

Webサイト高速化のための 静的サイトジェネレーター 活用入門

GatsbyJSで実現する高速&実用的なサイト構築



副読本

microCMS対応ガイド

(Gatsby v4対応版)

Build blazing-fast websites with GatsbyJS



エビスコム 編著



本 PDF は下記書籍の副読本です。この PDF では、

サポート PDF (gatsby-v4.pdf)

Gatsby v4

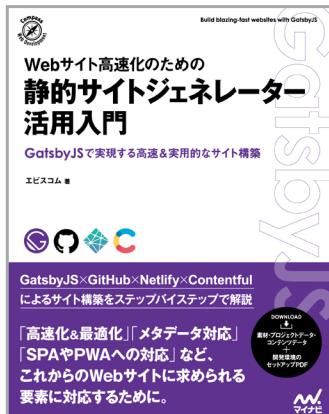
gatsby-source-contentful v6

new Gatsby image plugin

対応ガイド

で Gatsby v4 に対応させたプロジェクトをもとに、microCMS へ対応させていきます。

PDF は GitHub (<https://github.com/ebisucom/gatsbyjs-book/>) で配布しています。



Webサイト高速化のための 静的サイトジェネレーター活用入門

GatsbyJSで実現する高速＆実用的なサイト構築

URL: <https://book.mynavi.jp/ec/products/detail/id=115483>

↗ <https://ebisu.com/gatsbyjs-book/>

➡ <https://amzn.to/2x5nuyq>

- ・本書に記載された内容は、情報の提供のみを目的としております。したがって、本書を用いての運用はすべてお客様自身の責任と判断において行ってください。
- ・本書の制作にあたっては正確な記述につとめましたが、著者や出版社のいずれも、本書の内容に関してなんらかの保証をするものではなく、内容に関するいかなる運用結果についてもいっさいの責任を負いません。あらかじめご了承ください。
- ・本書中に掲載している画面イメージなどは、特定の設定に基づいた環境にて再現される一例です。ハードウェアやソフトウェアの環境によっては、必ずしも本書通りの画面にならないことがあります。あらかじめご了承ください。
- ・本書は 2021 年 11 月段階での情報に基づいて執筆されています。本書に登場するソフトウェアのバージョン、URL、製品のスペックなどの情報は、すべてその原稿執筆時点でのものです。執筆以降に変更されている可能性がありますので、ご了承ください。
- ・本書中に登場する会社名および商品名は、該当する各社の商標または登録商標です。本書では®および™マークは省略させていただいております。

CONTENTS

もくじ

CHAPTER

1

アカウントとコンテンツの準備

5

STEP 1-1	アカウントの準備	6
	microCMS	6
	microCMS のアカウントを作成する	6
STEP 1-2	コンテンツの準備	8
	サービスの作成	9
	API の作成	10
	コンテンツデータのインポート	14
STEP 1-3	GraphQL で microCMS のデータを扱うための準備	17
	API キーの確認	17
	プラグインのインストールと設定	17
STEP 1-4	変更点の確認	20
	contentfulBlogPost と microcmsBlog	20
	contentfulCategory と microcmsCategory	23

CHAPTER

2

記事一覧

24

STEP 2-1	トップページ	25
	クエリを追加する	26
	microCMS からのデータに置き換える	27
	レスポンシブイメージの設定を行う	28

STEP 2-2	記事一覧ページ	30
	クエリを追加する	30
	microCMS からのデータに置き換える	31
STEP 2-3	カテゴリーページ	33
	クエリを追加する	33
	microCMS からのデータに置き換える	35

CHAPTER

3**記事****37**

STEP 3-1	記事ページ	38
	microCMS からのデータに置き換える	38
	メタデータ	39
	アイキャッチ画像	40
	記事	41
STEP 3-2	リッチテキストの処理	43
STEP 3-3	仕上げ	47
	COLUMN microCMS によるサイトの更新	48

APPENDIX GatsbyImageを使う **49**

APPENDIX A	createRemoteFileNode	50
	eyecatch 画像	51
	記事内の画像	54

microCMS

1

アカウントとコンテンツの準備

- 1-1 アカウントの準備
- 1-2 コンテンツの準備
- 1-3 GraphQLでmicroCMSのデータを扱うための準備
- 1-4 変更点の確認

Build blazing-fast websites with GatsbyJS

GatsbyJS

1-1 アカウントの準備

microCMS

microCMS は日本製ヘッドレス CMS サービスで、非常にわかりやすいインターフェースが特徴です。無料のプランが用意されていますので、手軽に試すことができます。

- API 数： 10 個
- Webhook 数： 無制限
- カスタムフィールド数： 無制限
- コンテンツ数： 無制限
- データ転送量： 100GB

※プラン詳細はこちら <https://microcms.io/pricing/>

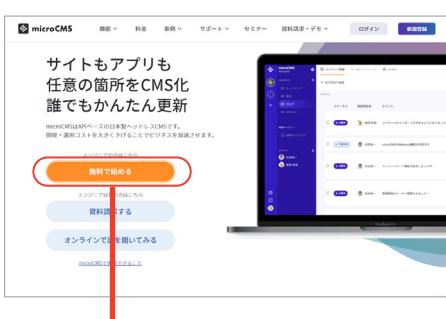
また、画像処理には imgix の API が利用可能です。



microCMS
<https://microcms.io/>

microCMSのアカウントを作成する

①



上記 microCMS のサイトにアクセスし、「新規登録」または「無料で始める」をクリックします。

(2)

アカウント登録

メールアドレス
mailaddress@microcms.io

パスワード

6文字以上で入力してください。
大文字、小文字、数字を含める必要があります。

利用規約、プライバシーポリシーに同意します

登録する

アカウント登録へ
こちらの方はごちら

「アカウント登録」ページが表示されますので、メールアドレスとパスワードを入力します。 「利用規約、プライバシーポリシー」を確認し、問題がなければ同意をオンにして「登録する」をクリックします。

(3)

確認コード入力

メールに記載された確認コードを入力してください。

確認コード
123456

送信する

確認コードの入力を求められますので、メールで送られてきたコードを入力し、「送信する」をクリックします。

(4)

サービス情報を入力

microCMSにご登録ありがとうございます

以下のステップでコンテンツを構築していきましょう！

01 02 03

1. パーティー登録
会員登録をする際の会員登録情報を入力して下さい。会員登録を行って下さい。

2. API登録
API登録を行う際のAPI登録情報を入力して下さい。会員登録を行って下さい。

3. コンテンツ登録
会員登録を行って下さい。

1/3ステップ

無事にアカウントが作成されると、左のような画面が表示されます。

メッセージを閉じたら、コンテンツの準備をしていきます。

サービス情報を入力

サービス名
会員登録やAPI登録などの情報を入力してください。

サービスURL
サービス登録する際のサブドメインを入力して下さい。

microcms.io

1/3ステップ

1-2 コンテンツの準備

microCMS でコンテンツを準備します。ここでは Contentful で構築したコンテンツと同様の構造で作成していきます。

The image shows the microCMS interface with several windows open, illustrating the process of creating content:

- 記事 (Article):** A window titled "新規作成" (New) for creating a "Food" article. It includes fields for "タイトル" (Title), "スラッグ" (Slug), "内容" (Content), and "カバーフoto" (Cover photo). A preview of the article content is shown below.
- サービス (Service):** A list view showing various services like "everyday", "spice02", "orange", "cake", "cupcake", "lemon", "grape", "cheese", "variation", "rice", "sandwich", "red", "hot", and "juice". Each item has a thumbnail image and a "編集" (Edit) button.
- カテゴリー (Category):** A list view showing categories: "everyday", "spice", "rice", "cupcake", "juice", and "hot".
- メディア (Media):** A list view showing media items: "everyday", "spice02", "orange", "cake", "cupcake", "lemon", "grape", "cheese", "variation", "rice", "sandwich", "red", "hot", and "juice". Each item has a thumbnail image and a "編集" (Edit) button.

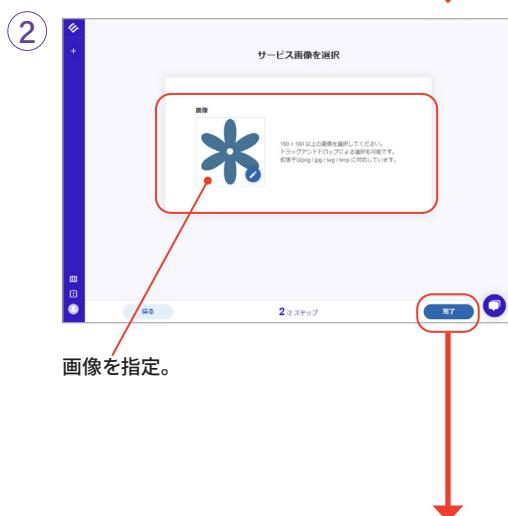
サービスの作成

まずは、サービスの作成です。Contentful のスペースに相当します。



サービス名とサービス ID を指定します。自分でわかりやすいもので問題ありません。ここではサービス名を「Essentials ブログ」、サービス ID を「essentials-blog」と指定しています。

設定ができたら「次へ」をクリックします。



サービスのアイコンとして表示される画像を指定し、「完了」をクリックします。



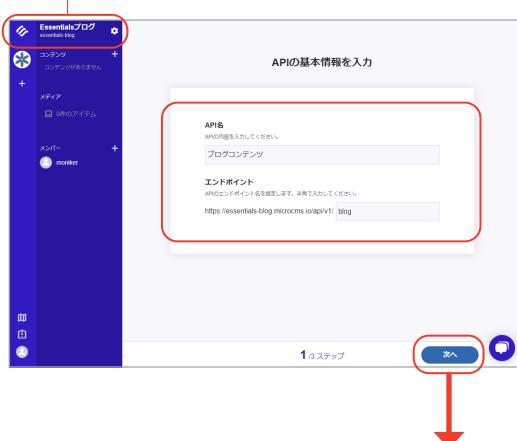
「サービス登録が完了しました」と表示されますので、表示された URL をクリックします。

APIの作成

続いて、APIを作成していきます。APIはContentfulのコンテンツタイプに相当するものです。

① ブログコンテンツ用のAPIを作成する

「Essentialsブログ」サービス。



まずは、ブログのコンテンツデータのためのAPIを作成します。ここではAPI名を「ブログコンテンツ」、エンドポイントを「blog」と指定しています。

エンドポイントで指定したものはGraphQLのフィールド名として、「microcmsBlog」といった形で使用されます。

設定ができたら「次へ」をクリックします。



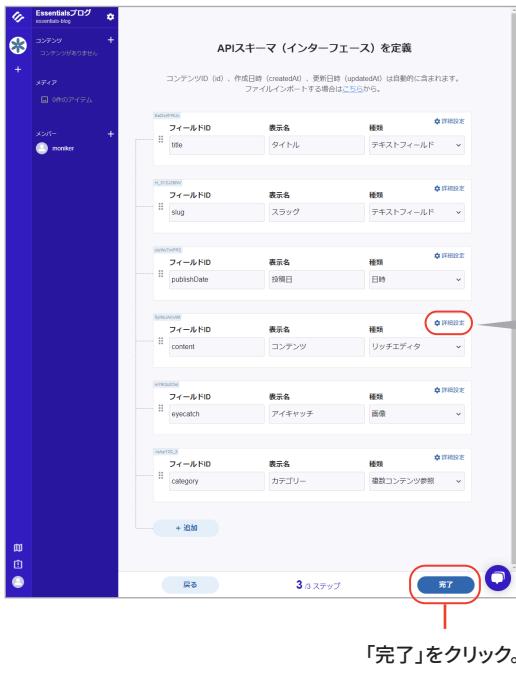
「APIの型を選択」では「リスト形式」を選択して、次へをクリックします。



ダウンロードデータに同梱したapi-blogpost.jsonをインポート。

「APIスキーマ（インターフェース）を定義」では、コンテンツを管理するためのフィールドを用意していきます。

本書ではインポートデータを用意しましたので、「ファイルをインポートする場合はこちらから」からapi-blogpost.jsonをインポートします。



左のように6つのフィールドが追加されます。リッチエディタのフィールド(content)については「詳細設定」をクリックし、「画像のレスポンスにwidthとheightを含む」がオンになっていることを確認します。問題がなければ「完了」をクリックします。

画像のレスポンスにwidthとheightを含む
オンの場合はタグにwidthとheightが含まれます。

画像に関する設定。

フィールドID	表示名	種類
title	タイトル	テキストフィールド
slug	スラッグ	テキストフィールド
publishDate	投稿日	日時
content	コンテンツ	リッチエディタ
eyecatch	アイキャッチ	画像
category	カテゴリー	複数コンテンツ参照

※全フィールド必須項目(入力必須)に設定。

設定をインポートせずにフィールドを作成する場合、フィールドID、表示名、種類を指定して作成します。

必須項目
設定をONにすると入力時の入力が必須となります

説明文
入力画面に表示する説明文です。入力者にとってわかりやすい説明を入力しましょう。
例 1160x400サイズの画像を選択してください

閉じる フィールド削除

入力を必須にする設定は「詳細設定」に用意されています。

選択できる
フィールドの種類。

フィールドID。	表示名。	種類。
title	タイトル	テキストフィールド

種類を選択してください

- テキストフィールド
自由入力の行テキストです。タイトル等に適しています。
- テキストエリア
自由入力の複数行テキストです。ブレーンストロームによる入力になります。
- リッチエディタ
自由入力の複数行テキストです。リッチエディタによる機能が可能です。APIからHTMLが取得できます。
- 画像
画像用のフィールドです。APIから直接画像が取得されます。
- ファイル
Starterプラン以降をご利用いただけます
- 日時
日付のフィールドです。カレンダーや日付を選択することができます。
- 真偽値
Boolean型のフィールドです。スタイルでオプションを切り替えることができます。
- コンテンツ多選
複数のコンテンツを複数選ぶことができます。静的ルートリスト型の場合には選択肢になります。
- 複数コンテンツ参照
複数のコンテンツを複数選ぶことができます。レスポンシブは矩形形式となります。
- 数字
number型のフィールドです。入力時は数値型のキーボードが使用されます。
- カタログ
カラムのフィールドです。設定済みのカラムフィールドを用いて入力ができます。
- 組み込み
内蔵されたカスタムフィールドを複数選択し、複数挿入が可能になります。

2 カテゴリー用のAPIを作成する



続いて、カテゴリーのための API を作成します。「コンテンツ」の右にある「+」をクリックし、API 名を「カテゴリー」、エンドポイントを「category」と指定します。

APIの型を選択

リスト形式
JSONの配列を返す形で作成します。プロジェクト内でセーブ一覧、カーディナルに適しています。

オブジェクト形式
JSONオブジェクトを返す形で作成します。既存ファイルや複数ページ情報をなどに適しています。

次へ

「API の型を選択」ではこちらもリスト形式を選択し、api-category.json をインポートします。

APIスキマ（インターフェース）を定義

コンテンツID (id)、作成日時 (createdAt)、更新日時 (updatedAt) は自動的に含まれます。
ファイルインポートする場合はここから。

フィールドID [例: category]	表示名 [例: カテゴリ名]	種類 [例: テキストフィールド]
入力してください	入力してください	選択してください

次へ

ダウンロードデータに同梱したapi-category.jsonをインポート。

APIスキマ（インターフェース）を定義

コンテンツID (id)、作成日時 (createdAt)、更新日時 (updatedAt) は自動的に含まれます。
ファイルインポートする場合はここから。

フィールドID [例: category]	表示名 [例: カテゴリ名]	種類 [例: テキストフィールド]
category	カテゴリー名	テキストフィールド
フィールドID [例: categorySlug]	表示名 [例: カテゴリースラッグ]	種類 [例: テキストフィールド]
categorySlug	カテゴリー名	テキストフィールド

戻る 次へ 3 / 3 歩き

完了

問題がなければ左のような表示になりますので、完了をクリックします。

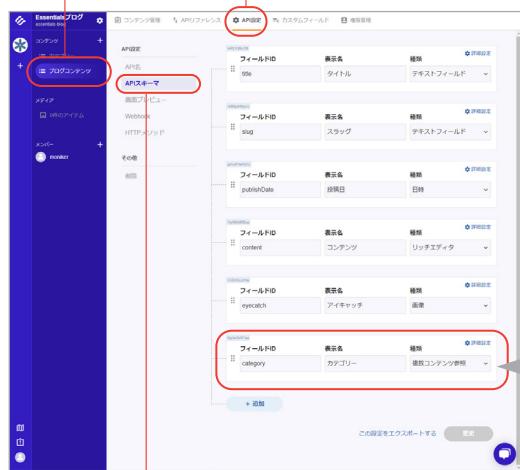
ここでは 2 つのフィールドを用意しています。

フィールドID	表示名	種類
category	カテゴリー名	テキストフィールド
categorySlug	カテゴリー名	テキストフィールド

※全フィールド必須項目
(入力必須)に設定。

③ 複数コンテンツの参照を設定する

「ログコンテンツ」をクリック。



「APIスキーマ」を選択。



「API設定」をクリック。

最後に、複数コンテンツの参照を設定します。
「ログコン텐ツ」をクリックし、「API 設定」で「API スキーマ」を選択します。

作成した API スキーマが表示されますので、
category のフィールドの「種類」の欄をクリックします。

「種類」の欄をクリック。



「カテゴリー (category)」を選択。

「複数コンテンツ参照」を選択。

「決定」をクリック。

「カテゴリー (category)」を選択。

「決定」をクリック。

「変更」をクリック。

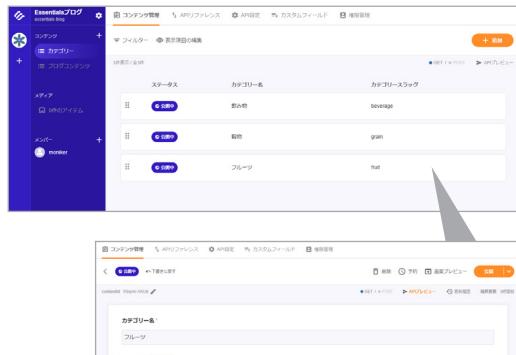
種類の選択画面が開きますので、「複数コンテンツ参照」を選択します。参照したいコンテンツとして「カテゴリー (category)」を選択し、「決定」をクリックします。

元の画面に戻ります。最後に「変更」をクリックして設定を保存したら完了です。

コンテンツデータのインポート

コンテンツのデータをインポートしていきます。

① カテゴリーのデータをインポートする



「フルーツ」カテゴリーの
編集画面。

まずはカテゴリーのデータをインポートするため、「カテゴリー」を選択して、「コンテンツ管理」を開きます。

空の状態ですので、「インポートする」をクリックします。

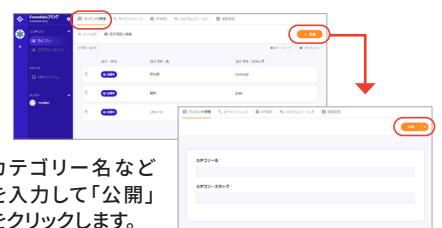
「ファイルの選択」で `input-category.csv` を選択し、インポートを開始します。

ダウンロードデータに同梱した `input-category.csv` を選択。

インポートが完了すると、左のように表示されます。ここでは3つのカテゴリーを用意しています。

クリックして、中身を確認しておいてください。

新しいカテゴリーを追加する場合、「コンテンツ管理」の画面で「追加」をクリックします。



カテゴリー名などを
入力して「公開」
をクリックします。

② ブログ記事のデータをインポートする



続けて、ブログ記事のデータをインポートするため、「ブログコンテンツ」の「コンテンツ管理」を開きます。

カテゴリーのときと同様に、「インポートする」から input-blog.csv をインポートします。

ダウンロードデータに同梱した
input-blog.csvを選択。

インポートが完了すると、以下のように表示されます。ここでは 14 件の記事を用意しています。ただし、アイキャッチ、リッチテキスト内の画像と見出しの設定、カテゴリーの設定はインポートできていないため、各記事の編集画面を開いて設定をしていきます。

スラッグ	タイトル	日付
毎日のフルーツで爽やかさを加えて	everyday	2020/01/16
スパイスの香りと爽快感	spice	2020/01/16
朝のフルーツの簡単レシピ	orange	2020/01/16
甘いもの好きのため	cake	2020/01/16
カッパーを主な材料の料理	cupcake	2020/01/16
レモンの柑橘類の簡単レシピ	lemon	2020/01/16
桃のフルーツの簡単レシピ	peach	2020/01/16
バナナの簡単レシピ	banana	2020/01/16
苺の簡単レシピ	strawberry	2020/01/16
桃の簡単レシピ	peach	2020/01/16

「毎日のフルーツで爽やかさを加えて」
の編集画面。

記事を編集したら「公開」をクリックし、編集内容を保存するのを忘れないようにします。すべての記事を編集したら、コンテンツの準備は完了です。

編集後は「公開」をクリックして保存。

The image shows two screenshots of the microCMS interface. The left screenshot displays a list of blog posts with titles like '毎日のフルーツで爽やかさを加えて', 'スパイスの香りと刺激', and '彩り鮮やかなオレンジの秘密'. The right screenshot shows a detailed edit view for the post '毎日のフルーツで爽やかさを加えて'. It includes fields for the title ('毎日のフルーツで爽やかさを加えて'), subtitle (''), category (''), tags (''), and rich text editor. The rich text editor contains placeholder text and a preview of the image 'season.jpg'. A red circle highlights the '公開' (Publish) button at the top right of the edit screen.

「見出し2」に設定。
画像を挿入。
アイキャッチ画像を指定。
カテゴリーを指定。

「毎日のフルーツで爽やかさを加えて」の編集画面。

本書のサンプルでは次のように設定しています。使用する画像はダウンロードデータ内の base/images-blogpost/ フォルダに収録しています。

タイトル	アイキャッチ	カテゴリー	リッチテキスト内の画像と見出し
毎日のフルーツで爽やかさを加えて	everyday.jpg	フルーツ、穀物	画像 (season.jpg)、見出し
スパイスの香りと刺激	spice.jpg	飲み物	画像 (spice01.jpg)、見出し
彩り鮮やかなオレンジの秘密	orange.jpg	フルーツ	画像 (color.jpg)
甘いものとフルーツの相性	cake.jpg	穀物	
カップケーキを焼くときのちょっとしたコツ	cup.jpg	穀物	
レモンがないときに代わりになるもの	lemon.jpg	フルーツ	
色とりどりのぶどう	grape.jpg	フルーツ	見出し
チーズやパンといっしょに食べたい果物	eat.jpg	フルーツ、穀物	
穀物のバリエーション	variation.jpg	穀物	
どうしても甘いものが欲しくなるとき	taste.jpg	飲み物、穀物	
お手軽サンドイッチ	sandwich.jpg	フルーツ、穀物	
イチゴの色が赤いわけ	red.jpg	フルーツ	
ハーブの香りでひとやすみ	herb.jpg	飲み物	
フレッシュジュースが飲みたい	juice.jpg	フルーツ、飲み物	

1-3

GraphQLでmicroCMSのデータを扱うための準備



APIキーの確認

microCMS のデータを利用するため、必要になる API キーを確認します。

歯車アイコンをクリック。 「API-KEY」を選択。

登録されたキーを選択。

デフォルト権限でGETが選択されていることを確認。

キーを作成する場合は「追加」をクリック。デフォルト権限でGETが選択されていることを確認し、名前をつけて作成・登録します。

「表示」をクリックしてキーの値を確認します。

プラグインのインストールと設定

サポート PDF (`gatsby-v4.pdf`) で Gatsby v4 に対応させたプロジェクトを用意し、ブログ部分を microCMS のデータに置き換えていきます。まずは、Gatsby から microCMS を利用するための、ソースプラグインをインストールします。

```
$ yarn add gatsby-source-microcms
```

`gatsby-source-microcms`
<https://www.gatsbyjs.org/packages/gatsby-source-microcms/>

※MicroCMSの新APIキー(X-MICROCMS-API-KEY)にはv2.0.0以上が必要です。

続けて、次ページのように gatsby-config.js にプラグインの設定を追加します。
ここでは Contentful の設定の下に追加しています。

```
...
{
  resolve: `gatsby-source-contentful`,
  options: {
    spaceId: process.env.CONTENTFUL_SPACE_ID,
    accessToken: process.env.CONTENTFUL_ACCESS_TOKEN,
    host: process.env.CONTENTFUL_HOST,
  },
},
{
  resolve: "gatsby-source-microcms",
  options: {
    apiKey: process.env.microCMS_API_KEY,
    serviceId: "essentials-blog",
    apis: [
      {
        endpoint: "blog", •
      },
      {
        endpoint: "category", •
      },
    ],
  },
}
```

Contentfulの設定。

microCMSの設定。

ブログコンテンツ用APIの
エンドポイントを指定。

カテゴリー用APIの
エンドポイントを指定。

gatsby-config.js

apiKey

apiKey には先程確認した API キーを指定しますが、ここでは環境変数で指定できるようにしています。

serviceId

serviceId には、P.9 で設定したサービス ID を指定します。

endpoint

endpoint には各 API (P.10 と P.12) で指定したエンドポイントを指定します。ここでは「blog」と「category」を指定しています。

設定が完了したら、開発サーバーを起動します。起動の際には、Contentful に加えて microCMS の環境変数も指定するのを忘れないでください。

```
$ microCMS_API_KEY=xxxxxxxxx CONTENTFUL_SPACE_ID=xxxxxxxxx CONTENTFUL_ACCESS_TOKEN=xxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxx CONTENTFUL_HOST=cdn.contentful.com gatsby develop -H 0.0.0.0
```

無事に起動すると、**GraphiQL** には microCMS のフィールドが現れます。

The screenshot shows the GraphiQL interface with the following details:

- Explorer** tab is selected.
- Query**: MyQuery
- Result** pane shows the following JSON output (with some text redacted):


```
query MyQuery {
  allMicrocmsBlog {
    edges {
      node {
        title
      }
    }
  }
}
```

Execution results (redacted text):

```
{
  "data": {
    "allMicrocmsBlog": [
      {
        "edges": [
          {
            "node": {
              "title": "毎日のフルーツで爽やかさを加えて"
            }
          },
          {
            "node": {
              "title": "スパイスの香りと刺激"
            }
          },
          {
            "node": {
              "title": "彩り鮮やかなオレンジの秘密"
            }
          },
          {
            "node": {
              "title": "甘いものとフルーツの相性"
            }
          },
          {
            "node": {
              "title": "カッブケーキを焼くときのちょっとしたコツ"
            }
          },
          {
            "node": {
              "title": "レモンがないときに代わ"
            }
          }
        ]
      }
    ]
  }
}
```
- Left Sidebar (Explorer)** shows available fields:
 - allContentfulAsset
 - allContentfulBlogPost
 - allContentfulCategory
 - allContentfulContentType
 - allContentfulEntry
 - allDirectory
 - allFile
 - allImageSharp
 - allMicrocmsBlog
 - filter:
 - limit:
 - skip:
 - sort:
 - distinct
 - edges:
 - next
 - node:
 - blogId
 - category
 - children
 - content
 - createdAt
 - eyecatch
 - id
 - internal
 - parent
 - publishDate
 - slug
 - sortIndex
 - title
 - updatedAt
 - previous
- Bottom Left Sidebar** shows available fields:
 - allFile
 - allImageSharp
 - allMicrocmsBlog
 - allMicrocmsCategory
 - allSite
 - directory
 - file
 - imageSharp
 - microcmsBlog
 - microcmsCategory
 - site
 - siteBuildMetadata

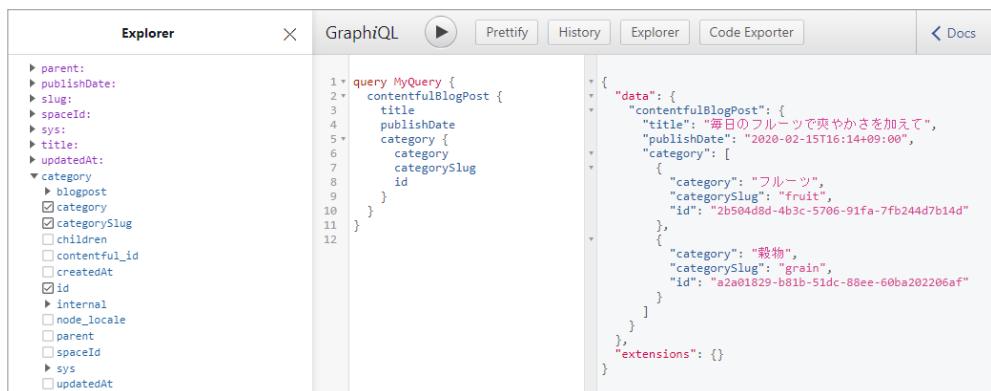
A red box highlights the 'allMicrocmsBlog' and 'allMicrocmsCategory' fields in the sidebar, and another red box highlights the 'microcmsBlog' and 'microcmsCategory' fields in the sidebar. A red arrow points from the 'microcmsBlog' field to the 'title' field in the JSON result, with the annotation 'microCMSから取得した記事のタイトル。' (Title of the article obtained from microCMS).

1-4 変更点の確認

GraphQL を利用して、取得できるデータの違いを確認していきます。

contentfulBlogPost と microcmsBlog

まずは、contentfulBlogPost と microcmsBlog です。構成を揃えるように準備していますので、取得できるデータに違いはありません。



```

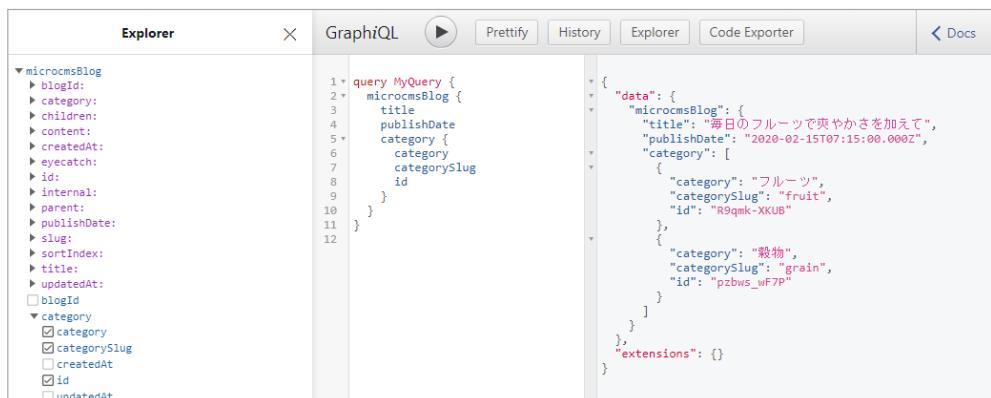
Explorer X GraphiQL ▶ Prettify History Explorer Code Exporter < Docs

▶ parent:
▶ publishDate:
▶ slug:
▶ spaceId:
▶ sys:
▶ title:
▶ updatedAt:
▼ category
  ▶ blogpost
   category
   categorySlug
  □ children
  □ contentful_id
  □ createdAt
   id
  ▶ internal
  □ node_locale
  □ parent
  □ spaceId
  ▶ sys
  □ updatedAt

1+ query MyQuery {
2+   contentfulBlogPost {
3+     title
4+     publishDate
5+     category {
6+       category
7+       categorySlug
8+       id
9+     }
10+   }
11+
12+ }

{
  "data": [
    {
      "contentfulBlogPost": {
        "title": "毎日のフルーツで爽やかさを加えて",
        "publishDate": "2020-02-15T16:14+09:00",
        "category": [
          {
            "category": "フルーツ",
            "categorySlug": "fruit",
            "id": "2b504d8d-4b3c-5706-91fa-7fb244d7b14d"
          },
          {
            "category": "穀物",
            "categorySlug": "grain",
            "id": "a2a01829-b81b-51dc-88ee-60ba202206af"
          }
        ]
      }
    }
  ],
  "extensions": {}
}

```

```

Explorer X GraphiQL ▶ Prettify History Explorer Code Exporter < Docs

▼ microcmsBlog
  ▶ blogId:
  ▶ category:
  ▶ children:
  ▶ content:
  ▶ createdAt:
  ▶ eyecatch:
  ▶ id:
  ▶ internal:
  ▶ parent:
  ▶ publishDate:
  ▶ slug:
  ▶ sortIndex:
  ▶ title:
  ▶ updatedAt:
  □ blogId
  ▼ category
     category
     categorySlug
    □ createdAt
     id
    □ updatedAt

1+ query MyQuery {
2+   microcmsBlog {
3+     title
4+     publishDate
5+     category {
6+       category
7+       categorySlug
8+       id
9+     }
10+
11+   }
12+ }

{
  "data": [
    {
      "microcmsBlog": {
        "title": "毎日のフルーツで爽やかさを加えて",
        "publishDate": "2020-02-15T07:15:00.000Z",
        "category": [
          {
            "category": "フルーツ",
            "categorySlug": "fruit",
            "id": "R9qmk-XKUB"
          },
          {
            "category": "穀物",
            "categorySlug": "grain",
            "id": "pzbws_wF7P"
          }
        ]
      }
    }
  ],
  "extensions": {}
}

```



ただし、eyecatch に関してはかなりの違いがあります。

Contentful からは gatsby-image 用のデータをはじめ、さまざまなデータが取得できますが、microCMS から取得できるのは画像の url と画像の width&height となっています。そのため、画像関連はちょっと工夫が必要です。

Explorer X GraphiQL ▶ Prettyfy History Explorer Code Exporter < Docs

▼ eyecatch
 □ children
 □ contentful_id
 □ createdAt
 ☑ description
 ▼ file
 □ contentType
 ▼ details
 ▼ image
 ☑ height
 ☑ width
 □ size
 □ fileName
 □ url
 ► fixed
 ► fluid
 ☑ gatsbyImageData
 □ aspectRatio:
 □ backgroundColor:
 □ breakpoints:
 □ cornerRadius:
 □ cropFocus:
 □ formats:
 □ height:
 □ jpegProgressive:
 □ layout:
 □ outputPixelDensities:
 □ placeholder:
 □ quality:
 □ resizingBehavior:
 □ sizes:
 □ width:
 □ id

1 query MyQuery {
2 contentfulBlogPost {
3 title
4 eyecatch {
5 gatsbyImageData
6 description
7 file {
8 details {
9 image {
10 height
11 width
12 }
13 }
14 }
15 }
16 }
17 }
18

 {
 "data": {
 "contentfulBlogPost": {
 "title": "毎日のフルーツで爽やかさを加えて",
 "eyecatch": {
 "gatsbyImageData": {
 "images": {
 "sources": [
 {
 "srcSet": "<https://images.ctfassets.net/k3ffctx2yqhjQhJekTkj52cLyuoqciTNI/efd71ef7d3c07e5d27b7f86033353cf/everyday.jpg?w=400&h=165&q=50&fm=webp>
 "<https://images.ctfassets.net/k3ffctx2yqhjQhJekTkj52cLyuoqciTNI/efd71ef7d3c07e5d27b7f86033353cf/ever>
 "yday.jpg?w=800&h=331&q=50&fm=webp">yday.jpg?w=800&h=331&q=50&fm=webp
 "<https://images.ctfassets.net/k3ffctx2yqhjQhJekTkj52cLyuoqciTNI/efd71ef7d3c07e5d27b7f86033353cf/ever>
 "yday.jpg?w=1600&h=661&q=50&fm=webp">yday.jpg?w=1600&h=661&q=50&fm=webp
 "<https://images.ctfassets.net/k3ffctx2yqhjQhJekTkj52cLyuoqciTNI/efd71ef7d3c07e5d27b7f86033353cf/ever>
 "yday.jpg?w=1600px">1600px,
 "sizes": "(min-width: 1600px) 1600px,
 100vw",
 "type": "image/webp"
 }
 },
 "fallback": {
 "src": "<https://images.ctfassets.net/k3ffctx2yqhjQhJekTkj52cLyuoqciTNI/efd71ef7d3c07e5d27b7f86033353cf/progressive+q=50&fm=jpg>",
 "srcSet": "<https://images.ctfassets.net/k3ffctx2yqhjQhJekTkj52cLyuoqciTNI/efd71ef7d3c07e5d27b7f86033353cf/progressive+q=50&fm=jpg>"
 }
 },
 "src": "<https://images.ctfassets.net/k3ffctx2yqhjQhJekTkj52cLyuoqciTNI/efd71ef7d3c07e5d27b7f86033353cf/progressive+q=50&fm=jpg>"
 },
 "srcSet": "<https://images.ctfassets.net/k3ffctx2yqhjQhJekTkj52cLyuoqciTNI/efd71ef7d3c07e5d27b7f86033353cf/progressive+q=50&fm=jpg>"
 }



GraphiQL

Explorer

Prettify

History

Code Exporter

Docs

```
query MyQuery {
  microcmsBlog {
    title
    eyecatch {
      url
      height
      width
    }
  }
}

{
  "data": {
    "microcmsBlog": {
      "title": "毎日のフルーツで爽やかさを加えて",
      "eyecatch": {
        "url": "https://images.microcms-assets.io/protected/ap-northeast-1:b6f0665c-97b7-4807-bfac-f6dce328dbc3/service/essentials-blog/media/everyday.jpg",
        "height": 661,
        "width": 1600
      }
    },
    "extensions": {}
  }
}
```



また、リッチテキストのデータが、Contentful では AST が取得できるのに対して、microCMS では HTML が取得できるあたりも異なります。



```

Explorer × GraphiQL ▶ Prettify History Explorer Code Exporter < Docs

contentfulBlogPost
▶ category:
▶ children:
▶ content:
▶ contentful_id:
▶ createdAt:
▶ eyecatch:
▶ id:
▶ internal:
▶ node_locale:
▶ parent:
▶ publishDate:
▶ slug:
▶ spaceId:
▶ sys:
▶ title:
▶ updatedAt:
▶ category
□ children
▼ content
 raw
▶ references
□ contentful_id
□ createdAt
▶ eyecatch
□ id
▶ internal
□ node_locale
□ parent
□ publishDate
□ slug
□ spaceId
□ sys

1+ query MyQuery {
  2+   contentfulBlogPost {
  3+     title
  4+     content {
  5+       raw
  6+     }
  7+   }
  8+ }

1+ query MyQuery {
  2+   contentfulBlogPost {
  3+     title: "毎日のフルーツで爽やかさを加えて",
  4+     content: {
  5+       raw: "[{"data": {}, "content": [{"data": {}, "marks": []}], "value": "\u00a5フルーツには適度な甘みと酸味と爽やかさがあって、毎日食べてても飽きません。パンやヨーグルトとの相性もばっちりです。朝食にたくさんさんのフルーツを取り入れてみると、いつの間にかたくさんさんのフルーツを食べるようになっていました。"}, {"nodeType": "text"], "nodeType": "paragraph"}, {"data": {}, "content": [{"data": {}, "marks": []}], "value": "\u00a5フルーツには適度な甘みと酸味と爽やかさがあって、毎日食べてても飽きません。パンやヨーグルトとの相性もばっちりです。朝食にたくさんさんのフルーツを取り入れてみると、いつの間にかたくさんさんのフルーツを食べるようになっていました。"}, {"nodeType": "text"], "nodeType": "heading-2"}, {"data": {}, "content": [{"data": {}, "marks": []}], "value": "\u00a5野菜と同じようにフルーツにも旬があります。ただ、地域によって旬の時期には違いがありますので、居住地域の旬を押さえておくのがおすすめです。"}, {"nodeType": "text"], "nodeType": "paragraph"}, {"data": {}, "target": {"sys": {"id": "3qjxvdvRUT088BpgyOFUyxS"}, "type": "Link", "linkType": "Asset"}}, "content": [{"nodeType": "embedded-asset-block"}, {"data": {}, "marks": []}], {"value": "\u00a5早い時期に手に入る果物はラズベリーとブルーベリーです。これらの果物は生で食べることも、乾燥させてデザートとして調理することもできます。少し甘くて酸味があります。また、ベリーから作るフルーティーワインもあります。火を通してそのまま食べるのがおいしいです。"}, {"nodeType": "text"], {"nodeType": "paragraph"}, {"data": {}, "content": [{"data": {}, "marks": []}], "value": "\u00a5フルーツの旬を活かす。<br><p>野菜と同じようにフルーツにも旬があります。ただ、地域によって旬の時期には違いがありますので、居住地域の旬を押さえておくのがおすすめです。<br><br><img src='https://images.microcms-assets.io/protected/ap-northeast-1:b6f0f665c-97b7-4807-bfac-f6dce328dc3/service/essentials-blog/media/season.jpg' alt=''\ width='1600' height='800'><br>\u00a5早い時期に手に入る果物はラズベリーとブルーベリーです。これらの果物は生で食べることも、乾燥させてデザートとして調理することもできます。少し甘くて酸味があります。また、ベリーから作るフルーティーワインもあります。次はイチゴですが、今年の前半はイチゴジャムが新鮮です。火を通してそのまま食べるのがおいしいです。<br><br>パンとの組み合わせもいろいろ試してみましたが、特にライ麦パンがとても気に入りました！クリームとハチミツ、塩キャラ、バターサラに果物を加えるとペロッと食べられます。</p>"}], "extensions": {}}

```



```

Explorer × GraphiQL ▶ Prettify History Explorer Code Exporter < Docs

microcmsBlog
▶ blogId:
▶ category:
▶ children:
▶ content:
▶ createdAt:
▶ eyecatch:
▶ id:
▶ internal:
▶ parent:
▶ publishDate:
▶ slug:
▶ sortIndex:
▶ title:
▶ updatedAt:
□ blogId
▶ category
□ children
▼ content
 createdAt
▶ eyecatch
□ id
▶ internal
□ parent
□ publishDate
□ slug
□ sortIndex
 title
□ updatedAt
▶ microcmsCategory
▶ site
▶ siteBuildMetadata
▶ siteFunction

1+ query MyQuery {
  2+   microcmsBlog {
  3+     title
  4+     content
  5+   }
  6+ }

1+ query MyQuery {
  2+   microcmsBlog: {
  3+     title: "毎日のフルーツで爽やかさを加えて",
  4+     content: "毎日のフルーツには適度な甘みと酸味と爽やかさがあって、毎日食べてても飽きません。パンやヨーグルトとの相性もばっちりです。朝食にたくさんさんのフルーツを取り入れてみると、いつの間にかたくさんさんのフルーツを食べるようになっていました。<br><p>野菜と同じようにフルーツにも旬があります。ただ、地域によって旬の時期には違いがありますので、居住地域の旬を押さえておくのがおすすめです。<br><br><img src='https://images.microcms-assets.io/protected/ap-northeast-1:b6f0f665c-97b7-4807-bfac-f6dce328dc3/service/essentials-blog/media/season.jpg' alt=''\ width='1600' height='800'><br>\u00a5早い時期に手に入る果物はラズベリーとブルーベリーです。これらの果物は生で食べることも、乾燥させてデザートとして調理することもできます。少し甘くて酸味があります。また、ベリーから作るフルーティーワインもあります。次はイチゴですが、今年の前半はイチゴジャムが新鮮です。火を通してそのまま食べるのがおいしいです。<br><br>パンとの組み合わせもいろいろ試してみましたが、特にライ麦パンがとても気に入りました！クリームとハチミツ、塩キャラ、バターサラに果物を加えるとペロッと食べられます。</p>"},
  5+   "extensions": {}
}
```

contentfulCategory と microcmsCategory

続いて、contentfulCategory と microcmsCategory です。こちらも構造は揃えていますので、カテゴリーに関する情報に違いはありません。

ただし、contentfulCategory ではそのカテゴリーに含まれる記事が参照できるのに対して、microcmsCategory では記事の情報を得ることはできません。そのため、カテゴリーに属した記事の数は自分で用意することになります。



```

Explorer × GraphQL ▶ Prettify History Explorer Code Exporter < Docs
▼ contentfulCategory
  ▷ blogpost:
  ▷ category:
  ▷ categorySlug:
  ▷ children:
  ▷ contentful_id:
  ▷ createdAt:
  ▷ id:
  ▷ internal:
  ▷ node_locale:
  ▷ parent:
  ▷ spaceId:
  ▷ sys:
  ▷ updatedAt:
  ▷ blogpost:
    ▷ category
    □ children
    ▷ content
    □ contentful_id
    □ createdat
    ▷ eyecatch
    □ id
    ▷ internal
    □ node_locale
    □ parent
    □ publishDate
    □ slug
    □ spaceId
    ▷ sys
    ▷ title
    □ updatedAt
  ▷ category
  ▷ categorySlug
  □ children
  
```

```

1+ query MyQuery {
2+   contentfulCategory {
3+     category
4+     categorySlug
5+     blogpost {
6+       title
7+     }
8+   }
9+ }
10}

{
  "data": [
    {
      "contentfulCategory": {
        "category": "穀物",
        "categorySlug": "grain",
        "blogpost": [
          {
            "title": "毎日のフルーツで爽やかさを加えて"
          },
          {
            "title": "お手軽サンドイッチ"
          },
          {
            "title": "どうしても甘いものが欲しくなるとき"
          },
          {
            "title": "穀物のバリエーション"
          },
          {
            "title": "チーズやパンといっしょに食べたい果物"
          },
          {
            "title": "甘いものとフルーツの相性"
          },
          {
            "title": "カップケーキを焼くときのちょっとしたコツ"
          }
        ],
        "extensions": {}
      }
    }
  ]
}

```



```

Explorer × GraphQL ▶ Prettify History Explorer Code Exporter < Docs
▼ microcmsCategory
  ▷ category:
  ▷ categoryId:
  ▷ categorySlug:
  ▷ children:
  ▷ createdAt:
  ▷ id:
  ▷ internal:
  ▷ parent:
  ▷ sortIndex:
  ▷ updatedAt:
  ▷ category
  □ categoryId
  ▷ categorySlug
  □ children
  □ createdat
  □ id
  ▷ internal
  □ parent
  □ sortIndex
  □ updatedAt
  
```

```

1+ query MyQuery {
2+   microcmsCategory {
3+     category
4+     categorySlug
5+   }
6+ }
7

{
  "data": [
    {
      "microcmsCategory": {
        "category": "飲み物",
        "categorySlug": "beverage"
      }
    }
  ],
  "extensions": {}
}

```

他にも異なる部分はありますが、ひとまずこのあたりをおさえつつ対応を進めます。

microCMS

2

記事一覧

2-1 トップページ

2-2 記事一覧ページ

2-3 カテゴリーページ

Build blazing-fast websites with GatsbyJS

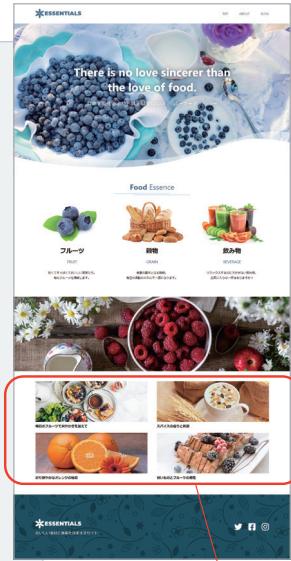
GatsbyJS

2-1 トップページ

トップページ (index.js) の記事一覧を microCMS に対応させます。index.js でヘッレス CMS のデータを利用しているのは、この部分です。

```
<section>
  <div className="container">
    <h2 className="sr-only">RECENT POSTS</h2>
    <div className="posts">
      {data.allContentfulBlogPost.edges.map(({ node }) => (
        <article className="post" key={node.id}>
          <Link to={`/blog/post/${node.slug}`}>
            <figure>
              <GatsbyImage
                image={node.eyecatch.gatsbyImageData}
                alt={node.eyecatch.description}
                style={{ height: "100%" }}
              />
            </figure>
            <h3>{node.title}</h3>
          </Link>
        </article>
      ))}
    </div>
  </div>
</section>
```

src/pages/index.js



トップページの
記事一覧。

id, **slug**, **title** はそのまま置き換える問題はありません。しかし、**GatsbyImage** はそのままでは使えません。

Contentful では管理している画像の **GatsbyImage** 用のデータを直接取得することができます。しかし、そうした機能がない場合にはリモートの画像をダウンロードした上で変換し、**GatsbyImage** 用のデータを GraphQL から利用できる形で用意することになります。

この一連の処理をしてくれるのが、**createRemoteFileNode** です。ただし、ビルドのたびにこの作業が行われるため、処理が重くなるという問題があります。

また、microCMS では高機能な画像処理 API である imgix が利用できます。そこで、imgix を活用する形で対応を進めています（GatsbyImage を使いたい場合は、P.49 の Appendix を参照してください）。



imgix
<https://www.imgix.com/>

クエリを追加する

まずは Contentful のクエリに倣って、microCMS のクエリを追加します。

```
...
export const query = graphql` 
query {
  ...
  allContentfulBlogPost(
    sort: { order: DESC, fields: publishDate }
    skip: 0
    limit: 4
  ) {
    edges {
      node {
        title
        id
        slug
        eyecatch {
          gatsbyImageData(width: 573, layout: CONSTRAINED)
          description
        }
      }
    }
  }
  allMicrocmsBlog(
    sort: { order: DESC, fields: publishDate }
    skip: 0
    limit: 4
  ) {
    edges {
      node {
        title
        id
        slug
        eyecatch {
          url
          width
          height
        }
      }
    }
  }
}
`
```

Contentfulのクエリ

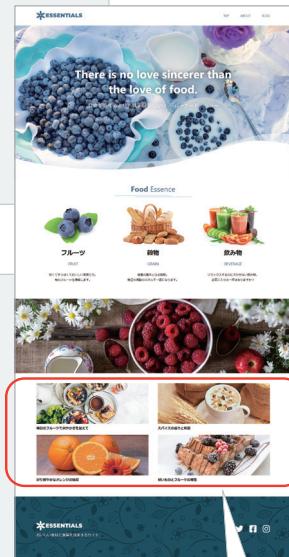
microCMSのクエリ

src/pages/index.js

microCMSからのデータに置き換える

`allContentfulBlogPost` を `allMicrocmsBlog` に置き換え、`GatsbyImage` を `` で書き換えます。`description` を用意していないので、`alt` は空にしています。

```
<section>
  <div className="container">
    <h2 className="sr-only">RECENT POSTS</h2>
    <div className="posts">
      {data.allContentfulBlogPost.edges.map(({ node }) => (
        <article className="post" key={node.id}>
          <Link to={`/blog/post/${node.slug}`}>
            <figure>
              <GatsbyImage
                image={node.eyecatch.gatsbyImageData}
                alt={node.eyecatch.description}
                style={{ height: "100%" }}
              />
            </figure>
            <h3>{node.title}</h3>
          </Link>
        </article>
      ))}
    </div>
  </div>
</section>
```



```
<section>
  <div className="container">
    <h2 className="sr-only">RECENT POSTS</h2>
    <div className="posts">
      {data.allMicrocmsBlog.edges.map(({ node }) => (
        <article className="post" key={node.id}>
          <Link to={`/blog/post/${node.slug}`}>
            <figure>
              <img src={node.eyecatch.url} alt="" />
            </figure>
            <h3>{node.title}</h3>
          </Link>
        </article>
      ))}
    </div>
  </div>
</section>
```

src/pages/index.js

```
<figure>
  
</figure>
```

これで、microCMSからのデータへの置き換えが完了です。ただし、画像はオリジナルサイズ（横幅 1600px）のものを `src` 属性で読み込んでいるだけの状態です。

レスポンシブイメージの設定を行う

レスポンシブイメージの設定を行い、解像度などに応じて最適なサイズの画像で表示します。microCMS では imgix の API が利用できますので、imgix の公式プラグインである `@imgix/gatsby` を利用します。このプラグインを使うことで、imgix を使った `GatsbyImage` 互換のデータを生成することができます。いくつかの使い方が用意されていますが、ここではシンプルにコンポーネントとして利用します。

```
$ yarn add @imgix/gatsby
```

`@imgix/gatsby` をインストールしたら `ImgixGatsbyImage` を import し、先程の `` を次のように書き換えます。これで、レスポンシブイメージの設定が出力されるようになります。

```
...
import { ImgixGatsbyImage } from '@imgix/gatsby'

const Home = ({ data }) => (
  ...
  <section>
    <div className="container">
      <h2>RECENT POSTS</h2>
      <div className="posts">
        {data.allMicrocmsBlog.edges.map(({ node }) => (
          <article className="post" key={node.id}>
            <Link to={`/blog/post/${node.slug}/`}>
              <figure>
                <ImgixGatsbyImage
                  src={node.eyecatch.url}
                  imgixParams={{ auto: ["format", "compress"] }}
                  layout="constrained"
                  width={573}
                  sourceWidth={node.eyecatch.width}
                  sourceHeight={node.eyecatch.height}
                  style={{ height: "100%" }}
                />
              </figure>
              <h3>{node.title}</h3>
            </Link>
          </article>
        ...
      </div>
    </div>
  </section>
)
```

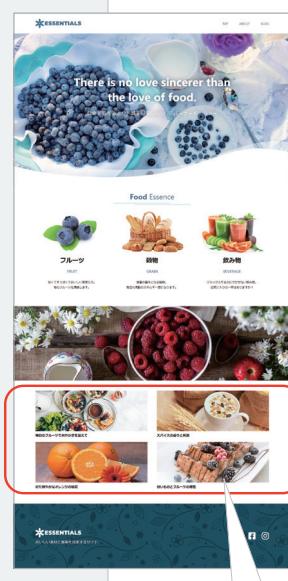
src/pages/index.js

`@imgix/gatsby`

<https://github.com/imgix/gatsby>

より詳しい使い方については、公式のドキュメントを参照してください。また、簡単な使い方についてはこちらでも解説しています。

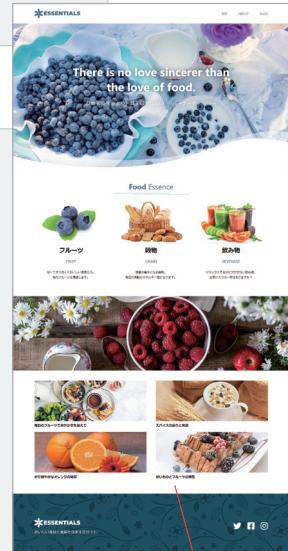
<https://ebisu.com/note/microcms-imgix-gatsby/>



```
<figure>
  <div data-gatsby-image-wrapper="" class="gatsby-image-wrapper gatsby-image-wrapper-constrained" style="height: 100%;">...  ...</div>
</figure>
```

最後に Contentful へのクエリを削除して、トップページ (index.js) の対応は完了です。

```
...
export const query = graphql`  
query {  
  ...
    pattern: file(relativePath: { eq: "pattern.jpg" }) {  
      childImageSharp {  
        fluid(maxWidth: 1920, quality: 90) {  
          ...GatsbyImageSharpFluid_withWebp  
        }
      }
    }
    allContentfulBlogPost(
      sort: { order: DESC, fields: publishDate }
      skip: 0
      limit: 4
    ) {
      ...
    }
    allMicrocmsBlog(
      sort: { order: DESC, fields: publishDate }
      skip: 0
      limit: 4
    ) {
      ...
    }
  }
}
```



```
...
export const query = graphql`  
query {  
  ...
    pattern: file(relativePath: { eq: "pattern.jpg" }) {  
      childImageSharp {  
        fluid(maxWidth: 1920, quality: 90) {  
          ...GatsbyImageSharpFluid_withWebp  
        }
      }
    }
    allMicrocmsBlog(
      sort: { order: DESC, fields: publishDate }
      skip: 0
      limit: 4
    ) {
      ...
    }
  }
}
```

src/pages/index.js

Contentfulのクエリを削除しても表示には影響しません。

2-2 記事一覧ページ

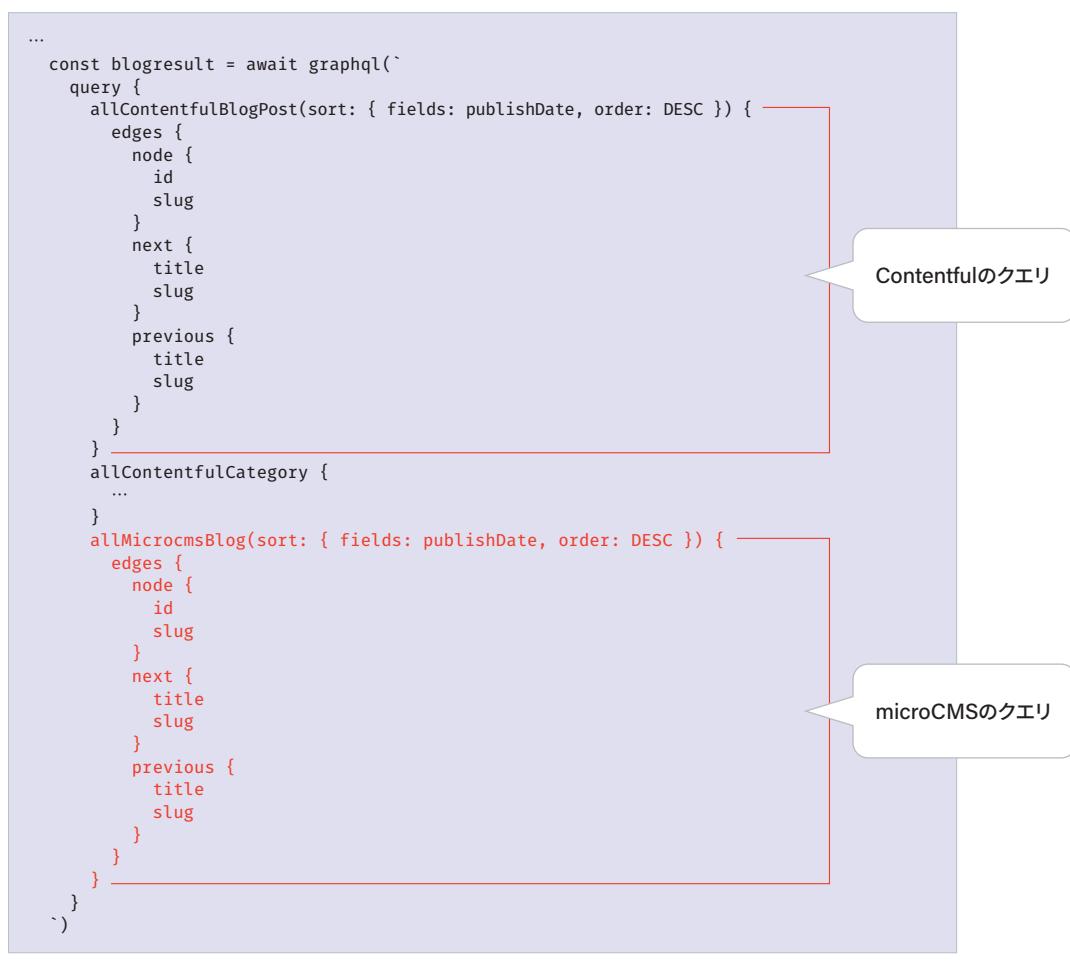
記事一覧ページ（blog-template.js）を microCMS に対応させます。

クエリを追加する

まずは gatsby-node.js を開き、context で blog-template.js へ送るデータのクエリを追加します。

```
...
const blogresult = await graphql(`  
query {  
  allContentfulBlogPost(sort: { fields: publishDate, order: DESC }) {  
    edges {  
      node {  
        id  
        slug  
      }  
      next {  
        title  
        slug  
      }  
      previous {  
        title  
        slug  
      }  
    }  
  }  
  allContentfulCategory {  
    ...  
  }  
  allMicrocmsBlog(sort: { fields: publishDate, order: DESC }) {  
    edges {  
      node {  
        id  
        slug  
      }  
      next {  
        title  
        slug  
      }  
      previous {  
        title  
        slug  
      }  
    }  
  }  
}`)

```



Contentfulのクエリ

microCMSのクエリ

gatsby-node.js

追加したクエリに合わせて、blog-template.js を呼び出している createPage 周辺を対応させます。ここでは allContentfulBlogPost を allMicrocmsBlog に変更しています。

```
const blogPostsPerPage = 6 // 1ページに表示する記事の数
const blogPosts = blogresult.data.allContentfulBlogPost.edges.length // 記事の総数
...
```

```
const blogPostsPerPage = 6 // 1ページに表示する記事の数
const blogPosts = blogresult.data.allMicrocmsBlog.edges.length // 記事の総数
const blogPages = Math.ceil(blogPosts / blogPostsPerPage) // 記事一覧ページの総数

Array.from({ length: blogPages }).forEach((_, i) => {
  createPage({
    path: i === 0 ? `/blog/` : `/blog/${i + 1}/`,
    component: path.resolve("./src/templates/blog-template.js"),
    context: {
      ...
    },
  })
})
```

gatsby-node.js

microCMSからのデータに置き換える

blog-template.js は index.js と同様に microCMS からデータを取得するクエリに書き換えます。

```
export const query = graphql`  
query($skip: Int!, $limit: Int!) {  
  allContentfulBlogPost {  
    sort: { order: DESC, fields: publishDate }  
    skip: $skip  
    limit: $limit  
  } {  
    edges {  
      node {  
        title  
        id  
        slug  
        eyecatch {  
          gatsbyImageData(width: 500, layout:  
CONSTRINED)  
          description  
        }  
      }  
    }  
  }  
}
```

```
export const query = graphql`  
query($skip: Int!, $limit: Int!) {  
  allMicrocmsBlog {  
    sort: { order: DESC, fields: publishDate }  
    skip: $skip  
    limit: $limit  
  } {  
    edges {  
      node {  
        title  
        id  
        slug  
        eyecatch {  
          url  
          width  
          height  
        }  
      }  
    }  
  }  
}
```

src/templates/blog-template.js

そして、GatsbyImage を利用していた部分を ImgixGatsbyImage を利用する形に書き換えます。

```
const BlogTemp = ({ data, location, pageContext }) => (
  ...
  <section className="content bloglist">
    <div className="container">
      <h1 className="bar">RECENT POSTS</h1>
      <div className="posts">
        {data.allContentfulBlogPost.edges.map(({ node }) => (
          <article className="post" key={node.id}>
            <Link to={`/blog/post/${node.slug}`}>
              <figure>
                <GatsbyImage
                  image={node.eyecatch.gatsbyImageData}
                  alt={node.eyecatch.description}
                  style={{ height: "100%" }}
                />
              </figure>
              <h3>{node.title}</h3>
            </Link>
          ...
        ))
      
```

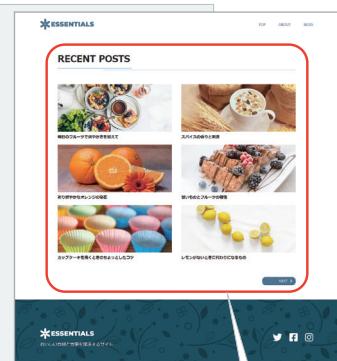
他にImgを使用している箇所はないので削除。

```
import { GatsbyImage } from "gatsby-plugin-image"
...
import { ImgixGatsbyImage } from '@imgix/gatsby'

const BlogTemp = ({ data, location, pageContext }) => (
  ...
  <section className="content bloglist">
    <div className="container">
      <h1 className="bar">RECENT POSTS</h1>
      <div className="posts">
        {data.allMicrocmsBlog.edges.map(({ node }) => (
          <article className="post" key={node.id}>
            <Link to={`/blog/post/${node.slug}`}>
              <figure>
                <ImgixGatsbyImage
                  src={node.eyecatch.url}
                  imgixParams={{ auto: ["format", "compress"] }}
                  layout="constrained"
                  width={500}
                  sourceWidth={node.eyecatch.width}
                  sourceHeight={node.eyecatch.height}
                  style={{ height: "100%" }}
                />
              </figure>
              <h3>{node.title}</h3>
            </Link>
          ...
        ))
      
```

src/templates/blog-template.js

以上で、対応は完了です。



```
<figure>
  <div data-gatsby-image-wrapper="" class="gatsby-image-wrapper gatsby-image-wrapper-constrained" style="height: 100%; ">... ...</div>
</figure>
```

microCMSからのデータに置き換わります。

2-3 カテゴリーページ

カテゴリーページ (cat-template.js) を microCMS に対応させます。

クエリを追加する

カテゴリーページで問題となるのが、各カテゴリーに属する記事の数です。取得したデータからカウントしてもよいのですが、ここでは GraphQL の group を利用します。

GraphiQL で次のようなクエリを作成すると、各カテゴリーに属するポストの数が取得できます。

The screenshot shows the GraphiQL interface with the following components:

- Explorer** sidebar on the left containing filters like `allMicrocmsBlog`, `group`, `field*`, and `totalCount`.
- GraphiQL** main area with the following query:

```
query MyQuery {
  allMicrocmsBlog {
    group(field: category__categorySlug) {
      fieldValue
      totalCount
    }
  }
}
```

- Prettify**, **History**, **Explorer**, **Code Exporter**, and **Docs** buttons at the top right.
- Result** pane on the right displaying the JSON response:

```
{
  "data": {
    "allMicrocmsBlog": [
      {
        "group": [
          {
            "fieldValue": "beverage",
            "totalCount": 4
          },
          {
            "fieldValue": "fruit",
            "totalCount": 8
          },
          {
            "fieldValue": "grain",
            "totalCount": 7
          }
        ]
      }
    ],
    "extensions": {}
  }
}
```

そこで、このクエリと、`allMicrocmsCategory` からデータを取得するためのクエリを `gatsby-node.js` に追加します。

```
...
const blogresult = await graphql(`query {
  ...
  allMicrocmsBlog(sort: { fields: publishDate, order: DESC }) {
    edges {
      ...
    }
    group(field: category__categorySlug) {
      totalCount
      fieldValue
    }
  }
  allMicrocmsCategory {
    nodes {
      category
      categorySlug
      categoryId
    }
  }
}`)
`)
```

gatsby-node.js

そして、このクエリから得られるデータを利用して `createPage` 周辺を対応させます。

```
blogresult.data.allContentfulCategory.edges.forEach(({ node }) => {
  const catPostsPerPage = 6 // 1ページに表示する記事の数
  const catPosts = node.blogpost.length // カテゴリーに属した記事の総数
  const catPages = Math.ceil(catPosts / catPostsPerPage) // カテゴリーページの総数

  Array.from({ length: catPages }).forEach(_, i) => {
    createPage({
      path:
        i === 0
          ? `/cat/${node.categorySlug}/`
          : `/cat/${node.categorySlug}/${i + 1}/`,
      component: path.resolve(`./src/templates/cat-template.js`),
      context: {
        catid: node.id,
        catname: node.category,
        catslug: node.categorySlug,
        skip: catPostsPerPage * i,
        limit: catPostsPerPage,
        currentPage: i + 1, // 現在のページ番号
        isFirst: i + 1 === 1, // 最初のページ
        isLast: i + 1 === catPages, // 最後のページ
      },
    })
  }
})
```

```

blogresult.data.allMicrocmsBlog.group.forEach(node => {
  const catPostsPerPage = 6 // 1ページに表示する記事の数
  const catPosts = node.totalCount // カテゴリーに属した記事の総数
  const catPages = Math.ceil(catPosts / catPostsPerPage) // カテゴリーページの総数

  Array.from({ length: catPages }).forEach((_, i) => {
    createPage({
      path:
        i === 0
          ? `/cat/${node.fieldValue}`
          : `/cat/${node.fieldValue}/${i + 1}/`,
      component: path.resolve(`./src/templates/cat-template.js`),
      context: {
        catid: blogresult.data.allMicrocmsCategory.nodes.find(
          n => n.categorySlug === node.fieldValue
        ).categoryId,
        catname: blogresult.data.allMicrocmsCategory.nodes.find(
          n => n.categorySlug === node.fieldValue
        ).category,
        catslug: node.fieldValue,
        skip: catPostsPerPage * i,
        limit: catPostsPerPage,
        currentPage: i + 1, // 現在のページ番号
        isFirst: i + 1 === 1, // 最初のページ
        isLast: i + 1 === catPages, // 最後のページ
      },
    })
  })
})

```

gatsby-node.js

microCMSからのデータに置き換える

`cat-template.js` のクエリも、`microCMS` へのクエリに書き換えます。

```
export const query = gql`  
  query($catid: String!, $skip: Int!, $limit: Int!) {  
    allContentfulBlogPost(  
      sort: { order: DESC, fields: publishDate }  
      ...  
    ) {  
      edges {  
        node {  
          title  
          id  
          slug  
          eyecatch {  
            gatsbyImageData(width: 500, layout:  
CONSTRINED)  
            description  
          }  
        }  
      }  
    }  
  }`
```

```
export const query = gql`  
  query($catid: String!, $skip: Int!, $limit: Int!) {  
    allMicrocmsBlog(  
      sort: { order: DESC, fields: publishDate }  
      ...  
    ) {  
      edges {  
        node {  
          title  
          id  
          slug  
          eyecatch {  
            url  
            width  
            height  
          }  
        }  
      }  
    }  
  }`
```

src/templates/cat-template.js

GatsbyImage を利用していた部分を ImgixGatsbyImage を利用する形に書き換えます。

```
const CatTemp = ({ data, location, pageContext }) => (
  ...
  <section className="content bloglist">
    <div className="container">
      <h1 className="bar">CATEGORY: {pageContext.catname}</h1>
      <div className="posts">
        {data.allContentfulBlogPost.edges.map(({ node }) => (
          <article className="post" key={node.id}>
            <Link to={`/blog/post/${node.slug}`}>
              <figure>
                <GatsbyImage
                  image={node.eyecatch.gatsbyImageData}
                  alt={node.eyecatch.description}
                  style={{ height: "100%" }}
                />
              </figure>
              <h3>{node.title}</h3>
            </Link>
          ...
        ))
      </div>
    </div>
  </section>
)
```

他にImgを使用している箇所はないので削除。

```
import { GatsbyImage } from "gatsby-plugin-image"
...
import { ImgixGatsbyImage } from '@imgix/gatsby'

const CatTemp = ({ data, location, pageContext }) => (
  ...
  <section className="content bloglist">
    <div className="container">
      <h1 className="bar">CATEGORY: {pageContext.catname}</h1>
      <div className="posts">
        {data.allMicrocmsBlog.edges.map(({ node }) => (
          <article className="post" key={node.id}>
            <Link to={`/blog/post/${node.slug}`}>
              <figure>
                <ImgixGatsbyImage
                  src={node.eyecatch.url}
                  imgixParams={{ auto: ["format", "compress"] }}
                  layout="constrained"
                  width={500}
                  sourceWidth={node.eyecatch.width}
                  sourceHeight={node.eyecatch.height}
                  style={{ height: "100%" }}
                />
              </figure>
              <h3>{node.title}</h3>
            </Link>
          ...
        ))
      </div>
    </div>
  </section>
)
```

src/templates/cat-template.js



```
<figure>
<div data-gatsby-image-wrapper="" class="gatsby-image-wrapper gatsby-image-wrapper-constrained" style="height: 100%; "></div>
</figure>
```

microCMSからのデータに置き換わります。

microCMS

3

記事

3-1 記事ページ

3-2 リッチテキストの処理

3-3 仕上げ

Build blazing-fast websites with GatsbyJS

GatsbyJS



3-1 記事ページ

記事ページ (blogpost-template.js) を microCMS に対応させます。



microCMSからのデータに置き換える

まずは、blogpost-template.js を呼び出している gatsby-node.js の createPage 周辺を対応させます。ここでは allContentfulBlogPost を allMicrocmsBlog に変更しています。

```
blogresult.data.allContentfulBlogPost.edges.forEach(
  ({ node, next, previous }) => {
  ...
)
```



```
blogresult.data.allMicrocmsBlog.edges.forEach(
  ({ node, next, previous }) => {
    createPage({
      path: `/blog/post/${node.slug}/`,
      component: path.resolve(`./src/templates/blogpost-template.js`),
      context: {
        id: node.id,
        next,
        previous,
      },
    })
  }
)
```

gatsby-node.js

blogpost-template.js を開き、microCMS へのクエリに書き換えます。

```
export const query = graphql`  
query($id: String!) {  
  contentfulBlogPost(id: { eq: $id }) {  
    title  
    publishDateJP: publishDate(...)  
    publishDate  
    category {  
      category  
      categorySlug  
      id  
    }  
    eyecatch {  
      gatsbyImageData(layout: FULL_WIDTH)  
      description  
      file {  
        details {  
          image {  
            width  
            height  
          }  
        }  
        url  
      }  
    }  
    content {  
      raw  
      references {  
        ... on ContentfulAsset {  
          contentful_id  
          __typename  
          gatsbyImageData(layout: FULL_WIDTH)  
          title  
          description  
        }  
      }  
    }  
  }  
}
```



```
export const query = graphql`  
query($id: String!) {  
  microcmsBlog(id: { eq: $id }) {  
    title  
    publishDateJP: publishDate(formatString:  
"YYYY 年 MM 月 DD 日")  
    publishDate  
    category {  
      category  
      categorySlug  
      id  
    }  
    eyecatch {  
      url  
      width  
      height  
    }  
    content  
  }  
}
```

src/templates/blogpost-template.js

あとは、上から順に対応していきます。

メタデータ

メタデータ <Seo /> の中身を microcmsBlog から取得したデータに書き換えます。

ただし、pagedesc（説明）はリッチテキストの HTML から用意しなければなりません。ここでは html-to-text を利用してテキストを用意します。

html-to-text

<https://github.com/werk85/node-html-to-text>

```
$ yarn add html-to-text
```

インストールしたら convert を import し、pagedesc を次のように書き換えます。

```
const BlogpostTemp = ({ data, pageContext, location }) => (
  <Layout>
    <Seo
      pagetitle={data.contentfulBlogPost.title}
      pagedesc={`${documentToString(
        data.contentfulBlogPost.content.json
      )}.slice(0, 70)}`}
      pagepath={location.pathname}
      blogImg={"https:${data.contentfulBlogPost.eyecatch.file.url}"}
      pageimgw={data.contentfulBlogPost.eyecatch.file.details.image.width}
      pageimgh={data.contentfulBlogPost.eyecatch.file.details.image.height}
    />
```



```
import { convert } from "html-to-text"
...
const BlogpostTemp = ({ data, pageContext, location }) => (
  <Layout>
    <Seo
      pagetitle={data.microcmsBlog.title}
      pagedesc={`${convert(data.microcmsBlog.content, {
        selectors: [
          { selector: 'a', options: { ignoreHref: true } },
          { selector: 'img', format: 'skip' }
        ]
      })}.slice(0, 70)`}
      pagepath={location.pathname}
      blogImg={data.microcmsBlog.eyecatch.url}
      pageimgw={data.microcmsBlog.eyecatch.width}
      pageimgh={data.microcmsBlog.eyecatch.height}
    />
```

src/templates/blogpost-template.js

アイキャッチ画像

アイキャッチ画像は ImgixGatsbyImage を利用する形に書き換えます。

```
<div className="eyecatch">
  <figure>
    <GatsbyImage
      image={data.contentfulBlogPost.eyecatch.gatsbyImageData}
      alt={data.contentfulBlogPost.eyecatch.description}
    />
  </figure>
</div>
```



```

import { ImgixGatsbyImage } from '@imgix/gatsby'
...
<div className="eyecatch">
  <figure>
    <ImgixGatsbyImage
      src={node.eyecatch.url}
      imgixParams={{ auto: ["format", "compress"] }}
      layout="fullWidth"
      sourceWidth={node.eyecatch.width}
      sourceHeight={node.eyecatch.height}
    />
  </figure>
</div>

```

src/templates/blogpost-template.js

記事

<article> の部分は、記事のタイトル、投稿日、カテゴリーを microcmsBlog から取得したデータを使うように書き換えます。

記事の本文 <div className="postbody"> は、取得した HTML をそのまま表示するように、dangerouslySetInnerHTML で書き換えています。

```

<article className="content">
  <div className="container">
    <h1 className="bar">{data.contentfulBlogPost.title}</h1>
    <aside className="info">
      <time dateTime={data.contentfulBlogPost.publishDate}>
        <FontAwesomeIcon icon={faClock} />
        {data.contentfulBlogPost.publishDateJP}
      </time>
      <div className="cat">
        <FontAwesomeIcon icon={faFolderOpen} />
      <ul>
        {data.contentfulBlogPost.category.map(cat => (
          <li className={cat.categorySlug} key={cat.id}>
            <Link to={`/cat/${cat.categorySlug}/`}>{cat.category}</Link>
          </li>
        ))}
      </ul>
    </div>
  </aside>
  <div className="postbody">
    {documentToReactComponents(
      data.contentfulBlogPost.content.json,
      options
    )}
  </div>
  ...

```

```

<article className="content">
  <div className="container">
    <h1 className="bar">{data.microcmsBlog.title}</h1>
    <aside className="info">
      <time dateTime={data.microcmsBlog.publishDate}>
        <FontAwesomeIcon icon={faClock} />
        {data.microcmsBlog.publishDateJP}
      </time>
    </aside>
    <div className="cat">
      <FontAwesomeIcon icon={faFolderOpen} />
      <ul>
        {data.microcmsBlog.category.map(cat => (
          <li className={cat.categorySlug} key={cat.id}>
            <Link to={`/cat/${cat.categorySlug}/`}>{cat.category}</Link>
          </li>
        ))}
      </ul>
    </div>
  </div>
  <div className="postbody"
    dangerouslySetInnerHTML={{ __html: data.microcmsBlog.content }}>
  </div>
...

```

src/templates/blogpost-template.js

これで、ページが表示されるようになります。



```

<figure>
<div data-gatsby-image-wrapper="" class="gatsby-image-wrapper">
  ... 
</div>
</figure>

```

```

<article className="content">
  <div className="container">
    <h1 className="bar">毎日のフルーツで爽やかさを加えて </h1>
    ...
    <div className="postbody">
      <p>フルーツには適度な甘みと酸味と爽やかさがあって、… </p>
      <h2 id="h52d275bcba">フルーツの旬を活かす </h2>
      <p>…</p>
       ...
    </div>
    ...
  </div>

```

3-2 リッチテキストの処理

リッチテキストの HTML をそのまま表示するだけでは、**3-1**までの設定で十分です。しかし、見出しがアイコンを付けたり、記事中の画像をレスポンシブイメージにする場合には、Contentful のときと同様に htmlAST に変換した上で処理するのが簡単です。

そこで、unified を利用した処理を追加します。以下のようにライブラリをインストールしてください。

```
$ yarn add unified@9.2.2 rehype-parse@7.0.1 rehype-react@6.2.0
```

バージョンを指定してインストールしていますが、unified の新しいバージョンでは、ESModules のみの対応となっているためです。ページ側で利用する限りは問題がないのですが、Gatsby API files と呼ばれる設定ファイル（gatsby-node.js や gatsby-config.js など）では現時点では面倒なことになるため、Gatsby の公式プラグインに合わせたバージョンを指定しています。

レスポンシブイメージの設定には、画像の横幅と高さのデータが必要です。リッチエディタの content フィールドの詳細設定で「画像のレスポンスに width と height を含む」がオンになっていることを確認しておきます。



「プロックコンテンツ>API設定>APIスキーマ>contentフィールドの詳細設定」を選択。



インポートして、htmlASTへの変換処理を追加します。HTMLをパースしてhtmlASTにするシンプルな物ですが、fragment: trueを忘れるとなにかが付いたhtmlASTに変換されますので注意が必要です。

```
import unified from "unified"
import parse from "rehype-parse"
import rehypeReact from "rehype-react"
...
const BlogpostTemp = ({ data, pageContext, location }) => {
  const htmlAst = unified()
    .use(parse, { fragment: true })
    .parse(data.microcmsBlog.content)

  return (
    <Layout>
    ...
    </Layout>
  )
}
```



注意

src/templates/blogpost-template.js

続いて、htmlASTをレンダリングする処理を用意します。

```
...
const renderAst = new rehypeReact({
  createElement: React.createElement,
  Fragment: React.Fragment,
  components: {},
}).Compiler

const BlogpostTemp = ({ data, pageContext, location }) => {
  const htmlAst = unified()
    .use(parse, { fragment: true })
    .parse(data.microcmsBlog.content)
  ...
}
```

src/templates/blogpost-template.js

そして、その処理を利用して表示します。

```
<div
  className="postbody"
  dangerouslySetInnerHTML={{ __html: data.microcmsBlog.content }}
></div>
```



```
<div className="postbody">{renderAst(htmlAst)}</div>
```

src/templates/blogpost-template.js

この段階では出力される HTML は変化しません。

```


フルーツには適度な甘みと酸味と爽やかさがあって、毎日楽で食べきせん。バケツフルートの柑橘もおすすめです。新たにたくさんのフルーツを取り入れてみると、いつの間にかくらのフルーツを食べるようになっています。



## フルーツの旬を活かす



野菜と共にフルーツにも拘ります。ただ、地元によって旬の時期には違いがありますので、野菜地図の旬を活かしていただけます。



小さな粒に丁寧な粒のラグリーブルーベリーです。これらの季節は本当に美味しいことも、野菜させてデータとして覚えておくことができます。少し自分で確認ができます。また、ページからなるフルーツ一覧も作成されています。ぜひチェックです。今後はまたフルーツ情報を更新します。次回またそのお話を述べたいです。



パンの嗜みが付いたいつも美味しいパンだ。特にコク味のパンでとても喜んでいました！ クリームパンとミニパン、パンダ、バターミニ食パンをさくらんぼパンに食べられます。


```

あとは、Contentful のときと同様に、options を参考にして `<h2>` と `` への処理を追加します。

```

const renderAst = newrehypeReact({
  createElement: React.createElement,
  Fragment: React.Fragment,
  components: {
    h2: props => {
      return (
        <h2>
          <FontAwesomeIcon icon={faCheckSquare} />
          {props.children}
        </h2>
      )
    },
    img: props => {
      return (
        <ImgixGatsbyImage
          src={props.src}
          imgixParams={{ auto: ["format", "compress"] }}
          layout="constrained"
          width={785}
          sourceWidth={props.width}
          sourceHeight={props.height}
        />
      )
    },
  },
})

```

リッチテキスト内の見出し `<h2>` に Font Awesome のアイコンを追加。

リッチテキスト内の画像 `` をレスポンシブイメージに設定。

src/templates/blogpost-template.js

最後に、options や Contentful 関連の設定を削除して、リッチテキストの設定は完了です。

```

import React from "react"
import { graphql, Link } from "gatsby"
import { GatsbyImage } from "gatsby-plugin-image" 削除
...
import { renderRichText } from "gatsby-source-contentful/rich-text" 削除
import { BLOCKS } from "@contentful/rich-text-types" 削除
import { documentToPlainTextString } from "@contentful/rich-text-plain-text-renderer" 削除
...
const options = {
  ...
}
const options = { 削除
}

const renderAst = newrehypeReact({
  ...
})

```

src/templates/blogpost-template.js

アイコンが表示されます。



レスポンシブイメージになります。

```

<div class="postbody">
  <p>フルーツには適度な甘みと酸味と爽やかさがあって、毎日食べても飽きません。ノンヨーグルトとの相性もよかったです。特に大きめのフルーツを取り入れてみると、いつか間に合うくらいのフレッシュ感があるよかったです。</p>
  <h2><img alt="check-square icon" data-icon="check-square" />フルーツの旬を活かす</h2>
  <p>…</p>
  <div data-gatsby-image-wrapper="">
    
  </div>
  ...
</p>
</div>

```

3-3 仕上げ

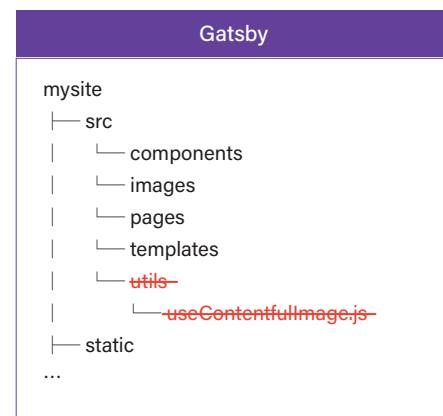
10 of 10

最後に、プロジェクト全体で必要なくなったライブラリやコンポーネント、クエリを削除していきます。まずは、`gatsby-node.js` に残っている Contentful へのクエリを削除します。`useContentfullImage.js` も忘れずに削除し、`gatsby-config.js` からは Contentful のソースプラグインの設定も削除します。

```
const blogresult = await graphql(`\n  query {\n    allContentfulBlogPost(sort: { ... DESC }) {\n      ...  
    }\n    allContentfulCategory {  
      ...  
    }\n    allMicrocmsBlog(sort: { ... DESC }) {  
      ...  
    }  
  }  
`)
```

```
  `gatsby-plugin-offline`,  
  {  
    resolve: `gatsby-source-contentful`,  
    options: {  
      spaceId: process.env.CONTENTFUL_SPACE_ID,  
      accessToken: process.env.CONTENTFUL_ACCESS_TOKEN,  
      host: process.env.CONTENTFUL_HOST,  
    },  
  },  
  {  
    resolve: "gatsby-source-microcms",  
  },  
  {  
    resolve: "gatsby-source-filesystem",  
    options: {  
      name: "content",  
      path: `${__dirname}/content`,  
    },  
  },  
  {  
    resolve: "gatsby-transformer-remark",  
  },  
  {  
    resolve: "gatsby-plugin-react-helmet",  
  },  
  {  
    resolve: "gatsby-plugin-sharp",  
  },  
  {  
    resolve: "gatsby-plugin-sass",  
  },  
  {  
    resolve: "gatsby-plugin-styled-components",  
  },  
  {  
    resolve: "gatsby-plugin-manifest",  
    options: {  
      name: "Gatsby Contentful MicroCMS",  
      short_name: "Contentful",  
      start_url: "/index.html",  
      background_color: "#fff",  
      theme_color: "#000",  
      display: "minimal-ui",  
      icon: "src/assets/icon.png",  
    },  
  },  
  {  
    resolve: "gatsby-plugin-offline",  
  },  
},  
{  
  resolve: "gatsby-plugin-sitemap",  
},  
{  
  resolve: "gatsby-plugin-google-tagmanager",  
},  
{  
  resolve: "gatsby-plugin-react-helmet",  
},  
{  
  resolve: "gatsby-plugin-sass",  
},  
{  
  resolve: "gatsby-plugin-styled-components",  
},  
{  
  resolve: "gatsby-plugin-manifest",  
},  
{  
  resolve: "gatsby-plugin-offline",  
},  
],  

```



設定を削除したら、ソースプラグインをアンインストールします（アンインストールしないと、ビルトでエラーになる場合があります）。

```
$ yarn remove gatsby-source-contentful
```

以上で、microCMS に対する設定は完了です。

microCMSによるサイトの更新

microCMSでの変更を Netlify に反映させるためには、Webhook を利用します。

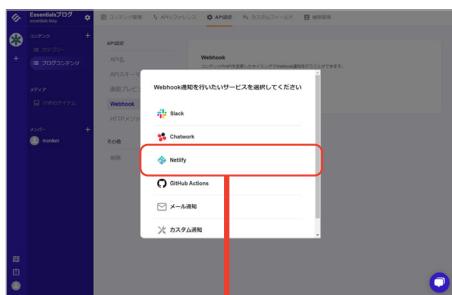
「ブログコンテンツ」APIを選択。



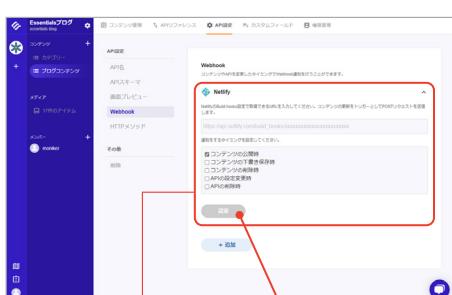
「API設定」を選択。

「Webhook」を選択。

「追加」をクリック。



「Netlify」を選択。



NetlifyのBuild hook
のURLなどを指定。

「設定」をクリック。

Webhook の設定は、API ごとに「API 設定 > Webhook」で行います。新規に追加する場合は「追加」をクリックします。

サービスの選択肢が表示されますので、「Netlify」を選択します。

Netlify で取得した Build hook(ビルドフック) の URL と、Netlify でデプロイが行われるトリガー（通知するタイミング）を指定します。

Build hook についてはセットアップ PDF の 3-5 を参照してください。

「設定」をクリックすれば完了です。

microCMS

APPENDIX

`GatsbyImage` を使う

Build blazing-fast websites with GatsbyJS

GatsbyJS

A

createRemoteFileNode



gatsby-source-filesystem の Helper functions である createRemoteFileNode を利用することで、リモートにある画像をダウンロードしてローカルの画像のように GatsbyImage で扱うことができます。

この機能を利用することで、microCMS ではなく、デプロイ先から画像をダウンロードするようにできます。

これまで、createRemoteFileNode は、onCreateNode や createResolvers など、どこで使うかの制限は特にありませんでした。しかし、Gatsby v4 では node の作成は sourceNodes で行うのが推奨となりましたので、ここではそれに習って進めます。

2. Data mutations need to happen during sourceNodes or onCreateNode

<https://v4.gatsbyjs.com/docs/reference/release-notes/migrating-source-plugin-from-v3-to-v4/#2-data-mutations-need-to-happen-during-sourcenodes-or-oncreatenode>

eyecatch画像

まずは、eyecatch 画像の設定を行います。gatsby-node.js に以下のソースを追加します。

```
const { createRemoteFileNode } = require(`gatsby-source-filesystem`)

exports.createSchemaCustomization = ({ actions }) => {
  const { createTypes } = actions
  createTypes(`
    type MicrocmsBlog implements Node {
      eyecatchimg: File @link
    }
  `)
}

exports.sourceNodes = async ({  

  store,  

  cache,  

  actions: { createNode, createNodeField },  

  createNodeId,  

  createContentDigest,  

  getNodesByType,  

}) => {  

  const MicrocmsBlogNodes = getNodesByType(`MicrocmsBlog`)  

  MicrocmsBlogNodes.map(async node => {
    const fileNode = await createRemoteFileNode({  

      url: `${node.eyecatch.url}?q=100`,  

      parentNodeId: node.id,  

      cache,  

      store,  

      createNode: createNode,  

      createNodeId: createNodeId,  

    })  

    if (fileNode) {
      node.eyecatchimg = fileNode.id
    }
  })
}
```

gatsby-node.js

1

2

3

順に見ていきます。

- createRemoteFileNode を require しています。

```
const { createRemoteFileNode } = require(`gatsby-source-filesystem`)
```

- 登録した画像は File node で扱われます。これを、MicrocmsBlog から扱えるようにするためのリンク先として eyecatchimg という node を用意しています。

```
exports.createSchemaCustomization = ({ actions }) => {
  const { createTypes } = actions
  createTypes(`type MicrocmsBlog implements Node {
    eyecatchimg: File @link
  }`)
```

- Gatsby のライフサイクルの sourceNodes の中で eyecatch 画像を追加しています。

まず、Helper functions の getNodesByType を使って MicrocmsBlog のデータを取得します。これで、allMicrocmsBlog からデータを取得したのと同等のデータが取得できますので、MicrocmsBlog.eyecatch.url を使って、createRemoteFileNode で登録しています。microCMS からは最高画質でダウンロードするように指定しています。

さらに、登録したことでできたノードを②で用意した eyecatchimg に接続しています。

```
exports.sourceNodes = async ({
  store,
  cache,
  actions: { createNode, createNodeField },
  createNodeId,
  getNodesByType,
}) => {
  const MicrocmsBlogNodes = getNodesByType(`MicrocmsBlog`)
  MicrocmsBlogNodes.map(async node => {
    const fileNode = await createRemoteFileNode({
      url: `${node.eyecatch.url}?q=100`,
      parentNodeId: node.id,
    })
    if (fileNode) {
      node.eyecatchimg = fileNode.id
    }
  })
}
```

ここで、MicrocmsBlog の中に eyecatchimg が追加され、アイキャッチの GatsbyImage に関するデータが取得できるようになります。

The screenshot shows the GraphQL Explorer interface with the following components:

- Explorer Tab:** Shows the schema structure. A red box highlights the `eyecatching` field under `microcmsBlog`.
- GraphiQL Tab:** Displays the raw GraphQL query and its results.
- Result Panel:** Shows the JSON response from the query.

Schema (Explorer Tab):

```

query MyQuery {
  microcmsBlog {
    eyecatching {
      childImageSharp {
        gatsbyImageData
      }
    }
  }
}
  
```

Query (GraphiQL Tab):

```

query MyQuery {
  microcmsBlog {
    eyecatching {
      childImageSharp {
        gatsbyImageData
      }
    }
  }
}
  
```

Result (GraphiQL Tab):

```

{
  "data": {
    "microcmsBlog": {
      "eyecatching": {
        "childImageSharp": {
          "gatsbyImageData": {
            "layout": "constrained",
            "backgroundColor": "#f8f8f8",
            "images": [
              {
                "fallback": {
                  "src": "/static/fd841b93eb2537f8e3af4df457777646/95519/everyday.jpg",
                  "srcSet": "/static/fd841b93eb2537f8e3af4df457777646/def50/everyday.jpg 400w, \n/static/fd841b93eb2537f8e3af4df457777646/8836c/everyday.jpg 800w, \n/static/fd841b93eb2537f8e3af4df457777646/95519/everyday.jpg 1600w",
                  "sizes": "(min-width: 1600px) 1600px, 100vw"
                },
                "sources": [
                  {
                    "srcSet": "/static/fd841b93eb2537f8e3af4df457777646/4d72d/everyday.webp 400w, \n/static/fd841b93eb2537f8e3af4df457777646/fb842/everyday.webp 800w, \n/static/fd841b93eb2537f8e3af4df457777646/970e2/everyday.webp 1600w",
                    "type": "image/webp",
                    "sizes": "(min-width: 1600px) 1600px, 100vw"
                  }
                ],
                "width": 1600,
                "height": 661
              }
            ],
            "extensions": {}
          }
        }
      }
    }
  }
}
  
```

記事内の画像

さらに、記事内の画像も追加してみましょう。記事内の画像の URL の抽出方法は色々と考えられますが、今回は unified を使ってみます。

まずは、追加のライブラリをインストールします。

```
$ yarn add unist-util-visit@2.0.3
```

続いて、今回使うライブラリーを require しておきます。また、記事の中で microCMS からの画像だけを区別するための URL を指定しておきます。

```
const unified = require("unified")
const parse = require("rehype-parse")
const visit = require("unist-util-visit")

const url = "https://images.microcms-assets.io"
```

gatsby-node.js

続いて、sourceNodes に処理を追加します。

```
exports.sourceNodes = async ({
  store,
  cache,
  actions: { createNode, createNodeField },
  createNodeId,
  getNodesByType,
}) => {
  const MicrocmsBlogNodes = getNodesByType(`MicrocmsBlog`)

  MicrocmsBlogNodes.map(async node => {
    const fileNode = await createRemoteFileNode({
      url: `${node.eyecatch.url}?q=100`,
      parentNodeId: node.id,
      cache,
      store,
      createNode: createNode,
      createNodeId: createNodeId,
    })
  })
}
```

※次ページに続きます。

※前ページからの続きです。

```

if (fileNode) {
  await createNodeField({
    node: fileNode,
    name: "imageType",
    value: "eyecatch",
  })
  await createNodeField({
    node: fileNode,
    name: "url",
    value: node.eyecatch.url,
  })
  node.eyecatchimg = fileNode.id
}

const htmlAst = unified().use(parse, { fragment: true }).parse(node.content)
let imglist = [] 4

visit(htmlAst, "element", async (hastnode, index, parent) => {
  if (
    hastnode.tagName === "img" &&
    hastnode.properties.src.indexOf(url) !== -1
  ) {
    imglist.push(hastnode.properties.src)
  }
}) 5

if (imglist.length) {
  // console.log(imglist)

  imglist.forEach(async url => {
    const inlineFileNode = await createRemoteFileNode({
      url,
      parentNodeId: node.id,
      cache,
      store,
      createNode: createNode,
      createNodeId: createNodeId,
    })

    await createNodeField({ 6
      node: inlineFileNode,
      name: "imageType",
      value: "inline",
    })

    await createNodeField({ 7
      node: inlineFileNode,
      name: "url",
      value: url,
    })
  })
}
}

```

gatsby-node.js

順に見ていきます。

- 4 P.44 と同じようにして、記事データの HTML を htmlAST へと変換します。
また、抽出した URL を保存しておく配列も準備しています。

```
const htmlAst = unified().use(parse, { fragment: true }).parse(node.content)
let imglist = []
```

- 5 htmlAST から特定のエレメントを探す visit を使っています。img タグでその src プロパティに設定した URL (<https://images.microcms-assets.io>) が含まれているものを見つかった時に imglist に追加しています。これで、URL の抽出が完了します。

```
visit(htmlAst, "element", async (hastnode, index, parent) => {
  if (
    hastnode.tagName === "img" &&
    hastnode.properties.src.indexOf(url) != -1
  ) {
    imglist.push(hastnode.properties.src)
  }
})
```

- 6 URL が抽出された場合 (imglist が空でない場合)、その配列を使って createRemoteFileNode で画像を登録しています。

```
if (imglist.length) {
  // console.log(imglist)

  imglist.forEach(async url => {
    const inlineFileNode = await createRemoteFileNode({
      url,
      parentNodeId: node.id,
      cache,
      store,
      createNode: createNode,
      createNodeId: createNodeId,
    })
    ...
  })
}
```

7

記事中の画像を取得しやすくするために、メタデータを追加しています。

ここでは、imageType と url を追加しています。

```
await createNodeField({
  node: inlineFileNode,
  name: "imageType",
  value: "inline",
})

await createNodeField({
  node: inlineFileNode,
  name: "url",
  value: url,
})
```

8

7 でメタデータを追加しただけでは、GraphQL でこのメタデータを取得することができません。これは、先に createRemoteFileNode で登録した eyecatch 画像にメタデータがついていないためです。そこで、eyecatch の方にもメタデータを追加します。

```
await createNodeField({
  node: fileNode,
  name: "imageType",
  value: "eyecatch",
})
await createNodeField({
  node: fileNode,
  name: "url",
  value: node.eyecatch.url,
})
```

以上で設定は完了です。

以下のようなクエリで記事中の画像の `gatsbyImageData` も取得できるようになりましたので、`url` を元にして記事中の `img` を `GatsbyImage` に置き換えることができるようになりました。

The screenshot shows the GraphQL Explorer interface with the following components:

- Explorer**: A sidebar containing a tree view of the schema. A red box highlights the `gatsbyImageData` field under the `allFile` node.
- GraphQL**: The main area where the query is entered. The query is:


```
query MyQuery {
  allFile(filter: {fields: {imageType: {eq: "inline"}}}) {
    nodes {
      fields {
        imageType
        url
      }
      childImageSharp {
        gatsbyImageData
      }
    }
  }
}
```
- Code Exporter**: A button at the top right to export the code.
- Docs**: A link at the top right.
- QUERY VARIABLES**: A section at the bottom left.
- Execution Results**: Two large boxes on the right side showing the expanded schema and the resulting JSON data.

Expanded Schema (Left Box):

```
query MyQuery {
  allFile(filter: {fields: {imageType: {eq: "inline"}}}) {
    nodes {
      fields {
        imageType
        url
      }
      childImageSharp {
        gatsbyImageData
      }
    }
  }
}
```

Resulting JSON Data (Right Box):

```
{
  "data": {
    "allFile": {
      "nodes": [
        {
          "fields": {
            "imageType": "inline",
            "url": "https://images.microcms-assets.io/protected/ap-northeast-1:b6f9665c-97b7-4807-bfac-f6dce328dbc3/service/essentials-blog/media/color.jpg"
          },
          "childImageSharp": {
            "gatsbyImageData": {
              "layout": "constrained",
              "backgroundColor": "...",
              "images": {
                ...
              },
              "width": 1600,
              "height": 800
            }
          }
        }
      ]
    }
  }
}
```

■著者**エビスコム**

<https://ebisu.com/>

さまざまなメディアにおける企画制作を世界各地のネットワークを駆使して展開。コンピュータ、インターネット関係では書籍、デジタル映像、CG、ソフトウェアの企画制作、WWWシステムの構築などを行う。

主な編著書：『作って学ぶ HTML & CSS モダンコーディング』マイナビ出版刊
『HTML5&CSS3 デザイン 現場の新標準ガイド【第2版】』同上
『CSS グリッドレイアウト デザインブック』同上
『WordPress レッスンブック 5.x 対応版』ソシム刊
『フレキシブルボックスで作る HTML5&CSS3 レッスンブック』同上
『CSS グリッドで作る HTML5&CSS3 レッスンブック』同上
『HTML&CSS コーディング・プラクティスブック』エビスコム電子書籍出版部刊
『グーテンベルク時代の WordPress ノート テーマの作り方（入門編）』同上
『グーテンベルク時代の WordPress ノート テーマの作り方
(ランディングページ&ワンカラムサイト編)』同上

ほか多数

Web サイト高速化のための 静的サイトジェネレーター活用入門【副読本】 **microCMS 対応ガイド（Gatsby v4 対応版）**

2020年6月1日 ver.1.0 発行
2021年11月14日 ver.1.2 発行