'preview.contentful.com'`. You can use your own host for debugging/testing purposes as long as you respect the same Contentful JSON structure.

`environment` [string][optional] [default: 'master']

The environment to pull the content from, for more info on environments check out this Guide.

`downloadLocal` [boolean][optional] [default: `false`]

Downloads and caches `ContentfulAsset`'s to the local filesystem. Allows you to query a `ContentfulAsset`'s `localFile` field, which is not linked to Contentful's CDN. See Download assets for static distribution for more information on when and how to use this feature.

`localeFilter` [function][optional] [default: `() => true`]

Possibility to limit how many locales/nodes are created in GraphQL. This can limit the memory usage by reducing the amount of nodes created. Useful if you have a large space in Contentful and only want to get the data from one selected locale.

For example, to filter locales on only germany `localeFilter: locale => locale.code === 'de-DE'`

List of locales and their codes can be found in Contentful app -> Settings -> Locales

`forceFullSync` [boolean][optional] [default: `false`]

Prevents the use of sync tokens when accessing the Contentful API.

`proxy` [object][optional] [default: `undefined`]

Axios proxy configuration. See the axios request config documentation for further information about the supported values.

`useNameForId` [boolean][optional] [default: `true`]

Use the content's `name` when generating the GraphQL schema e.g. a content type called `[Component] Navigation bar` will be named `contentfulComponentNavigationBar`.

When set to `false`, the content's internal ID will be used instead e.g. a content type with the ID `navigationBar` will be called `contentfulNavigationBar`.

Using the ID is a much more stable property to work with as it will change less often. However, in some scenarios, content types' IDs will be auto-generated (e.g. when creating a new content type without specifying an ID) which means the name in the GraphQL schema will be something like `contentfulC6XwpTaSiiI2Ak2Ww0oi6qa`. This won't change and will still function perfectly as a valid field name but it is obviously pretty ugly to work with.

If you are confident your content types will have natural-language IDs (e.g. `blogPost`), then you should set this option to `false`. If you are unable to ensure this, then you should leave this option set to `true` (the default).

`pageLimit` [number][optional] [default: `100`]

Number of entries to retrieve from Contentful at a time. Due to some technical limitations, the response payload should not be greater than 7MB when pulling content from Contentful. If you encounter this issue you can set this param to a lower number than 100, e.g `50`.

`assetDownloadWorkers` [number][optional] [default: `50`]

Number of workers to use when downloading Contentful assets. Due to technical limitations, opening too many concurrent requests can cause stalled downloads. If you encounter this issue you can set this param to a lower number than 50, e.g 25.

`contentfulClientConfig` [object][optional] [default: `{}`]

Additional config which will get passed to Contentfuls JS SDK.

Use this with caution, you might override values this plugin does set for you to connect to Contentful.

`enableTags` [boolean][optional] [default: `false`]

Enable the new tags feature. This will disallow the content type name `tags` till the next major version of this plugin.

Learn how to use them at the Contentful Tags section.

How to query for nodes

Two standard node types are available from Contentful: `Asset` and `ContentType`.

`Asset` nodes will be created in your site's GraphQL schema under `contentfulAsset` and `allContentfulAsset`.

`ContentType` nodes are a little different - their exact name depends on what you called them in your Contentful data models. The nodes will be created in your site's GraphQL schema under `contentful${entryTypeName}` and `allContentful${entryTypeName}`.

In all cases querying for nodes like `contentfulX` will return a single node, and nodes like `allContentfulX` will return all nodes of that type.

Query for all nodes

You might query for **all** of a type of node:

```
{
  allContentfulAsset {
    nodes {
      contentful_id
      title
      description
    }
  }
}
```

You might do this in your `gatsby-node.js` using Gatsby's `createPages` Node API.

Query for a single node

To query for a single `image` asset with the title `'foo'` and a width of 1600px:

```
export const assetQuery = graphql`
  {
    contentfulAsset(title: { eq: "foo" }) {
      contentful_id
      title
      description
      file {
        fileName
        url
        contentType
        details {
          size
          image {
            height
            width
          }
        }
      }
    }
  }
`
```

To query for a single `CaseStudy` node with the short text properties `title` and `subtitle`:

```
  {
    contentfulCaseStudy(filter: { title: { eq: 'bar' } })  {
      title
      subtitle
    }
  }
```

You might query for a **single** node inside a component in your `src/components` folder, using Gatsby's `StaticQuery` component.

A note about LongText fields On Contentful, a "Long text" field uses Markdown by default. The field is exposed as an object, while the raw Markdown is exposed as a child node.

```
{
  contentfulCaseStudy {
    body {
      body
    }
```

```
  }
}
```

Unless the text is Markdown-free, you cannot use the returned value directly. In order to handle the Markdown content, you must use a transformer plugin such as `gatsby-transformer-remark`. The transformer will create a `childMarkdownRemark` on the "Long text" field and expose the generated html as a child node:

```
{
  contentfulCaseStudy {
    body {
      childMarkdownRemark {
        html
      }
    }
  }
}
```

You can then insert the returned HTML inline in your JSX:

```
<div
  className="body"
  dangerouslySetInnerHTML={{
    __html: data.contentfulCaseStudy.body.childMarkdownRemark.html,
  }}
/>
```

Duplicated entries When Contentful pulls the data, all localizations will be pulled. Therefore, if you have a localization active, it will duplicate the entries. Narrow the search by filtering the query with `node_locale` filter:

```
{
  allContentfulCaseStudy(filter: { node_locale: { eq: "en-US" } }) {
    edges {
      node {
        id
        slug
        title
        subtitle
        body {
          body
        }
      }
    }
  }
}
```

Query for Assets in ContentType nodes

More typically your `Asset` nodes will be mixed inside of your `ContentType` nodes, so you'll query them together. All the same formatting rules for `Asset` and `ContentType` nodes apply.

To get **all** the `CaseStudy` nodes with `ShortText` fields `id`, `slug`, `title`, `subtitle`, `LongText` field `body` and `heroImage` asset field, we use `allContentful${entryTypeName}` to return all instances of that `ContentType`:

```
{
  allContentfulCaseStudy {
    edges {
      node {
        id
        slug
        title
        subtitle
        body {
          body
        }
        heroImage {
          title
          description
          gatsbyImageData(layout: CONSTRAINED)
          # Further below in this doc you can learn how to use these response images
        }
      }
    }
  }
}
```

More on Queries with Contentful and Gatsby

It is strongly recommended that you take a look at how data flows in a real Contentful and Gatsby application to fully understand how the queries, Node.js functions and React components all come together. Check out the example site at using-contentful.gatsbyjs.org.

Displaying responsive image with gatsby-plugin-image

To use it:

1. Install the required plugins:

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

2. Add the plugins to your `gatsby-config.js`:

```
module.exports = {
  plugins: [
    `gatsby-plugin-sharp`,
    `gatsby-plugin-image`,
    // ...etc
  ],
}
```

3. You can then query for image data using the new `gatsbyImageData` resolver:

```
{
  allContentfulBlogPost {
    nodes {
      heroImage {
        gatsbyImageData(layout: FULL_WIDTH)
      }
    }
  }
}
```

4. Your query will return a dynamic image. Check the documentation of gatsby-plugin-image to learn how to render it on your website.

Check the Reference Guide of gatsby-plugin-image to get a deeper insight on how this works.

Contentful Tags

You need to set the `enableTags` flag to `true` to use this new feature.

List available tags

This example lists all available tags. The sorting is optional.

```
query TagsQuery {
  allContentfulTag(sort: { fields: contentful_id }) {
    nodes {
      name
      contentful_id
    }
  }
}
```

Filter content by tags

This filters content entries that are tagged with the `numberInteger` tag.

```
query FilterByTagsQuery {
  allContentfulNumber(
    sort: { fields: contentful_id }
    filter: {
      metadata: {
        tags: { elemMatch: { contentful_id: { eq: "numberInteger" } } }
      }
    }
  ) {
    nodes {
      title
      integer
    }
  }
}
```

Contentful Rich Text

Rich Text feature is supported in this source plugin, you can use the following query to get the JSON output:

Note: In our example Content Model the field containing the Rich Text data is called `bodyRichText`. Make sure to use your field name within the Query instead of `bodyRichText`

Query Rich Text content and references

```
query pageQuery($id: String!) {
  contentfulBlogPost(id: { eq: $id }) {
    title
    slug
    # This is the rich text field, the name depends on your field configuration in Contentf
    bodyRichText {
      raw
      references {
        ... on ContentfulAsset {
          # You'll need to query contentful_id in each reference
          contentful_id
          __typename
          fixed(width: 1600) {
            width
            height
            src
            srcSet
          }
        }
        ... on ContentfulBlogPost {
```

```
          contentful_id
          __typename
          title
          slug
        }
      }
    }
  }
}
```

Rendering

```
import { BLOCKS, MARKS } from "@contentful/rich-text-types"
import { renderRichText } from "gatsby-source-contentful/rich-text"

const Bold = ({ children }) => <span className="bold">{children}</span>
const Text = ({ children }) => <p className="align-center">{children}</p>

const options = {
  renderMark: {
    [MARKS.BOLD]: text => <Bold>{text}</Bold>,
  },
  renderNode: {
    [BLOCKS.PARAGRAPH]: (node, children) => <Text>{children}</Text>,
    [BLOCKS.EMBEDDED_ASSET]: node => {
      return (
        <>
          <h2>Embedded Asset</h2>
          <pre>
            <code>{JSON.stringify(node, null, 2)}</code>
          </pre>
        </>
      )
    },
  },
}

function BlogPostTemplate({ data }) {
  const { bodyRichText } = data.contentfulBlogPost

  return <div>{bodyRichText && renderRichText(bodyRichText, options)}</div>
}
```

Note: The `contentful_id` field must be queried on rich-text references in order for the `renderNode` to receive the correct data.

Embedding an image in a Rich Text field

Import

```
import { renderRichText } from "gatsby-source-contentful/rich-text"
```

GraphQL

```
mainContent {
  raw
  references {
    ... on ContentfulAsset {
      contentful_id
      __typename
      gatsbyImageData
    }
  }
}
```

Options

```
const options = {
  renderNode: {
    "embedded-asset-block": node => {
      const { gatsbyImageData } = node.data.target
      if (!gatsbyImageData) {
        // asset is not an image
        return null
      }
      return <GatsbyImage image={image} />
    },
  },
}
```

Render

```
<article>
  {blogPost.mainContent && renderRichText(blogPost.mainContent, options)}
</article>
```

Check out the examples at @contentful/rich-text-react-renderer.

Download assets for static distribution

When you set `downloadLocal: true` in your config, the plugin will download and cache Contentful assets to the local filesystem.

There are two main reasons you might want to use this option:

- Avoiding the extra network handshake required to the Contentful CDN for images hosted there

- Gain more control over how images are processed or rendered (ie, providing AVIF with gatsby-plugin-image)

The default setting for this feature is `false`, as there are certain tradeoffs for using this feature:

- All assets in your Contentful space will be downloaded during builds
- Your build times and build output size will increase
- Bandwidth going through your CDN will increase (since images are no longer being served from the Contentful CDN)
- You will have to change how you query images and which image fragments you use.

Enable the feature with the `downloadLocal: true` option.

```
// In your gatsby-config.js
module.exports = {
  plugins: [
    {
      resolve: `gatsby-source-contentful`,
      options: {
        spaceId: `your_space_id`,
        // Learn about environment variables: https://gatsby.app/env-vars
        accessToken: process.env.CONTENTFUL_ACCESS_TOKEN,
        downloadLocal: true,
      },
    },
  ],
}
```

Updating Queries for downloadLocal

When using the `downloadLocal` setting, you'll need to update your codebase to be working with the locally downloaded images, rather than the default Contentful Nodes. Query a `ContentfulAsset`'s `localFile` field in GraphQL to gain access to the common fields of the `gatsby-source-filesystem` `File` node. This is not a Contentful node, so usage for `gatsby-plugin-image` is different:

```
query MyQuery {
  # Example is for a `ContentType` with a `ContentfulAsset` field
  # You could also query an asset directly via
  # `allContentfulAsset { edges{ node { } } }`
  # or `contentfulAsset(contentful_id: { eq: "contentful_id here" } ) { }`
  contentfulMyContentType {
    myContentfulAssetField {
      # Query for locally stored file(e.g. An image) - `File` node
      localFile {
        # Use `gatsby-plugin-image` to create the image data
```

```
        childImageSharp {
          gatsbyImageData(formats: AVIF)
        }
      }
    }
  }
}
```

Note: This feature downloads any file from a `ContentfulAsset` node that `gatsby-source-contentful` provides. They are all copied over from `./cache/gatsby-source-filesystem/` to the sites build location `./public/static/`.

For any troubleshooting related to this feature, first try clearing your `./cache/` directory. `gatsby-source-contentful` will acquire fresh data, and all `ContentfulAsset`s will be downloaded and cached again.

Sourcing From Multiple Contentful Spaces

To source from multiple Contentful environments/spaces, add another configuration for `gatsby-source-contentful` in `gatsby-config.js`:

```
// In your gatsby-config.js
module.exports = {
  plugins: [
    {
      resolve: `gatsby-source-contentful`,
      options: {
        spaceId: `your_space_id`,
        accessToken: process.env.CONTENTFUL_ACCESS_TOKEN,
      },
    },
    {
      resolve: `gatsby-source-contentful`,
      options: {
        spaceId: `your_second_space_id`,
        accessToken: process.env.SECONDARY_CONTENTFUL_ACCESS_TOKEN,
      },
    },
  ],
}
```