Sourcing from Gentics Mesh

Using Gentics Mesh

Gentics Mesh is an open-source self-hosted headless CMS that supports GraphQL. Contents like posts, products, images and more are represented by elements called "nodes". These nodes can then be queried with the familiar GraphQL syntax.

The model of nodes can be defined within "schemas". The demo data contains schemas for vehicles, images and categories.

Getting started

In this guide you'll create a complete project capable of querying data from a Gentics Mesh server.

Start Mesh

Gentics Mesh is a self-hosted CMS. You can run it for example via Docker, which you will need installed to run the following command, setting up a Gentics server in a container on your computer:

```
docker run -p 8080:8080 -d gentics/mesh-demo
```

The demo Gentics server that gets setup has a UI that can be accessed on `http://localhost:8080/mesh-ui/` with default login (username: admin, password: admin). It allows you to view the content stored on the CMS and add new content.

Install the boilerplate

Now create a Gatsby site from the default Gatsby starter.

```
gatsby new gatsby-site https://github.com/gatsbyjs/gatsby-starter-default
```

Navigate inside of the project with `cd gatsby-site`.

Add the source plugin

Additionally, you need the `gatsby-source-graphql` plugin. Because Gentics Mesh uses GraphQL, you will take advantage of Gatsby's ability to stitch two

GraphQL APIs together, reducing the time required to transform content. There is no need to use a special gatsby-source-x-cms plugin, the GraphQL source plugin is all you need.

Install the plugin:

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

Configure the plugin

The last step required before you can query your data is to configure the `gatsby-source-graphql` plugin. Open `gatsby-config.js` and add the following object to the plugins array. This example uses the local Gentics Mesh Demo server that you have just started.

```
{
    resolve: "gatsby-source-graphql",
        options: {
        // The top level query type, can be anything you want!
        typeName: "MESH",
        // The field you'll query against, can also be anything you want.
        fieldName: "mesh",
        // Local Gentics Mesh server url
        url: "http://localhost:8080/api/v2/demo/graphql",
    },
},
```

If everything works correctly, you should now have your Gentics Mesh API added to the Gatsby source API!

Querying for content

From the root of your project, run the development environment with `gatsby develop`. Once the server has started, you should be able to open the following URL in your browser:

```
http://localhost:8000/___graphql
```

This will show you an interface where you can test your new content GraphQL API.

Try running this query:

```
query {
  mesh {
    nodes(filter: { schema: { is: vehicle } }, perPage: 2, page: 5) {
      elements {
        uuid
        ... on MESH_vehicle {
          fields {
            slug
            weight
            vehicleImage {
              path
            }
          }
        }
      }
    }
  }
}
```

Again, if everything is working properly, you should see a successful response in the shape of:

```
{
  "data": {
    "mesh": {
      "nodes": {
        "elements": [
          {
            "uuid": "a5d81285b4884df1981285b488adf1b5",
            "fields": {
              "slug": "embraer-legacy-600",
              "weight": 16000,
              "vehicleImage": {
                "path": "/images/embraer-legacy-600.jpg"
              }
            }
          },
          {
            "uuid": "adaf48da8c124049af48da8c12a0493e",
            "fields": {
              "slug": "delorean-dmc-12",
              "weight": 1230,
              "vehicleImage": {
                "path": "/images/delorean-dmc-12.jpg"
              }
            }
          }
        ]
      }
    }
  }
}
```

You can read more about how to use the GraphQL API in the Gentics Mesh docs.

Getting content on the page

For the purpose of this guide I've removed all the layout, SEO, link or other components in the Gatsby starter `index.js` file. Open the index file located at `src/pages/index.js` and replace the content with this code:

```
import React from "react"
import { StaticQuery } from "gatsby"

const IndexPage = () => (
  <StaticQuery
    query={graphql`
      query {
        mesh {
          nodes(filter: { schema: { is: vehicle } }) {
            elements {
              uuid
              ... on MESH_vehicle {
                fields {
                  name
                  weight
                }
              }
            }
          }
        }
      }
    `}
    render={data => (
      <div>
        <h1>Vehicles</h1>
        <ul>
          {data.mesh.nodes.elements.map(vehicle => {
            const { name, weight } = vehicle.fields
            return (
              <li>
                <strong>{name}: </strong>
                {weight} kg
              </li>
            )
          })}
        </ul>
      </div>
    )}
  />
)

export default IndexPage
```

Once saved you can access the page via `http://localhost:8000`

You now have:

1. Added the `StaticQuery` component to a page that allows you to fetch content from the GraphQL API.
2. Fetched some vehicles from the demo data, namely the name and weight.
3. Rendered the list in the StaticQuery's RenderProp called "render".

Hopefully this has helped you see what it's like to start working with Gentics Mesh and Gatsby.

Learn more

There is a lot more to learn on how Gentics Mesh API can help you build your site. One big advantage of Mesh is that you can use a tree structure for your contents. This allows for easy path generation of contents. Combined with Gatsby this allows you to quickly add different kinds of pages to your site. The template for each site can be selected by the schema (e.g. `vehicle` → `vehiclePage`, `category` → `categoryPage`.)

This workflow is also explained hands-on in this in-depth guide which shows how to build a basic vehicle inventory with Gatsby.