What is Sanity.io?

Sanity is a hosted backend for structured content that comes with an open source editor built in React. It has powerful real-time APIs for both reading and writing data.

You can use Sanity as a headless CMS that lets your authors work in a user-friendly environment, or as a pure data backend for your apps. We make it easier for you to reuse content across multiple websites, apps, print, voice assistants, and other channels.

Getting started

Begin with setting up a Gatsby project. If you want to start from scratch, the Quick Start guide is a good place to begin. Come back to this guide when you're set up.

This guide will cover how configure and use the `gatsby-source-sanity` plugin.

Basic usage

```
npm install gatsby-source-sanity
```

```
module.exports = {
  plugins: [
    {
      resolve: "gatsby-source-sanity",
      options: {
        projectId: "abc123",
        dataset: "blog",
      },
    },
  ],
}
```

At this point you can choose to (and probably should) set up a GraphQL API for your Sanity dataset, if you have not done so already. This will help the plugin in knowing which types and fields exists, so you can query for them even without them being present in any current documents.

Go through `http://localhost:8000/___graphql` after running `gatsby develop` to understand the created data. Create a new query and check available collections and fields by using the autocomplete (`CTRL + SPACE`).

Options

Options	Type	Default	Description
projectId	string		**[required]** Your Sanity project's ID
dataset	string		**[required]** The dataset to fetch from
token	string		Authentication token for fetching data from private datasets, or when using `overlayDrafts` Learn more

overlayDrafts	boolean	`false`	Set to `true` in order for drafts to replace their published version. By default, drafts will be skipped.
watchMode	boolean	`false`	Set to `true` to keep a listener open and update with the latest changes in realtime.

Missing fields

Getting errors such as these?

> *Cannot query field "allSanityBlogPost" Unknown field* `preamble` *on type* `BlogPost`

By deploying a GraphQL API for your dataset, we are able to introspect and figure out which schema types and fields are available and make them available to prevent this problem. Once the API is deployed it will be transparently applied. If you have deployed your API and are still seeing similar issues, remember that you have to redeploy the API if your schema changes.

Some background for this problem:

Gatsby cannot know about the types and fields without having documents of the given types that contain the fields you want to query. This is a known problem with Gatsby - luckily there is ongoing work to solve this issue, which will lead to much clearer schemas and less boilerplate.

Using images

Image fields will have the image URL available under the `field.asset.url` key, but you can also use gatsby-plugin-image for a smooth experience. It's a React component that enables responsive images and advanced image loading techniques. It works great with this source plugin, without requiring any additional build steps.

There are three types of responsive images supported; *constrained* (the default), *fixed*, and *full width*. To decide which one to use, ask yourself: "do I know the exact size this image will be?" If yes, you'll want to use *fixed*. If no and its width and/or height need to vary depending on the size of the screen, then you'll want to use *constrained* or *full width*. (For more information, refer to the `layout` option in the `gatsby-plugin-image` Reference Guide).

Constrained

```
import React from "react"
import { GatsbyImage } from "gatsby-plugin-image"

const Person = ({ data }) => (
  <article>
    <h2>{data.sanityPerson.name}</h2>
    <GatsbyImage image={data.sanityPerson.profileImage.asset.gatsbyImageData} />
  </article>
)

export default Person

export const query = graphql`
  query PersonQuery {
    sanityPerson {
      name
```

```
      profileImage {
        asset {
          gatsbyImageData(placeholder: BLURRED)
        }
      }
    }
  }
`
```

Fixed

```
import React from "react"
import { GatsbyImage } from "gatsby-plugin-image"

const Person = ({ data }) => (
  <article>
    <h2>{data.sanityPerson.name}</h2>
    <GatsbyImage image={data.sanityPerson.profileImage.asset.gatsbyImageData} />
  </article>
)

export default Person

export const query = graphql`
  query PersonQuery {
    sanityPerson {
      name
      profileImage {
        asset {
          gatsbyImageData(layout: FIXED, placeholder: BLURRED, width: 400)
        }
      }
    }
  }
`
```

Overlaying drafts

Sometimes you might be working on some new content that is not yet published, which you want to make sure looks alright within your Gatsby site. By setting the `overlayDrafts` setting to `true`, the draft versions will as the option says "overlay" the regular document. In terms of Gatsby nodes, it will *replace* the published document with the draft.

Keep in mind that drafts do not have to conform to any validation rules, so your frontend will usually want to double-check all nested properties before attempting to use them.

Watch mode

While developing, it can often be beneficial to get updates without having to manually restart the build process. By setting `watchMode` to true, this plugin will set up a listener which watches for changes. When it detects a change,

the document in question is updated in real-time and will be reflected immediately.

If you add an environment token and set `overlayDrafts` to true, each small change to the draft will immediately be applied.

Generating pages

Sanity does not have any concept of a "page", since it's built to be totally agnostic to how you want to present your content and in which medium, but since you're using Gatsby, you'll probably want some pages!

As with any Gatsby site, you'll want to create a `gatsby-node.js` in the root of your Gatsby site repository (if it doesn't already exist), and declare a `createPages` function. Within it, you'll use GraphQL to query for the data you need to build the pages.

For instance, if you have a `project` document type in Sanity that you want to generate pages for, you could do something along the lines of this:

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

  const result = await graphql(`
    {
      allSanityProject(filter: { slug: { current: { ne: null } } }) {
        edges {
          node {
            title
            description
            tags
            launchDate(format: "DD.MM.YYYY")
            slug {
              current
            }
            image {
              asset {
                url
              }
            }
          }
        }
      }
    }
  `)

  if (result.errors) {
    throw result.errors
  }

  const projects = result.data.allSanityProject.edges || []
  projects.forEach((edge, index) => {
    const path = `/project/${edge.node.slug.current}`

    createPage({
```

```
      path,
      component: require.resolve("./src/templates/project.js"),
      context: { slug: edge.node.slug.current },
    })
  })
}
```

The above query will fetch all projects that have a `slug.current` field set, and generate pages for them, available as `/project/<project-slug>`. It will use the template defined in `src/templates/project.js` as the basis for these pages.

Most Gatsby starters have some example of building pages, which you should be able to modify to your needs.

Remember to use the GraphiQL interface to help write the queries you need - it's usually running at `http://localhost:8000/___graphql` while running `gatsby develop`.

"Raw" fields

Arrays and object types at the root of documents will get an additional "raw JSON" representation in a field called `_raw<FieldName>`. For instance, a field named `body` will be mapped to `_rawBody`. It's important to note that this is only done for top-level nodes (documents).

Portable Text / Block Content

Rich text in Sanity is usually represented as Portable Text (previously known as "Block Content").

These data structures can be deep and a chore to query (specifying all the possible fields). As noted above, there is a "raw" alternative available for these fields which is usually what you'll want to use.

You can install block-content-to-react from npm and use it in your Gatsby project to serialize Portable Text. It lets you use your own React components to override defaults and render custom content types. Learn more about Portable Text in our documentation.

Using .env variables

If you don't want to attach your Sanity project's ID to the repo, you can store it in .env files by doing the following:

```
SANITY_PROJECT_ID = abc123
SANITY_DATASET = production
SANITY_TOKEN = my-super-secret-token
```

```
require("dotenv").config({
  path: `.env.${process.env.NODE_ENV}`,
})

module.exports = {
  plugins: [
    {
      resolve: "gatsby-source-sanity",
      options: {
        projectId: process.env.SANITY_PROJECT_ID,
```

```
        dataset: process.env.SANITY_DATASET,
        token: process.env.SANITY_TOKEN,
      },
    },
  ],
}
```

This example is based off Gatsby Docs' implementation.