

作って学ぶ

# Next.js / React

## Webサイト構築

エビスコム 著

セットアップ PDF



本 PDF は下記書籍のセットアップ PDF です。



作って学ぶ

## Next.js/React Webサイト構築

- <https://book.mynavi.jp/ec/products/detail/id=130848>
- <https://ebisu.com/next-react-website/>
- <https://amzn.to/3RvfR8D>

HTML & CSS の各種機能、モダンコーディングについてのより詳しい情報は、下記の書籍を参考にしてください。



## 作って学ぶ HTML&CSS モダンコーディング

- <https://book.mynavi.jp/ec/products/detail/id=124054>
- <https://ebisu.com/html-css-modern-coding/>
- <https://amzn.to/2XsZHoU>



## HTML5 & CSS3 デザイン 現場の新標準ガイド

- <https://book.mynavi.jp/ec/products/detail/id=117364>
- <https://ebisu.com/html5-css3-practical-design-guide-2/>
- <https://amzn.to/378x17B>

- ・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づめましたが、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 2022 年 7 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<sup>®</sup>および TM マークは省略させていただいております。

# もくじ

## Setup 1 アカウントの作成 ..... 6

1.1 GitHub.....	7
1.2 microCMS.....	10
1.3 Vercel.....	12
1.4 Netlify.....	14
1.5 Figma.....	16

## Setup 2 開発環境の準備 ..... 18

2.1 Node.js.....	19
Node.js のバージョン管理ツール.....	19
Volta のインストール .....	19
Node.js のインストール.....	20
プロジェクトでのバージョンの固定 .....	21
不要になった Node.js を削除する .....	21
2.2 Git .....	22
Git のインストール .....	22
Git の初期設定 .....	23
2.3 エディタ (Visual Studio Code) .....	24
Visual Studio Code のインストール .....	24
Visual Studio Code の日本語化 .....	24
Prettier と ESLint .....	26
■ Remote WSL .....	28

## Setup 3 サイトの公開 ..... 29

<b>3.1 GitHub の設定</b> .....	30
リポジトリを用意する.....	30
リポジトリへプッシュする.....	31
<b>3.2 Vercel の設定とサイトの公開</b> .....	33
環境変数の追加・編集.....	35
サブドメインの変更.....	35
<b>3.3 Netlify の設定とサイトの公開</b> .....	36
環境変数の追加・編集.....	38
サブドメインの変更.....	39
<b>3.4 プロジェクトの更新をサイトに反映させる</b> .....	40
<b>3.5 microCMS によるサイトの更新</b> .....	41
Vercel での準備.....	41
Netlify での準備.....	42
microCMS の設定.....	43

## Setup 4 コンテンツの準備 ..... 45

<b>4.1 microCMS で管理するコンテンツの構成</b> .....	46
<b>4.2 サービスの作成</b> .....	47
<b>4.3 カテゴリ API の作成とカテゴリーデータのインポート</b> .....	48
<b>4.4 ブログ API の作成と記事データのインポート</b> .....	50
<b>4.5 コンテンツデータを扱うのに必要な API キーや ID を確認する</b> .....	55
API キー.....	55
サービス ID.....	56
エンドポイント名.....	56
フィールド ID.....	57
■ フィールドの詳細設定.....	58

## Setup 5 デザインデータの利用 ..... 59

<b>5.1 Figma でデザインデータを開く .....</b>	60
■ ファイルブラウザの開き方 .....	60
デザインデータの構成 .....	61
<b>5.2 デザインの設定を確認する .....</b>	62
グループ化された個々のレイヤーの選択 .....	63
表示範囲と倍率の変更 .....	63
Web ページの横幅 .....	64
レイヤーの間隔 (スペースのサイズ) .....	64
テキストと色のスタイル .....	65
コンテンツの横幅を示すレイアウトグリッドのスタイル .....	66
コンポーネントとプロパティ .....	67
オートレイアウト .....	68

## Setup

1

### アカウントの 作成

本書では GitHub、microCMS、Vercel、Netlify、Figma のサービスを利用します。いずれのサービスでも、支払いの設定をすることなしにアカウントを作成できますので、安心して使い始めることができます。ここでは、これらがどのようなサービスなのかを確認し、アカウントの作成方法を解説します。



Next.js / React

## 1.1 GitHub

GitHub はソフトウェアの開発のプラットフォームで、ソースコードをホスティングすることができます。コードのバージョン管理システムには、Git を採用しています。個人向けには Free プランが用意されており、以下のような設定となっています。

- 無制限のパブリックリポジトリ
- 無制限のプライベートリポジトリ

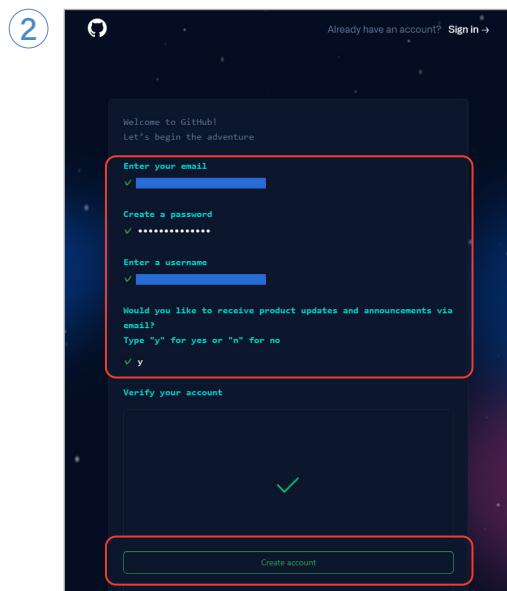
※プラン詳細: <https://github.com/pricing>

GitHub  
<https://github.com/>

### ❖ GitHubのアカウントの作成



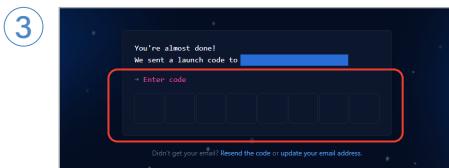
GitHub のサイトにアクセスし、「Sign up」をクリックします。



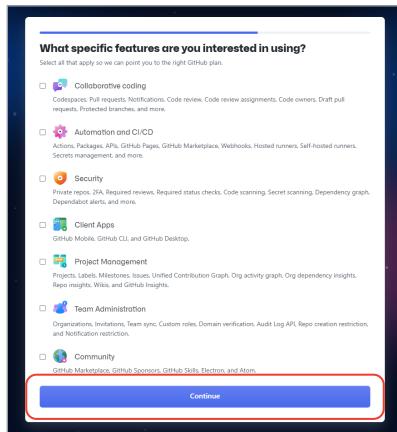
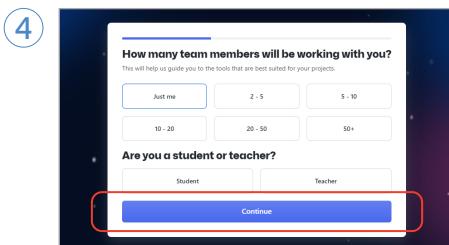
- email (メールアドレス)
- password (パスワード)
- username (ユーザー名)
- プロダクトのアップデートやアナウンスをメールで受け取るか?

を入力し、「Create account」をクリックします。

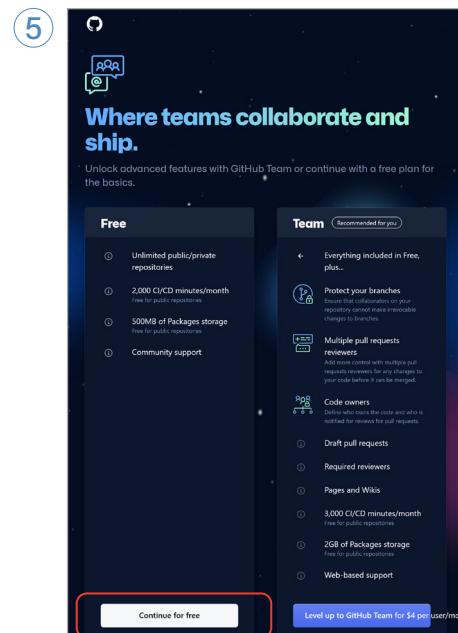
## Setup 1 アカウントの作成



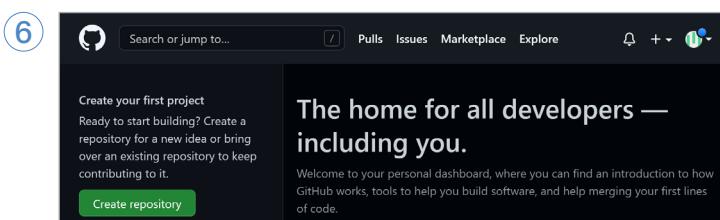
入力したメールアドレスに launch code が送られてきますので、それを入力します。



質問に答えて「Continue」をクリックします。



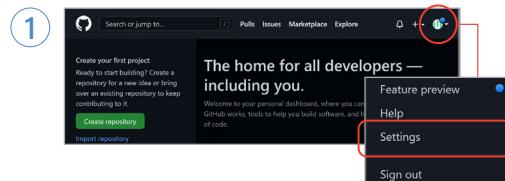
プランを選択します。ここでは「Continue for free」を選択します。



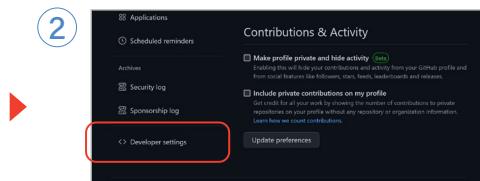
左のようなページが開き、アカウントの作成は完了です。

## Personal access tokensの作成

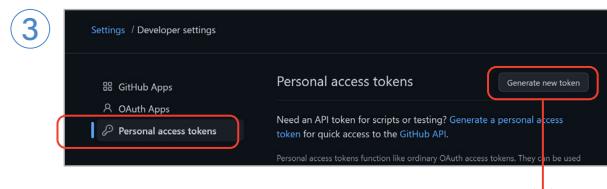
続いて、Personal access tokens を作成します。これは、GitHub にプッシュする際にパスワードの代わりに使用するものです。



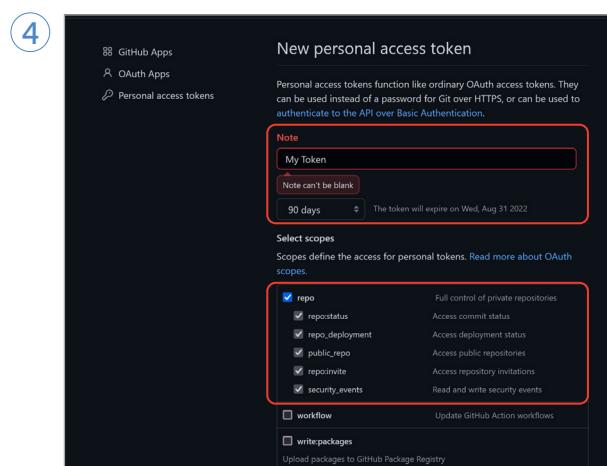
右上のプロフィール画像をクリックし、メニューから「Settings」をクリックします。



左側のメニューの一番下にある「Developer settings」をクリックします。

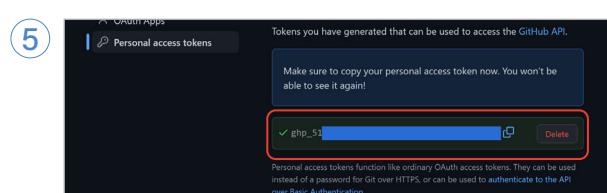
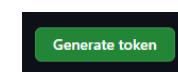


左側のメニューにある「Personal access tokens」を選択し、「Generate new token」をクリックします。



Note とトークンの有効期間、権限のスコープを指定します。ここでは、トークンの期限を 90 日、リポジトリのアクセス権限を選択しています。

選択したらページ下部にある「Generate token」ボタンをクリックします。



トークンが表示されますので、コピーしてしっかりと保存しておきます。このページを閉じてしまうと、もう確認できませんので注意してください。

## 1.2

Account

# 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。また、画像処理には imgix の API が利用可能です

- API リクエスト数：無制限
- Webhook 数：10 個
- API 数：3 個
- コンテンツ数：10000
- データ転送量：20GB

※プラン詳細：<https://microcms.io/pricing/>

microCMS  
<https://microcms.io/>

## ❖ microCMSのアカウントの作成

①



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

The screenshot shows the 'Free Account Registration' page. It has fields for 'メールアドレス' (Email Address) and 'パスワード' (Password), both of which are highlighted with a red border. Below these fields is a checkbox for '利用規約、プライバシーポリシーに同意します' (I agree to the Terms of Service and Privacy Policy), also highlighted with a red border. At the bottom is a large blue button labeled 'アカウントを登録する' (Register Account).

左のような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「アカウントを登録する」をクリックします。

## Setup 1 アカウントの作成

③

ご登録ありがとうございます  
より適切なサポートのため、アンケートにご協力をお願いします。

あなたの職種は何ですか？  
選択してください

開発しているサービスの種類はなんですか？  
選択してください

会社の規模を教えてください。  
選択してください

microCMS登録で知りましたか？  
選択してください

**続ける** スキップ

アンケートに答えて「続ける」をクリックするか、「スキップ」を選びます。



左のような画面が表示されたら、アカウントの作成は完了です。

入力したメールアドレスに確認メールが届きますので、メール内のリンクをクリックして確認しておきます。



なお、メッセージを閉じると、コンテンツの準備を始める画面になります。

コンテンツの準備を始める場合はセットアップ 4 (P.46 ~) を参照してください。

## 1.3 Vercel

Account

Next.js を開発している Vercel 社の Web ホスティングサービスです。フロントエンドフレームワークと静的サイトのためのプラットフォームであり、Next.js に最適化されています。

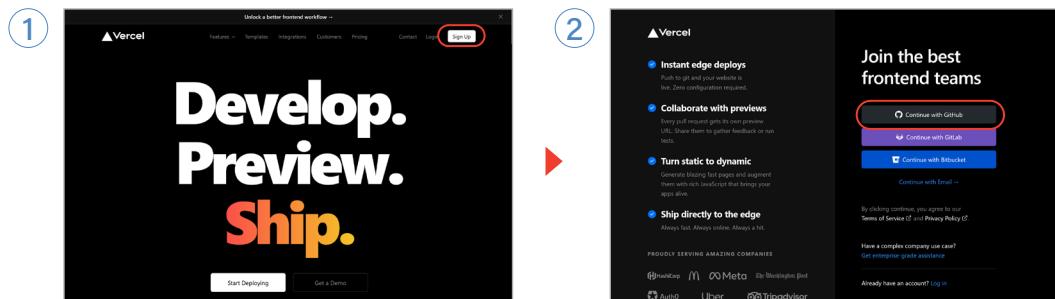
Hobby として無料のプランが用意されており、以下のような設定となっています。ただし、non-commercial sites のみの制限や、next/image による画像の最適化にも制限がありますので、注意が必要です。

- 転送量 : 100GB / 月
- ビルド時間 : 100 時間 / 月
- 画像の最適化 : 1000 画像

※プラン詳細: <https://vercel.com/docs/concepts/limits/overview>

Vercel  
<https://vercel.com/>

### ❖ Vercelのアカウントの作成

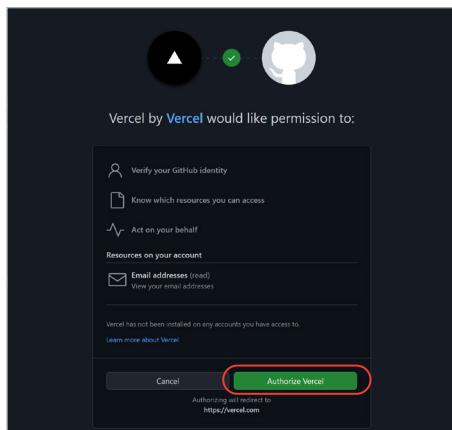


Vercel のサイトにアクセスし、「Sign Up」をクリックします。

セットアップ 1.1 で作成した GitHub のアカウント情報を使って Vercel のアカウントを作成しますので、「GitHub」をクリックします。

## Setup 1 アカウントの作成

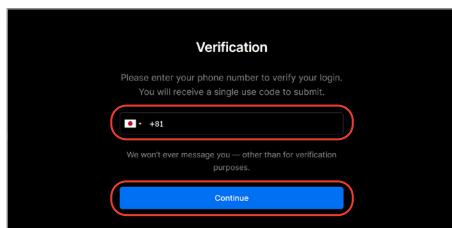
③



GitHub にリダイレクトされます。Vercel から GitHub を利用することを許可するため、「Authorize Vercel」をクリックします。

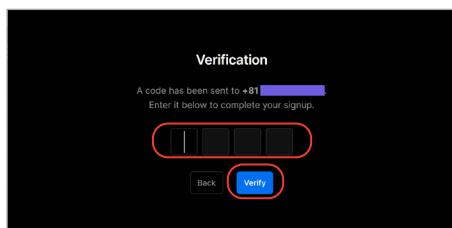
※ GitHub にログインしていない場合は、  
ログインして確認します。

④



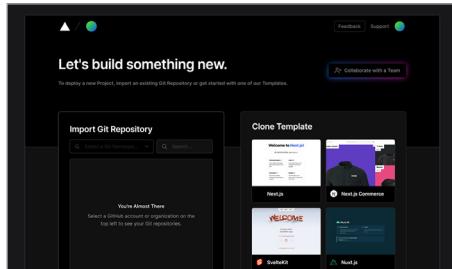
Vercel の「Verification」のページが開きます。  
認証のため、SMS の送り先の電話番号を求められ  
ますので、入力して「Continue」をクリックし  
ます

⑤



送られてきたコードを入力して、「Verify」を  
クリックします。

⑥



左のようなページが表示されます。  
これで、アカウントの作成は完了です。

## 1.4 Account

# Netlify

Netlify は最新の Web プロジェクトを自動化するためのオールインワンプラットフォームです。Web サイトのホスティングサービスを始め、様々なサービスが用意されており、Next.js にも対応しています。

Starter として無料のプランが用意されており、以下のような設定となっています。ちょっとしたサイトであれば、無料枠を使い切ることはないでしょう。

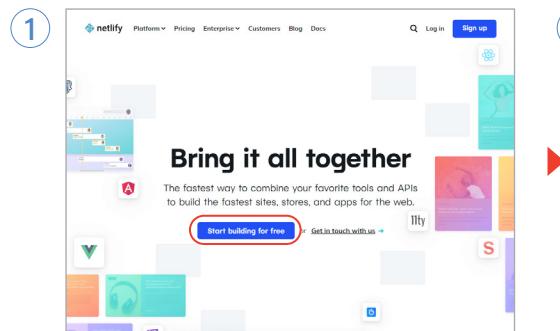
- 転送量 : 100GB/ 月
- ビルド時間 : 300 分 / 月
- Web サイト数 : 無制限

※プラン詳細: <https://www.netlify.com/pricing/>

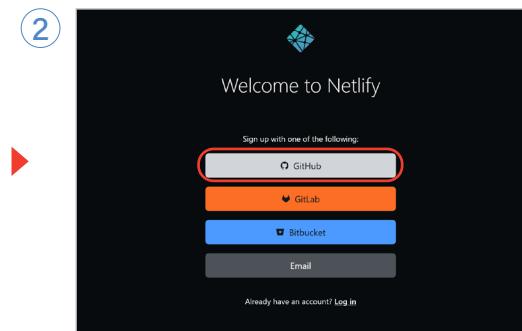
※Next.jsのOn-demand Revalidation (ISR) にはこの原稿を書いている時点では対応していません。

Netlify  
<https://www.netlify.com/>

## ❖ Netlifyのアカウントの作成



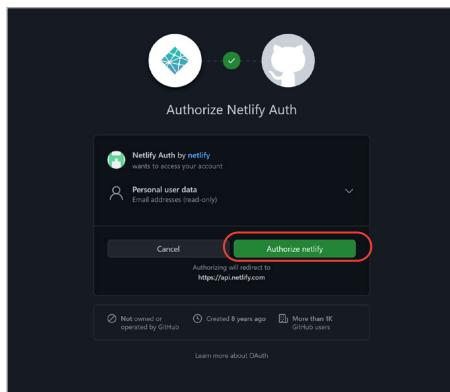
Netlify のサイトにアクセスし、「Sign up」または「Start building for free」をクリックします。



セットアップ 1.1 で作成した GitHub のアカウント情報をを使って Netlify のアカウントを作成しますので、「GitHub」をクリックします。

## Setup 1 アカウントの作成

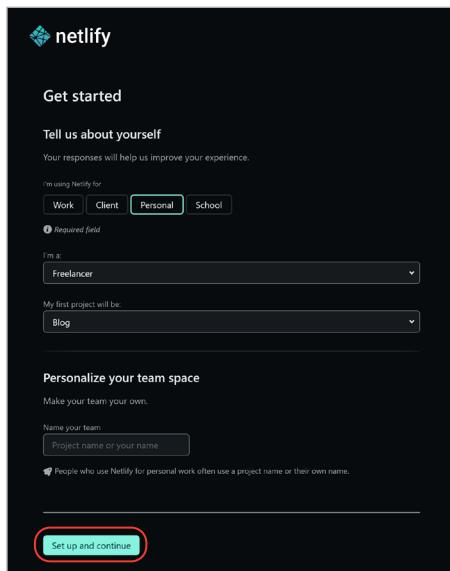
③



GitHub にリダイレクトされます。GitHub で設定したメールアドレスの情報を使ってもよいかと確認されますので、「Authorize netlify」をクリックします。

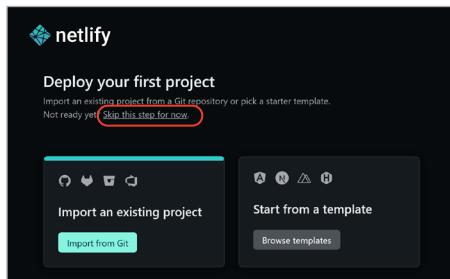
※ GitHub にログインしていない場合は、  
ログインして確認します。

④



Netlify の「Get Started」のページが開きますので、利用目的などを選択して「Set up and continue」をクリックします。

⑤



左のようなページが表示されたら準備完了です。  
ここでは「Skip this step for now」をクリックし、  
アカウントの作成を完了します。

## 1.5 Account

# Figma

Figma は Web サイトやモバイルアプリのデザイン、プロトタイプ、デザインシステムなどを作成し、共有・コラボレーションできるデザインツールです。Web をベースとしており、ブラウザとデスクトップアプリの両方で利用できます。

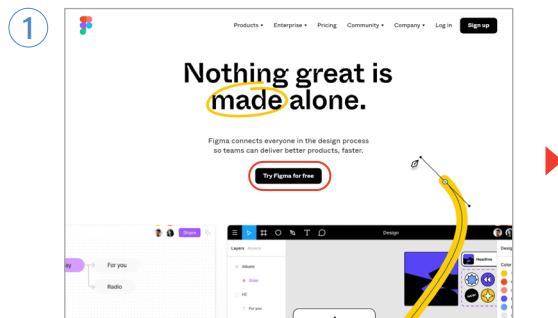
Starter として無料のプランが用意されており、以下のような設定となっています。

- 個人ファイル（Drafts ファイル）：無制限
- チーム内の Figma ／ FigJam ファイル：それぞれ 3 つまで
- バージョン履歴：30 日

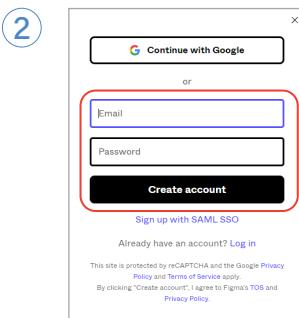
※プラン詳細：<https://www.figma.com/pricing/>

Figma  
<https://www.figma.com/>

## ❖ Figmaのアカウントの作成



Figma のサイトにアクセスし、「Sign up」または「Try Figma for free」をクリックします。



メールアドレスとパスワードを入力して、「Create account」をクリックします。

※Googleアカウントの情報を使って作成する場合は「Continue with Google」をクリックします。

## Setup 1 アカウントの作成

③

Tell us about yourself

Your name

What kind of work do you do? \*

I agree to join Figma's mailing list

Create account

Verify your email

To use Figma, click the verification button in the email we sent to [REDACTED]. This helps keep your account secure.

No email in your inbox or spam folder? Let's resend it.

Wrong address? Log out to sign in with a different email. If you mistyped your email when signing up, create a new account.

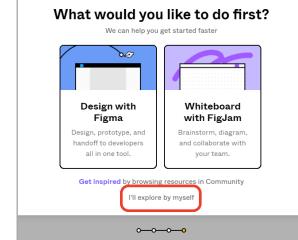
「Your name」に名前を入力し、質問に答えて「Create Account」をクリックします。

すると、メールを確認するように求められますので、メール内に記載されたリンクをクリックします。

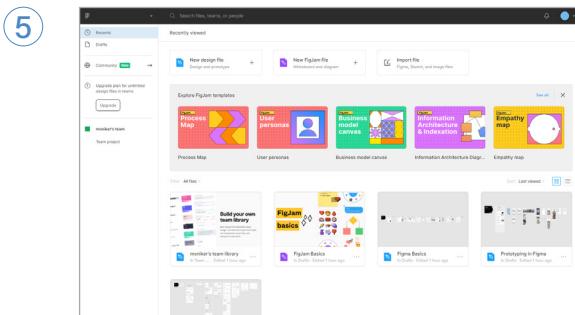


チームの設定が求められますが、ここでは「Do this later (後で設定)」を選択します。

プランの選択肢が表示されますので、「Starter」を選択します。



「I'll explore by myself」を選択します。



Figma のファイルブラウザが開きます。これで、アカウントの作成は完了です。  
Figma のデスクトップアプリ (Windows または macOS) を使用する場合は右のページからダウンロードしてインストールします。



デスクトップアプリのダウンロード

Figma downloads

<https://www.figma.com/downloads/>

## Setup

# 2

## 開発環境の 準備

Linux、macOS、Windows で Node.js、Git、エディタ（Visual Studio Code）を用意し、開発環境を準備する手順を解説します。

ただし、Windows 上で Next.js を使う場合、Microsoft Docs では WSL2 での環境構築の方法が紹介されています。そのため、この PDF でも WSL2 を使用することを前提として解説しています。

参照：Windows 上の Next.js の概要

<https://docs.microsoft.com/ja-jp/windows/dev-environment/javascript/nextjs-on-wsl>



# Next.js / React

## 2.1

# Node.js

Dev Environment

Node.js はブラウザ上で動く JavaScript とは異なり、サーバー上で動く JavaScript 環境です。Next.js を使うためには、この Node.js が必要です。

Node.js

<https://nodejs.org/ja/>

## ❖ Node.js のバージョン管理ツール

Node.js は直接インストールしても問題ありませんが、プロジェクトに合わせて Node.js のバージョンを管理しなければならないケースも少なくありません。そのため、Node.js のインストールには、Node.js のバージョン管理ツールを使うことをおすすめします。

Volta

<https://volta.sh/>

バージョン管理ツールの選択肢は色々とありますが、ここでは、非常にシンプルに扱える Volta を使います。

## ❖ Volta のインストール

Volta は、Linux / macOS / WSL (Windows Subsystem for Linux) の環境でターミナルから以下のコマンドでインストールします（現時点では、インストール済の Volta のアップグレードもこのコマンドです）。

```
$ curl https://get.volta.sh | bash
```

インストールが完了したら、次のコマンドでバージョンが表示されるのを確認しておきます（ターミナルの再起動が必要な場合もあります）。

```
$ volta --version
```

## ❖ Node.jsのインストール

Voltaを使ったNode.jsのインストールは簡単です。以下のコマンドで、LTS（Long Term Support）版の最新のものがインストールされます。

```
$ volta install node
```

バージョンを指定する場合は、次のようにします。

```
$ volta install node@14 // ← 14の最新がインストールされる
$ volta install node@17.2.0 // ←バージョンを指定してインストールも可能
```

インストール済のものを確認する場合は、`list all`を使います。`(default)`がついているものがデフォルトで使用されます。

```
$ volta list all
⚡ User toolchain:

  Node runtimes:
    v12.22.12
    v14.19.3 (default)
    v16.13.1
    v16.14.0
```

`install`コマンドの本来の機能はデフォルトの指定です。インストールされていないものを指定した場合には、ダウンロードの上、デフォルトの指定をしています。

## ❖ プロジェクトでのバージョンの固定

JavaScript のプロジェクトで Node.js のバージョンを固定することができます。たとえば、Next.js のプロジェクトディレクトリで

```
$ volta pin node@16
```

と指定すると、`package.json` には次のような項目が追加されます。

```
"volta": {  
  "node": "16.15.0"  
}
```

`package.json`

今後は、このディレクトリに移動すると自動的に指定したバージョンに切り替わります。`list` で次のように確認できます。

```
$ volta list  
⚡ Currently active tools:  
  
Node: v16.15.0 (current @ /xxx/blog/package.json)
```

## ❖ 不要になったNode.jsを削除する

インストールした Node.js は、`~/.volta/tools/image/` 以下のディレクトリにインストールされていますので、不要なものを削除してください。Volta の `uninstall` は、現時点では Node.js のアンインストールには対応していません。

# 2.2

Dev Environment

## Git

Git はフリーでオープンソースの分散型バージョン管理システムです。GitHub などを利用する上で必須のソフトウェアとなっています。

Git

<https://git-scm.com/>

### ❖ Gitのインストール

#### Linux & WSLでの確認

ここでは、Ubuntu 22.04 LTS を想定して進めていきます。ターミナルを起動し、次のコマンドで Git がインストールされていることを確認します。

```
$ git --version  
git version 2.34.1
```

#### macOSでの確認

ターミナルを起動し、Git がインストールされているかを確認します。

```
$ git --version
```

Git がインストールされていない場合には、「"git" コマンドを実行するには…」というダイアログが表示されるので、クリックしてインストールしてください。コマンドライン・デベロッパ・ツールがインストールされ、その中に Git が含まれています。



アンインストールする場合には、以下のコマンドで削除します。

```
$ sudo rm -rf /Library/Developer/CommandLineTools
```

Git を単体でインストールしたい場合は、Homebrew や公式のインストーラーを使ってインストールすることもできます。

Download for macOS

<https://git-scm.com/download/mac>



Git は ver 2.28 から、初期ブランチ名を設定できるようになりました。これは、これまで使われてきた初期ブランチ名の master が不適切であるとなったためです。GitHub などでも、初期ブランチ名が main と変更されるようになりましたので、ver 2.28 以降を使うことをオススメします。

## ❖ Gitの初期設定

続いて、ユーザー名と Email アドレスを設定します。これは、Git が個人を識別するために利用されます。メールアドレスには GitHub のアカウントを作成する際に登録したものを使用します。

```
$ git config --global user.name "Moniker"  
$ git config --global user.email "moniker@example.com"
```

Git のバージョンが ver2.28 以降の場合は、初期ブランチ名を設定することができます。GitHub に合わせて main に変更しておきます。

```
$ git config --global init.defaultBranch main
```

## 2.3

Dev Environment

# エディタ (Visual Studio Code)

JavaScript の開発環境では、エディタの存在が非常に重要です。ここでは、Microsoft の Visual Studio Code を使って、基本的な環境構築をしていきます。

## ❖ Visual Studio Codeのインストール

Visual Studio Code は、Windows、macOS、Linux をサポートした Microsoft 製のコードエディタです。オープンソースで開発されており、無料で使用することができます。

右記の Visual Studio Code のサイトからダウンロードし、インストーラーの指示に従ってインストールします。

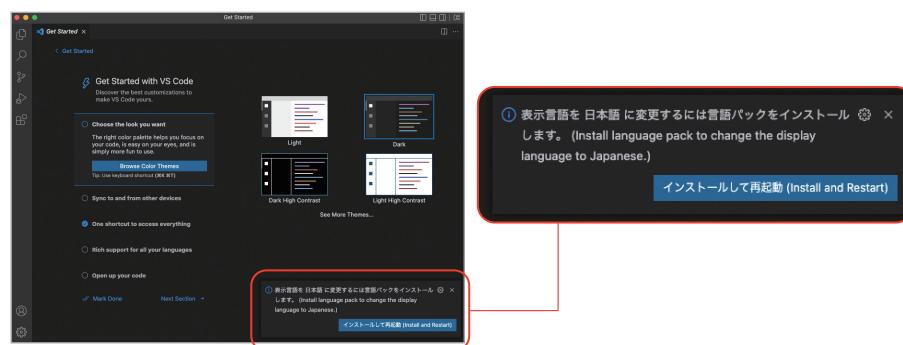
Visual Studio Code

<https://code.visualstudio.com/>

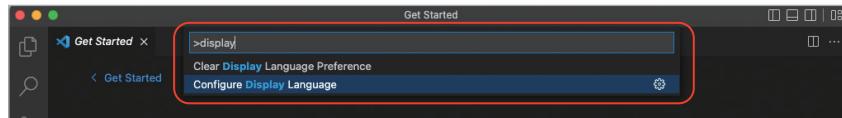
## ❖ Visual Studio Codeの日本語化

インストールが完了したら、Visual Studio Code を起動します。すると、英語表示で起動しますので、まずは、日本語の言語パック (Japanese Language Pack for Visual Studio Code) をインストールし、日本語化を行います。

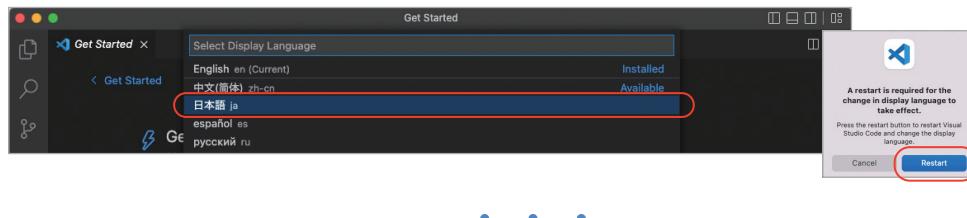
起動後、右下に日本語化に関する通知が表示された場合は「インストールして再起動」をクリックします。



メニューからインストールする場合、View > Command Palette… を選択し、コマンドパレットを開いて display と入力します。Configure Display Language が選択肢として表示されますので、選択します。

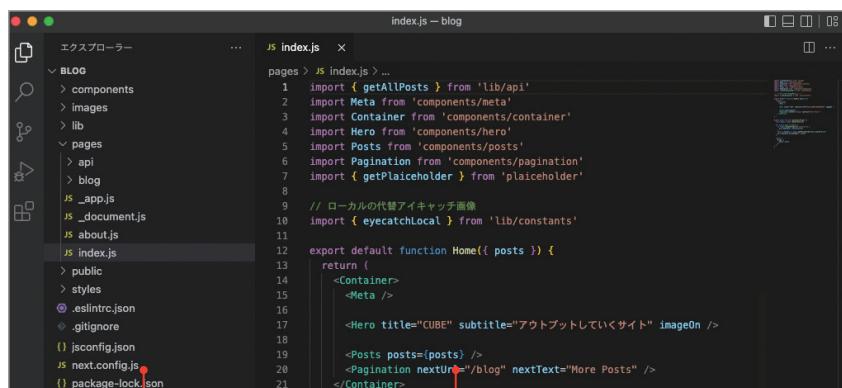


言語の選択肢が表示されますので、日本語 ja を選択します。すると、日本語の言語パックがインストールされます。インストールが完了すると再起動するように求められますので、「Restart」をクリックしてを再起動します。



Visual Studio Code が再起動すると、日本語化が完了しています。ファイル > フォルダーを開く… から Next.js のプロジェクトのディレクトリを開くと、プロジェクトを構成するファイルが左パネルに表示されます。

ファイルをシングルクリックすると表示し、ダブルクリックで編集可能になります。JavaScript のファイルを開くと、この段階でシンタックスハイライトが有効になっているのが確認できます。



プロジェクトを構成するファイル。

開いたファイル。

## ❖ Prettier と ESLint

続いて、Prettier と ESLint をインストールします。Prettier はフォーマッターで、コードを設定に合わせて整形してくれます。コードのスタイルに一貫性をもたせるのに便利です。

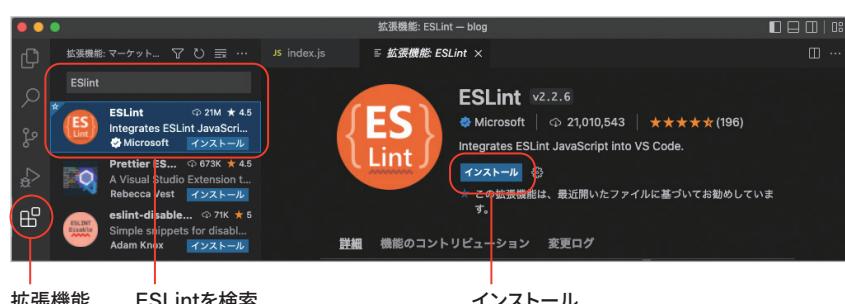
ESLint は JavaScript のリンターで、コードを解析し問題のある部分を指摘してくれます。フォーマッターの機能も持っています。

フォーマッターの機能がダブっているので ESLintだけでも良さそうに思えますが、フォーマッターとしては Prettierの方が優秀なのです。そのため、Prettier と ESLint を共存させ、ESLintにはコード解析だけをまかせるというのが、執筆時点では標準的な使い方になっています。

- ① 時期によってその役割分担や設定方法が変化しますので、確認をオススメします。

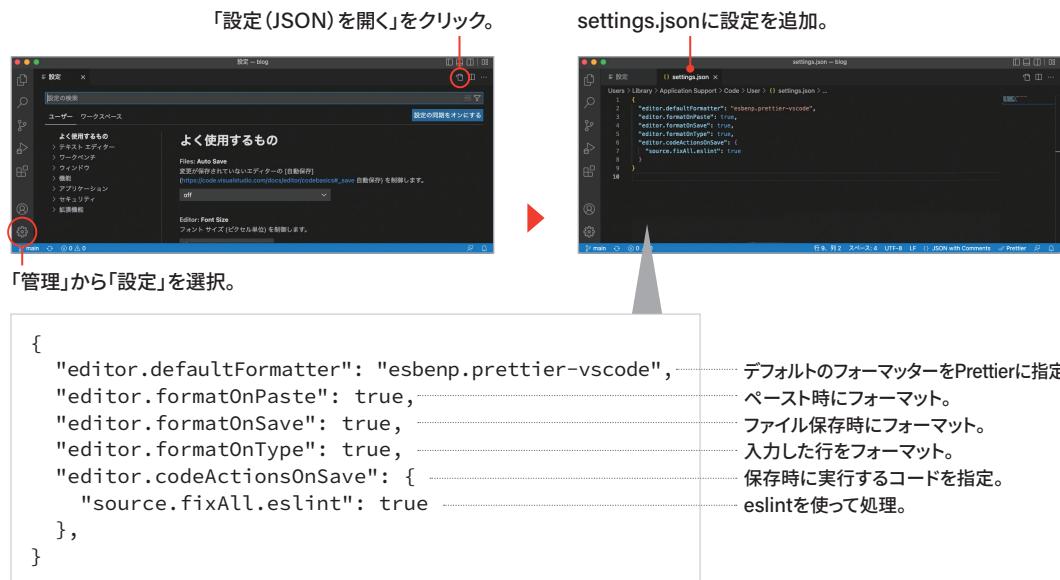
### Prettier と ESLint のインストール

「拡張機能」をクリックし、Prettier と ESLint を検索してインストールします。



## Prettier と ESLint の設定

「管理」から **設定** を選択し、設定画面を開きます。右上の **設定 (JSON)** を開くをクリックすると、**settings.json** が開きますので、ここに以下の設定を追加して保存します。



これで、Visual Studio Code 側の設定は完了です。

続いて、Prettier と ESLint の機能の衝突を回避するために、**eslint-config-prettier** を Next.js (JavaScript) のプロジェクトにインストールします。

**eslint-config-prettier**  
<https://github.com/prettier/eslint-config-prettier>

```
$ npm install --save-dev eslint-config-prettier
```

そして、**.eslintrc.json** (**.eslintrc.\***) に **"prettier"** が最後になるように追加します。Next.js のプロジェクトの場合、次のようにになります。これで、Prettier と ESLint の機能の衝突がなくなります。

```
{
  "extends": ["next/core-web-vitals", "prettier"]}
```

**.eslintrc.json**

Next.js のプロジェクトに標準で用意されている Next.js 用の ESLint の設定も機能するようになります。

コードフォーマットについては、プロジェクトのディレクトリに `.prettierrc` というファイルを用意することで設定ができます。本書では、以下の設定を使っています。

```
{
  "trailingComma": "all", ..... 末尾にカンマを付加。
  "tabWidth": 2, ..... インデントをスペース2つ分に指定。
  "semi": false, ..... 末尾のセミコロンを省略。
  "singleQuote": true, ..... シングルクオートを使用。
  "jsxSingleQuote": false, ..... JSXではダブルクオートを使用。
  "printWidth": 80 ..... 折り返す行の長さを80文字に指定。
}
```

`.prettierrc`



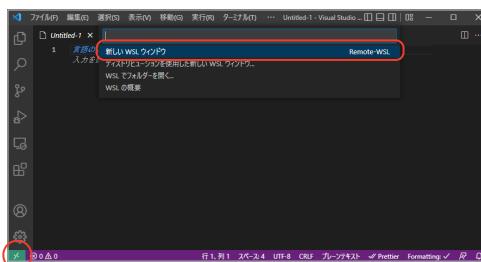
`.prettierrc` で指定できるオプションについては下記を参照してください。

#### Options

<https://prettier.io/docs/en/options.html>

## Remote WSL

Windows で WSL を使う場合、機能拡張から Remote WSL をインストールすることで、WSL 内に直接アクセスできるようになります。左下の緑の部分をクリックすると図のようなウインドウが開きますので、「新しい WSL ウィンドウ」を選択することで WSL にアクセスできます。



WSL の環境は、Windows 環境とは別の環境です。そのため、WSL 側にも設定や機能拡張のインストールが必要になる場合があります。Prettier と ESLint は WSL 側にインストールする必要があります。

## Setup

# 3

## サイトの公開

GitHub と Vercel / Netlify を組み合わせ、サイトの公開・更新が手軽にできる環境を設定していきます。さらに、microCMS で記事を追加・修正・削除した場合にも、サイトが更新されるように設定します。

なお、設定を始める前に、ローカル環境でビルト処理(next build) が問題なく完了することを確認してください。問題がなければ設定を始めましょう。



# Next.js / React

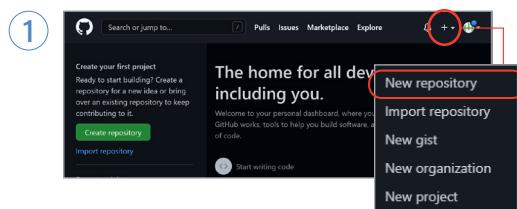
# 3.1

## Web Deployment

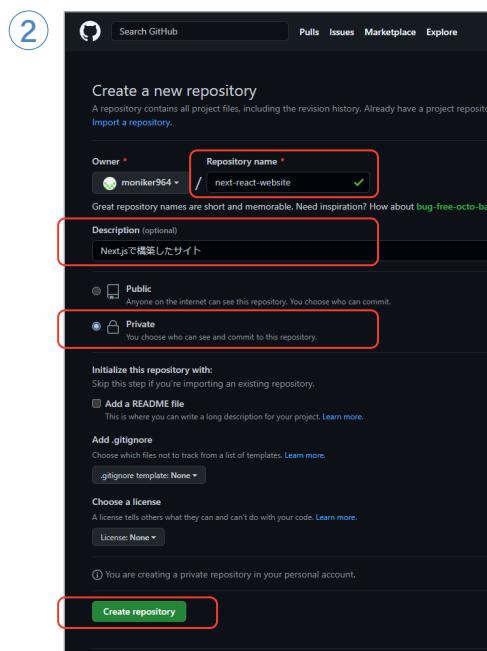
# GitHubの設定

まずは GitHub の設定を行い、Next.js のプロジェクトを push (プッシュ) します。

## ❖ リポジトリを用意する



Next.js のプロジェクトを管理するリポジトリを用意します。GitHub に Sign in (サインイン) し、右上の「+」をクリックして「New repository」を選択します。

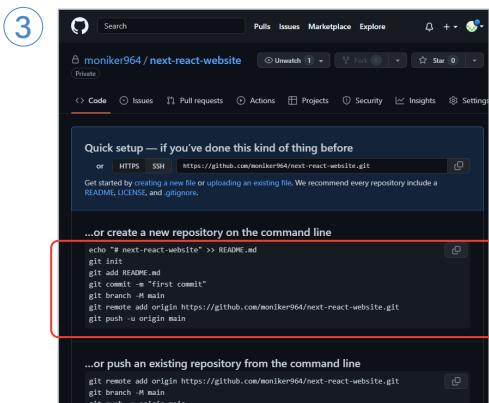


「Create a new repository」という画面が開きますので、Repository name (リポジトリ名) を指定します。ここでは、「next-react-website」と指定しています。

Description (説明) は必要に応じて入力します。

「Public」と「Private」では、このリポジトリを公開にするか、非公開にするかを指定します。ここでは「Private」を選択し、非公開にしています。

設定が完了したら、「Create repository」をクリックします。



左のような画面が開きます。

この部分を参考に、Next.js のプロジェクトをこのリポジトリへ push (プッシュ) します。なお、この部分をコピーしてメモしておくか、ブラウザで開いたままにしておきます。

## ❖ リポジトリへプッシュする

Next.js のプロジェクトのディレクトリ（本書のサンプルの場合は `blog` ディレクトリ）のルートに移動します。ルートにある `.git` ディレクトリを流用することもできますが、ここではリポジトリの新規作成から行うため、`.git` ディレクトリを削除します。

```
$ rm -rf .git
```

続けて、次のコマンドを実行していきます。

- ① ローカルのリポジトリを新規作成します。`.git` ディレクトリが作られます。

```
$ git init
```

- ② すべてのファイル (`blog` ディレクトリの中身) をインデックスに登録します。ただし、`.gitignore` に設定されているファイルやフォルダは登録しません

```
$ git add -A
```

- ③ メッセージを指定して、コミットします。

```
$ git commit -m "first commit"
```

- ④ ブランチを `main` にします（初期ブランチが `main` の場合は必要ありません）。

```
$ git branch -M main
```

- ⑤ リモートリポジトリを指定します。

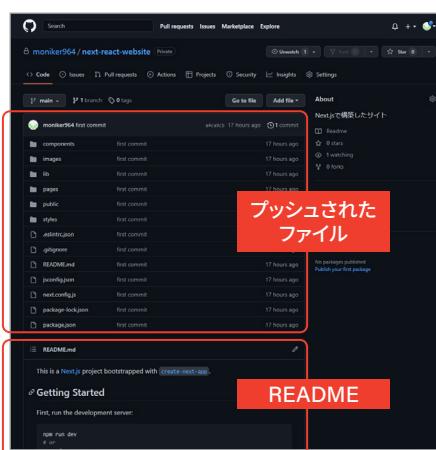
ここでは、GitHub に作成したリポジトリを指定しています。

```
$ git remote add origin https://github.com/moniker964/next-react-website.git
```

この部分がリポジトリごとに変わります。

- ⑥ リモートリポジトリへプッシュします。このとき、`Username` と `Password` の入力を求めてきますので入力します。ただし、`Password` にはセットアップ 1.1 (P.9) で作成した personal access token を入力します。

```
$ git push -u origin main
```



プッシュが完了したら、開いていた GitHub のページをリロードしてください。プッシュされたファイルが確認できます。

また、README.md の内容が表示されます（編集していない場合には、Next.js の標準の README がプッシュされています）。

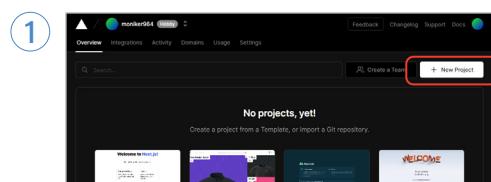
以上で、GitHub の準備は完了です。

## 3.2

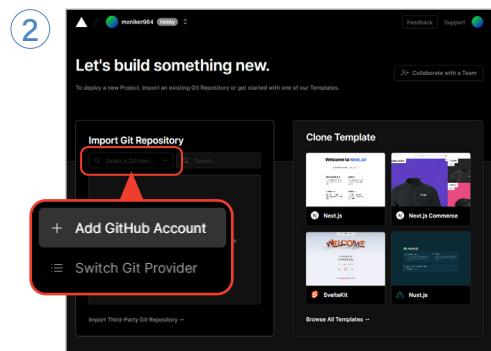
### Web Deployment

# Vercelの設定とサイトの公開

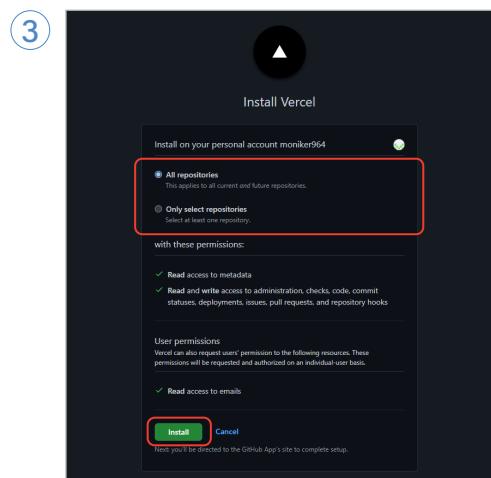
Vercel を使ってサイトを公開します。そのためには、GitHub にプッシュしたプロジェクトを Vercel にデプロイします。



Vercel にログイン (Login) します。左のようなページが開きますので、「New Project」をクリックします。



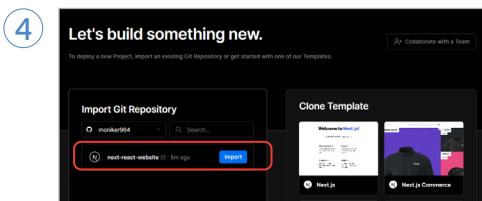
「Let's build something new.」というページが開きますので、左側のプルダウンから、「Add GitHub Account」を選択します。



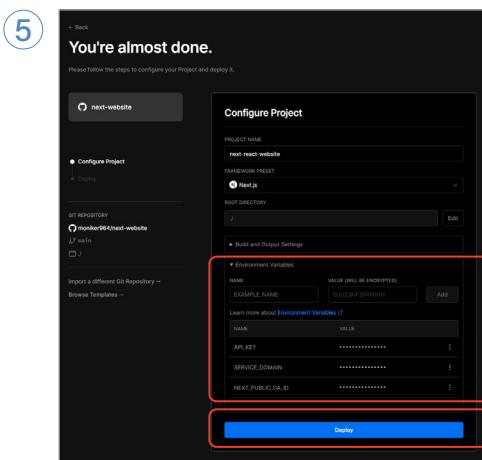
GitHub の画面が開きますので、Vercel からのリポジトリへのアクセスを許可します。

全てのリポジトリへのアクセスを許可する場合は「All repositories」を、選択したリポジトリへのアクセスのみを許可する場合は「Only select repositories」を選択します。

ここでは「All repositories」を選択し、「Install」をクリックします。



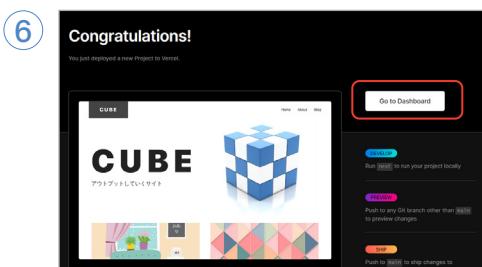
GitHub の画面が閉じ、Vercel のページに戻ります。GitHub 上のリポジトリが表示されますので、デプロイするリポジトリの「Import」をクリックします。



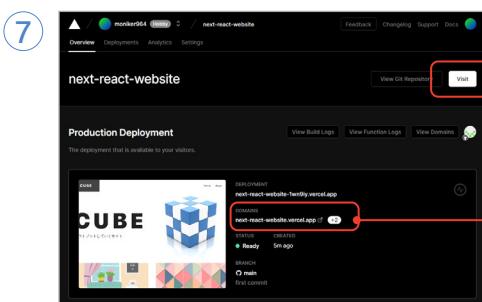
プロジェクトの設定を行う「Configure Project」のメニューが表示されます。

「Environment Variables」を開き、`.env.local` を通して使っていた環境変数を設定します。環境変数はあとから追加・編集することもできます。

設定が完了したら「Deploy」をクリックします。これで、サイトのデプロイが始まります。



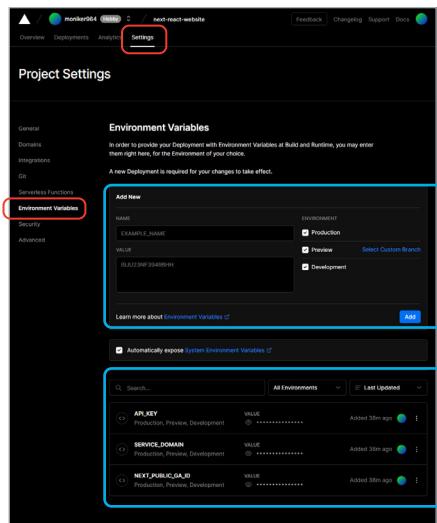
問題なく処理が完了すると、「Congratulations!」と表示されますので、「Go to Dashboard」をクリックします。



プロジェクトの管理画面が開きます。右上に表示されている「Visit」をクリックすると、公開されたサイトを確認できます。

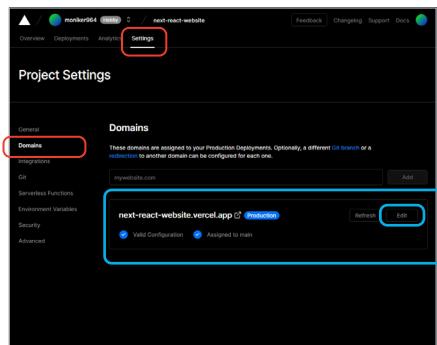
下には URL が表示されており、この URL でサイトにアクセスできます。

## ❖ 環境変数の追加・編集



環境変数を追加・編集する場合は、プロジェクトの管理画面上部で「Settings」をクリックし、開いたページの左側のメニューから「Environment Variables」をクリックします。ここで、先程と同じように環境変数を追加・編集することができます。

## ❖ サブドメインの変更

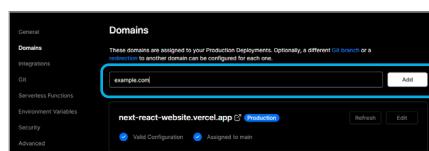


サイトの URL `~.vercel.app` はサブドメインの部分を変更することができます。プロジェクトの管理画面上部で「Settings」をクリックし、開いたページの左側のメニューから「Domains」をクリックします。

現在設定されているドメインが表示されますので、その右に表示された「Edit」をクリックし、サブドメインの部分を書き換えます。



このページでは自分が所有しているドメインを追加し、設定することもできます。



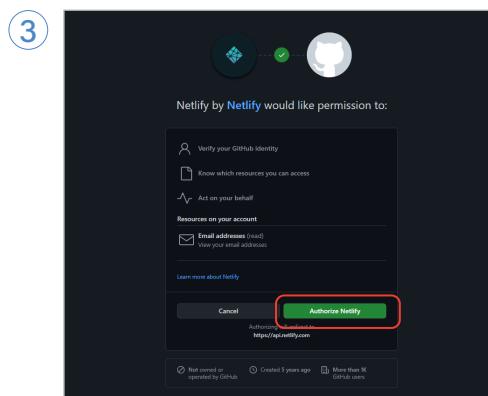
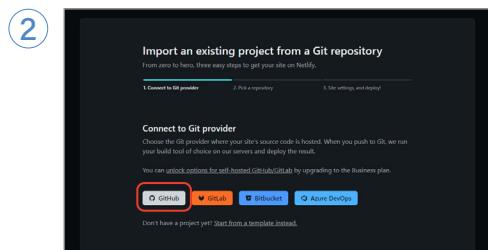
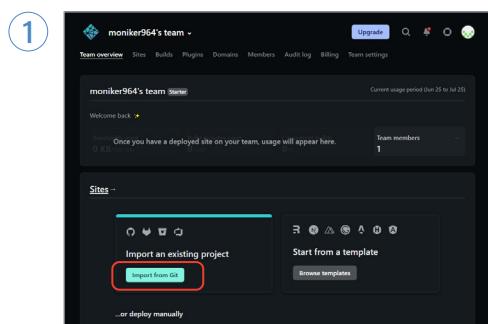
所有しているドメインを指定し、「Add」をクリック。

## 3.3

### Web Deployment

# Netlifyの設定とサイトの公開

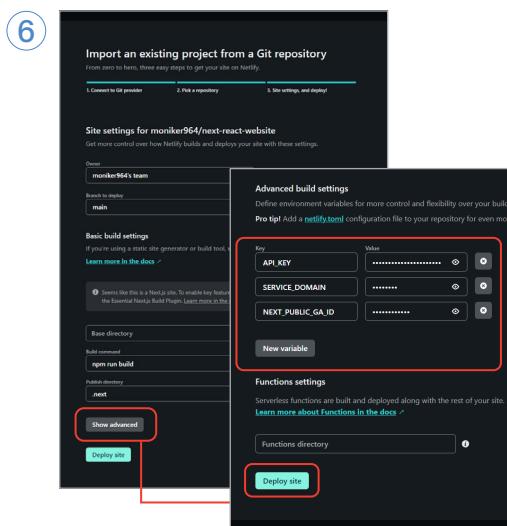
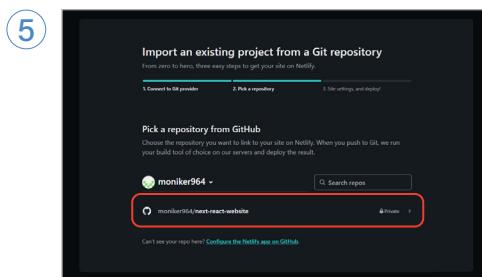
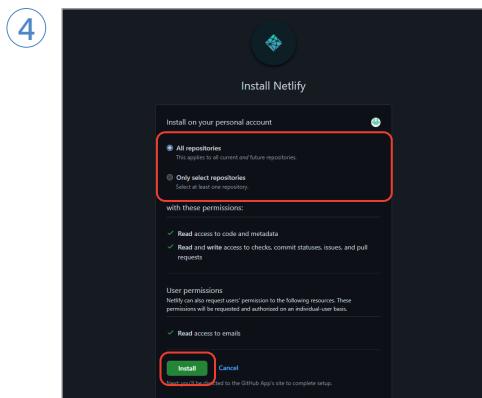
Netlify を使ってサイトを公開します。そのためには、GitHub にプッシュしたプロジェクトを Netlify にデプロイします。



Netlify にログイン (Log in) します。左のようなページが開きますので、「Import from Git」をクリックします。

左のようなページが開きます。公開するサイトのプロジェクトをどこから持ってくるかの設定です。ここでは、「GitHub」をクリックします。

GitHub の画面が開きます。Netlify から GitHub を利用することを許可するため、「Authorize Netlify」をクリックします。



続けて、Netlify からの GitHub のリポジトリへのアクセスを許可します。

全てのリポジトリへのアクセスを許可する場合は「All repositories」を、選択したリポジトリへのアクセスのみを許可する場合は「Only select repositories」を選択します。

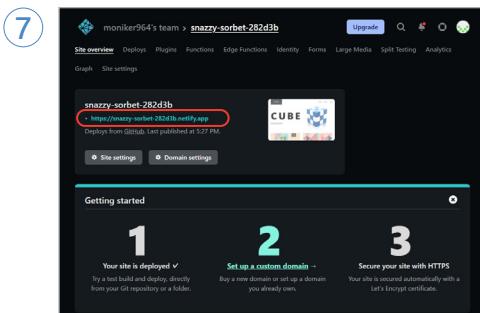
ここでは「All repositories」を選択し、「Install」をクリックします。

GitHub の画面が閉じ、Netlify のページに戻ります。GitHub 上のリポジトリが表示されますので、デプロイするリポジトリをクリックします。

プロジェクトの設定を行うページが表示されます。ここでは環境変数を設定するため、「Show Advanced」をクリックします。

追加の設定項目が表示されますので、「New variable」をクリックし、`.env.local` を通して使っていた環境変数を追加していきます。環境変数はあとから追加・編集することも可能です。

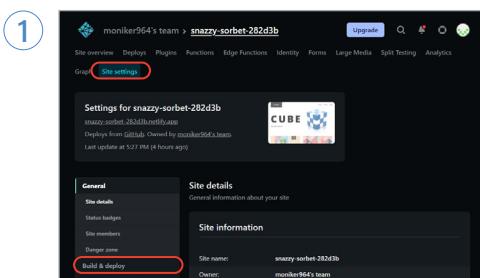
設定が完了したら「Deploy site」をクリックします。これで、サイトのデプロイが始まります。



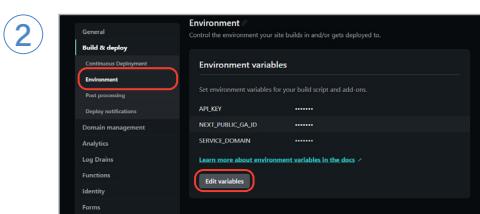
問題なく処理が完了すると、サイトが公開されます。プロジェクトの管理画面に URL が表示されていますので、この URL でサイトにアクセスすることができます。

**!** いつまでも「Site deploy in progress (デプロイ中)」のまま URL が表示されない場合はページをリロードしてみてください。

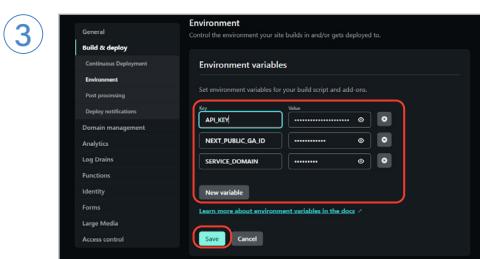
## ❖ 環境変数の追加・編集



環境変数を追加・編集する場合は、画面上部で「Site settings」をクリックし、開いたページの左側のメニューから「Build & deploy」をクリックします。



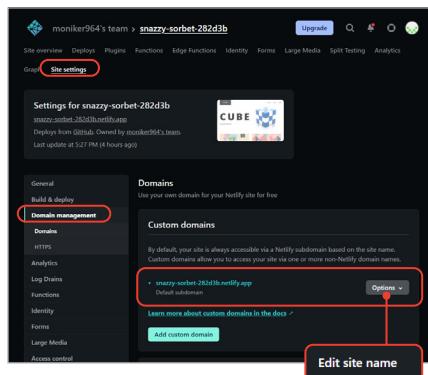
「Build & deploy」内の「Environment」を開くと、既存の環境変数の設定が表示されますので「Edit variables」をクリックします。



先程と同じように環境変数を追加・編集します。最後に、「Save」をクリックして忘れずに保存します。

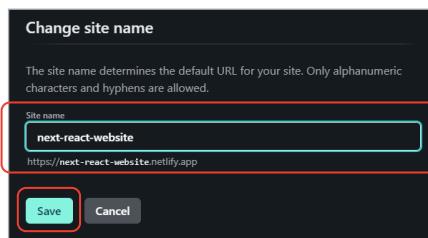
## ❖ サブドメインの変更

①



サイトの URL `~.netlify.app` はサブドメインの部分を変更することができます。画面上部で「Site settings」をクリックし、開いたページの左側のメニューから「Domain management」をクリックします。

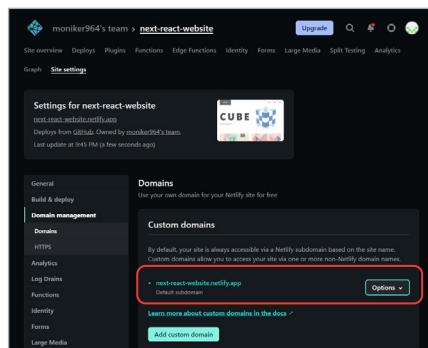
②



Default subdomain（デフォルトのサブドメイン）が表示されますので、その右に表示された「Options」から「Edit site name」を選択します。

「Change site name」という設定が開きますので、サブドメインを指定します。「Save」をクリックして保存します。

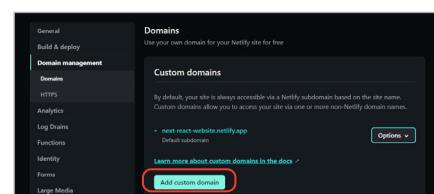
③



Default subdomain が変わったことを確認したら、設定は完了です。



「Add custom domain」をクリックすると、自分が所有しているドメインを追加し、設定することもできます。



## 3.4

### Web Deployment

# プロジェクトの更新をサイトに反映させる

プロジェクトを更新したものをサイトに反映してみましょう。まずは、ローカル環境でビルト処理（next build）が通ることを確認します。問題がなければ、以下のコマンドを実行していきます。

- すべての変更をインデックスに登録します。

```
$ git add -A
```

- メッセージを指定して、コミットします。

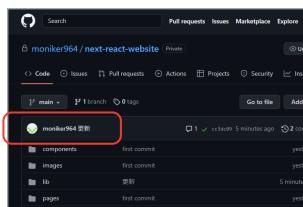
```
$ git commit -m "更新"
```

- リモートリポジトリへプッシュします。

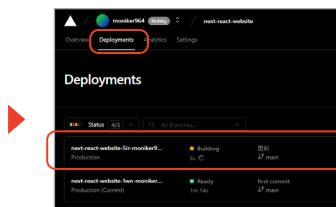
```
$ git push
```

※初めてプッシュを実行した際に「-u」オプションを付けたため、push先のリモートリポジトリは上流ブランチ(upstream)として設定されています。そのため、2回目以降は「git push」と指定するだけで同じリモートリポジトリへプッシュすることができます。

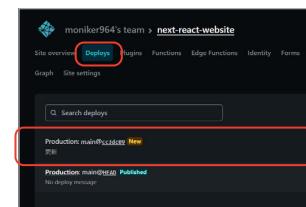
- 修正箇所が、GitHub のリポジトリに反映されます。すると、Vercel や Netlify は GitHub のリポジトリの変化に反応し、自動的にデプロイ処理を行います。処理が完了すると、プロジェクトの更新がサイトに反映されます。



GitHubに反映。



Vercelでは「Deployments」、Netlifyでは「Deploys」をクリックすると、自動的にデプロイ処理が行われているのを確認できます。



# 3.5

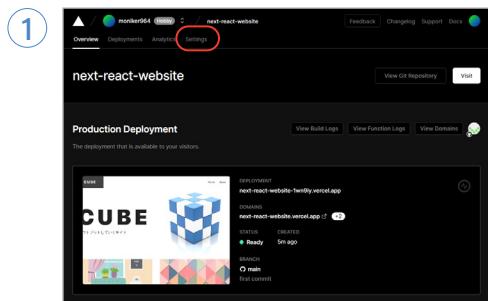
Web Deployment

## microCMSによるサイトの更新

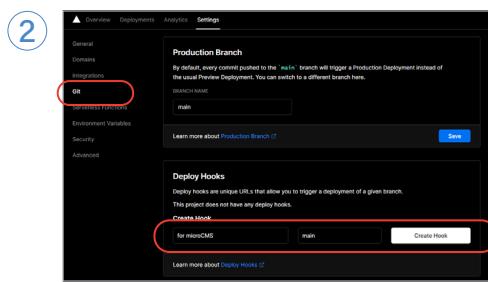
microCMS で記事を追加・修正・削除した場合にも、サイトが更新されるように設定します。そのためには、サイトを公開した Vercel / Netlify 側での準備と、microCMS の設定を行います。

### ❖ Vercelでの準備

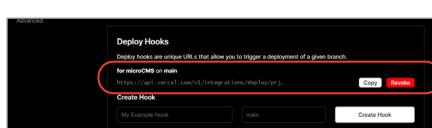
Vercel でサイトを公開している場合は、Vercel でフックを作成します。



公開しているサイトのプロジェクトを開き、「Settings」をクリックします。



「Git」の中の Deploy Hooks でフックを作成します。フックの名前とブランチ (main) を指定して、「Create Hook」をクリックします。

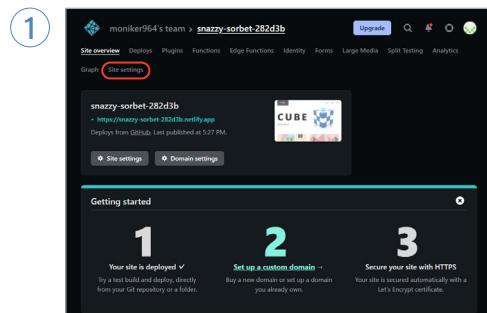


すると、URL が表示されますのでメモしておきます。この URL がこのプロジェクトをデプロイするトリガーとなります。この URL にアクセスすることで、デプロイの処理が実行できます。

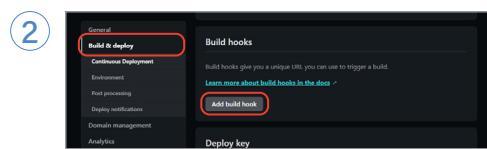
以上で準備は完了です。

## ❖ Netlifyでの準備

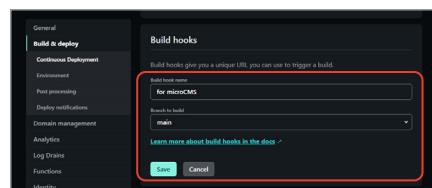
Netlify でサイトを公開している場合は、Netlify でフックを作成します。



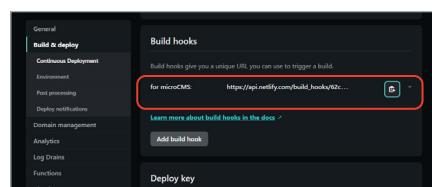
公開しているサイトのプロジェクトを開き、「Site Settings」をクリックします。



「Build & deploy」を選択します。Build hooks という設定項目がありますので、「Add build hook」をクリックします。



フックの名前とブランチ (main) を指定して「Save」をクリックします。

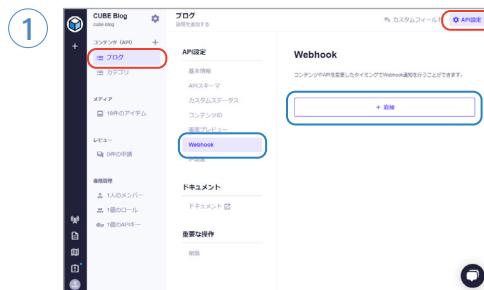


URL が表示されますので、メモしておきます。この URL がこのプロジェクトをデプロイするトリガーとなります。

以上で準備は完了です。

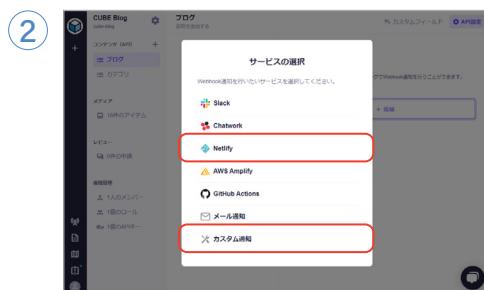
## ❖ microCMSの設定

microCMS では、記事を投稿したり、更新した際に、Vercel や Netlify に用意したフックの URL を呼び出すように設定します。



ブログの記事を管理している「ブログ」の「API 設定」をクリックします。

API 設定が開きますので、「Webhook」を選択して「追加」をクリックします。



「サービスの選択」が表示されますので、Vercel の場合は「カスタム通知」を、Netlify の場合は「Netlify」を選択します。



「基本設定」で設定の名前と、用意した URL を指定します。「カスタム通知」では Vercel に用意した URL を、「Netlify」では Netlify に用意した URL を指定しています。

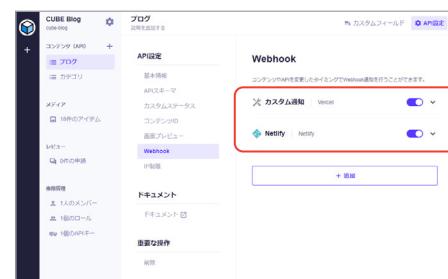
続けて、「通知タイミングの設定」でデプロイを実行するタイミングを指定します。

### Setup 3 サイトの公開

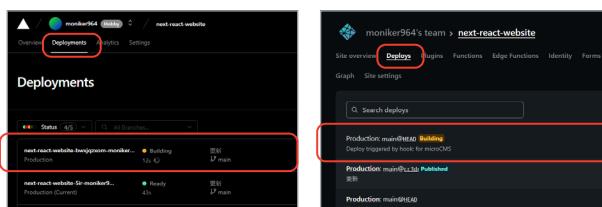


デプロイを実行するタイミングは、「コンテンツの公開時・更新時」と、「公開中コンテンツの削除時」の「コンテンツ編集画面による操作」を選択しています。

最後に、「設定」をクリックすれば完了です。作成した設定が次のように追加されます。



5 記事を投稿・編集してみると、Vercel や Netlify は microCMS からの通知に反応し、自動的にデプロイ処理を行います。処理が完了すると、更新がサイトに反映されます。



Vercelでは「Deployments」、Netlifyでは「Deploys」を開くと、自動的にデプロイ処理が行われているのを確認できます。

**Setup**

# 4

## コンテンツの 準備

Chapter 7 以降の制作を進めるためには、構築するブログサイトのコンテンツが必要です。本書では microCMS を利用して用意します。フィールドの構成と、フィールドに入力するコンテンツのデータは、インポートファイルを使って設定していきます。



Next.js / React

## 4.1

Contents

# microCMSで管理するコンテンツの構成

microCMS でコンテンツを管理するためには、「サービス」を作成し、その中にコンテンツを管理する「API」を用意します。ここでは「CUBE Blog (cube-blog)」というサービスを作成し、その中に「ブログ」と「カテゴリ」の 2 つの API を用意して、ブログの記事とカテゴリを管理します。

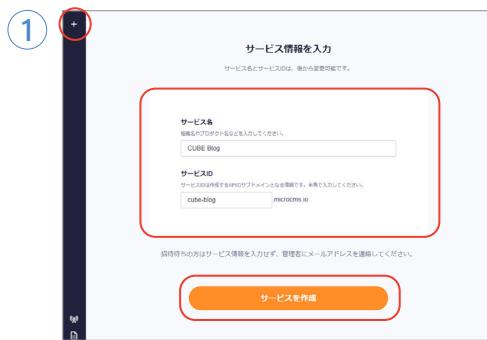
各 API では記事のタイトルやスラッグなど、個々のデータを入力するフィールド (API スキーマ) を用意し、コンテンツを入力できるようにします。



## 4.2 サービスの作成

Contents

まずは、サービスを作成します。



左上の「+」をクリックしてサービスの作成画面を開き、サービス名とサービス ID を指定します。自分の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「CUBE Blog」、サービス ID を「cube-blog」と指定しています。

設定ができたら「サービスを作成」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。

サービスの管理画面が開きます。以上で、サービスの作成は完了です。

画面左上に表示されるサービスのアイコンは、[歯車アイコン] > [基本情報] > [サービス画像] で設定できます。



## 4.3

Contents

# カテゴリAPIの作成と カテゴリーデータのインポート

サービスを作成したら、コンテンツの管理を行う API を追加します。本書のブログサイトでは、記事を管理する「ブログ API」と、カテゴリーを管理する「カテゴリ API」を作成します。

このとき、記事の分類に使用するカテゴリ API はブログ API から参照できるようにしたいので、まずはカテゴリ API から作成し、カテゴリーデータをインポートしていきます。



API を作成するため、「コンテンツ (API)」の横にある「+」をクリックします。テンプレートから選ぶこともできますが、ここではインポートデータを使って作成するため、「自分で決める」をクリックします。



API 名とエンドポイント名を指定します。API 名を「カテゴリ」、エンドポイント名を「categories」と指定して、「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。

④



api-categories.jsonをインポート。

⑤



「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作成します。

ここでは「ファイルインポートする場合はこちらから」をクリックし、本書のダウンロードデータに収録した `api-categories.json` をインポートします。

カテゴリーの名前とスラッグを管理する、2種類のフィールドの設定が追加されます。確認したら「完了」をクリックします。

⑥

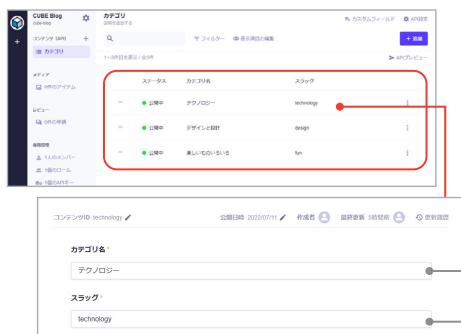


contents-categories.csvをインポート。

「コンテンツがありません」と表示されますので、「インポートする」をクリックします。

本書のダウンロードデータに収録したカテゴリーデータ `contents-categories.csv` を選択し、「インポートを開始」をクリックします。

⑦



3つのカテゴリーがインポートされます。各カテゴリーの編集画面を開くと、カテゴリーの名前とスラッグが入力されています。

以上で、カテゴリ API の作成とデータのインポートは完了です。

カテゴリー名

スラッグ

## 4.4

Contents

# ブログAPIの作成と 記事データのインポート

続けて、ブログの記事を管理する「ブログ API」を作成し、記事データをインポートしていきます。



ブログ API を作成するため、「コンテンツ (API)」の横にある「+」をクリックし、「自分で決める」をクリックします。



API 名を「ブログ」、エンドポイント名を「blogs」と指定して、「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。

④



api-blogs.jsonをインポート。

⑤



記事を管理するためのフィールドを作成します。  
「ファイルインポートする場合はこちらから」をクリックし、本書のダウンロードデータに収録した  
`api-blogs.json` をインポートします。

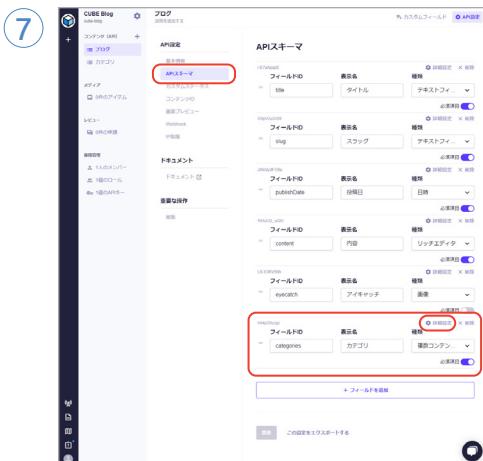
記事のタイトル、スラッグ、投稿日、内容、アイキャッチ、カテゴリーを管理する、6種類のフィールドの設定が追加されます。確認したら「完了」をクリックします。

⑥



「コンテンツがありません」と表示されます。このまま記事データをインポートしたいところですが、問題なくインポートするためには、作成したフィールドの設定を変更する必要があります。

そこで、「API 設定」をクリックします。



「API スキーマ」を選択すると、先ほど作成した 6 種類のフィールドの設定が表示されます。

このうち、設定の変更が必要なのはカテゴリ API を参照させたい「カテゴリ」フィールドです。このフィールドの「詳細設定」をクリックします。



参照先コンテンツを「カテゴリ (categories)」に、一覧画面に表示する項目を「カテゴリ名 (name)」に変更します。

設定ができたら、「閉じる」をクリックして「API スキーマ」の画面に戻り、「変更」をクリックして設定を保存します。



フィールドの設定ができましたので、記事データをインポートします。

左側のメニューで「ブログ」をクリックします。「コンテンツがありません」と表示された画面に戻りますので、「インポートする」をクリックします。



contents-blogs.csvをインポート。

本書のダウンロードデータに収録した記事データ `contents-blogs.csv` を選択し、「インポートを開始」をクリックします。

10

15件の記事がインポートされます。各記事の編集画面を開くと、記事のタイトルなどが入力されています。

タイトル  
スラッグ  
投稿日  
本文(内容)  
アイキャッチ画像  
カテゴリー

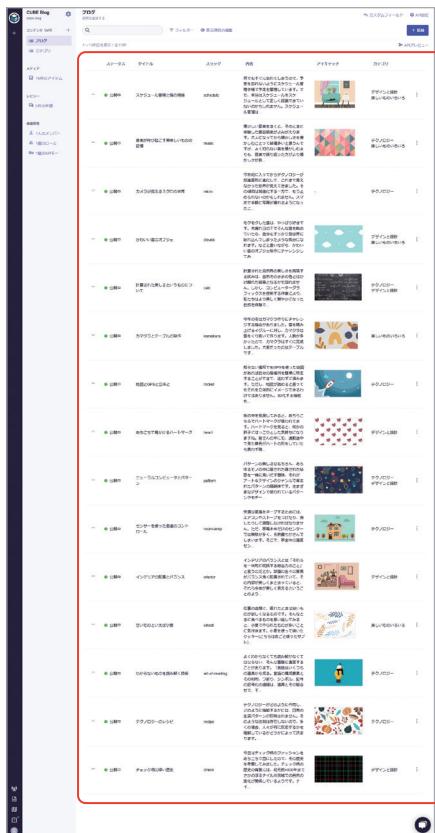
ただし、アイキャッチ画像や、本文中の画像や見出しの設定はインポートできないため、記事ごとに設定していきます。

11

たとえば、「スケジュール管理と猫の理論」という記事では次のように設定します。記事の編集が終わったら、右上の「公開」をクリックして保存します。

見出し2に設定。  
本文中に画像(cat.jpg)を挿入。  
アイキャッチ画像(schedule.jpg)を指定。

12



アイキャッチ画像は、各記事のスラッグと同じファイル名で用意してあります。アイキャッチ画像を設定していくと、ブログ記事の一覧は左のようになります。

なお、アイキャッチ画像が未指定な場合の処理を確認できるようにするために、3件目の記事「カメラが捉えるミクロの世界」のアイキャッチ画像は設定していません。

以上で、ブログサイトの構築に必要なコンテンツの準備は完了です。



アイキャッチ画像以外のフィールドは、入力を必須に設定してあります。

## 4.5

Contents

# コンテンツデータを扱うのに必要な API キーや ID を確認する

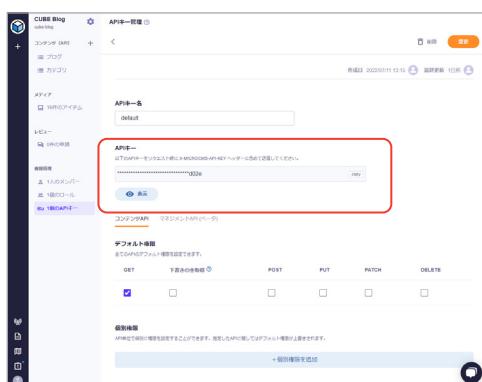
microCMS のコンテンツデータを Next.js で利用するためには、右の情報が必要になります。ここでは、これらの値を確認していきます。

- API キー
- サービス ID
- エンドポイント名
- フィールド ID

## ❖ APIキー



API キーを確認するには、「1 個の API キー」をクリックします。標準で用意された API キーがありますので、クリックします。



API キーの設定が開きます。「表示」をクリックして API キーの値を確認するか、「copy」をクリックしてコピーします。



デフォルト権限で GET が選択されていることも確認しておきます。

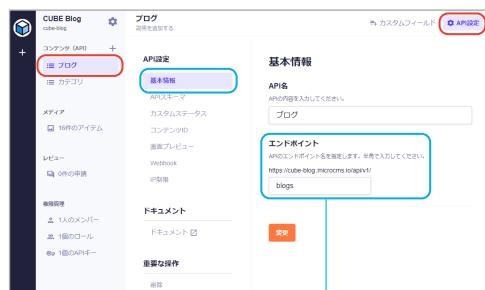


## ❖ サービスID



サービス ID は、画面左上のサービス名の下に表示されています。ここでは「cube-blog」に設定してあります。

## ❖ エンドポイント名



ブログAPIのエンドポイント名。

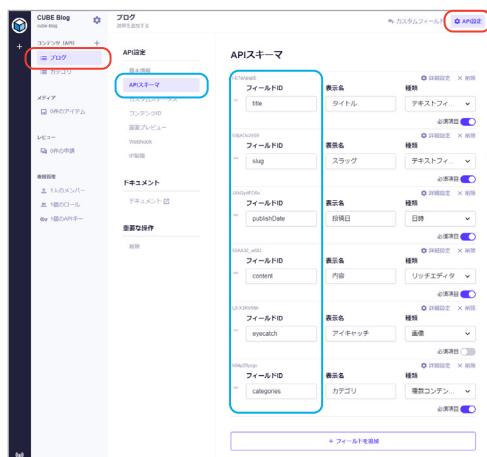
エンドポイント名は、「ブログ」と「カテゴリ」の2つのAPIを作成したときに指定した値です。改めて確認する場合、APIを選択し、右上の「API設定」をクリックします。

API設定の中から「基本情報」を選択すると、エンドポイント名を確認できます。ブログAPIは「blogs」、カテゴリAPIは「categories」に設定してあります。



カテゴリAPIのエンドポイント名。

## ❖ フィールドID



フィールド ID は API ごとに作成したフィールドの ID です。API を選択して右上の「API 設定」をクリックし、「API スキーマ」を選択します。

フィールドの設定がリストアップされますので、「フィールド ID」の値を確認します。ブログ API では次のように設定しています。

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
categories	カテゴリ	複数コンテンツ参照



カテゴリ API では次のように設定しています。

フィールドID	表示名	種類
name	カテゴリ名	テキストフィールド
slug	スラッグ	テキストフィールド

- ! 主要なエンドポイント名やフィールド名は、microCMS の「ブログ」テンプレートをベースに設定しています。ただし、細かな設定はテンプレートとは異なります。各フィールドの詳細設定については次ページを参照してください。

## フィールドの詳細設定

ブログ API とカテゴリ API に用意したフィールドは次のように設定しています。

### ブログAPI: タイトル

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

**重複を許可しない (ユニーク)**  
他のコンテンツで同じが登録された場合にはコンテンツを保存できなくなります。1つのAPIに対して最大で一つのフィールドに設定が可能です。

**特定のバーコードのみ入力を許可する**  
正常表現にマッチした入力のみに制限します。

**文字数を制限する**  
フィールドに入力する文字数の最小文字数、最大文字数を制限します。

### ブログAPI: スラッグ

フィールドID	表示名	種類
= slug	スラッグ	テキストフィールド

**重複を許可しない (ユニーク)**  
他のコンテンツで同じが登録された場合にはコンテンツを保存できなくなります。1つのAPIに対して最大で一つのフィールドに設定が可能です。

**特定のバーコードのみ入力を許可する**  
正常表現にマッチした入力のみに制限します。

**文字数を制限する**  
フィールドに入力する文字数の最小文字数、最大文字数を制限します。

### ブログAPI: 投稿日

フィールドID	表示名	種類
= publishDate	投稿日	日付

**日付指定のみ**  
設定をONにすると時間の指定はできなくなります。

### ブログAPI: 内容

フィールドID	表示名	種類
= content	内容	リッチテキスト

**装飾ボタン**  
使用可能な装飾ボタンを制限することができます。

**改行に<br>タグを用いる**  
オフの場合は<br>タグを表示します。オフの場合は<br>タグで改行分けします。

**画像のレスポンスにwidthとheightを含む**  
オフの場合は<img>タグにwidthとheightが含まれます。

### ブログAPI: アイキャッチ

フィールドID	表示名	種類
= eyecatch	アイキャッチ	画像

**必須項目**  
設定をONにすると入稿時の入力が必須となります。

**説明文**  
入稿画面に表示する説明文です。入稿者にとってわかりやすい説明を入力してください。

**画像のサイズ制限**  
フィールドに入力する画像のwidth、heightの値を制限します。

### ブログAPI: カテゴリ

フィールドID	表示名	種類
= categories	カテゴリ	複数コンテンツ

**参照先コンテンツ**  
カテゴリ (categories)

**一覧画面に表示する項目**  
コンテンツ、またはカスタムフィールドの項目から選択できます。指定した項目が存在しない場合は、コンテンツの順位で表示されます。

**複数コンテンツ別の最小値と最大値を設定する**  
複数コンテンツ別の最小値と最大値を設定します。

### カテゴリAPI: カテゴリ名

フィールドID	表示名	種類
= name	カテゴリ名	テキストフィールド

**重複を許可しない (ユニーク)**  
他のコンテンツで同じが登録された場合にはコンテンツを保存できなくなります。1つのAPIに対して最大で一つのフィールドに設定が可能です。

**特定のバーコードのみ入力を許可する**  
正常表現にマッチした入力のみに制限します。

**文字数を制限する**  
フィールドに入力する文字数の最小文字数、最大文字数を制限します。

### ブログAPI: スラッグ

フィールドID	表示名	種類
= slug	スラッグ	テキストフィールド

**重複を許可しない (ユニーク)**  
他のコンテンツで同じが登録された場合にはコンテンツを保存できなくなります。1つのAPIに対して最大で一つのフィールドに設定が可能です。

**特定のバーコードのみ入力を許可する**  
正常表現にマッチした入力のみに制限します。

**文字数を制限する**  
フィールドに入力する文字数の最小文字数、最大文字数を制限します。

## Setup

# 5

## デザインデータ の利用

本書で構築するブログサイトのデザインは、Figma のデザインデータとして収録しています。作成するページのデザインに加えて、デザインシステム（コンポーネント＆デザイントークン）もセットで収録していますので、実装時の参考にしてください。

ここでは、デザインデータを Figma で開き、デザインの設定を確認する方法を解説します。



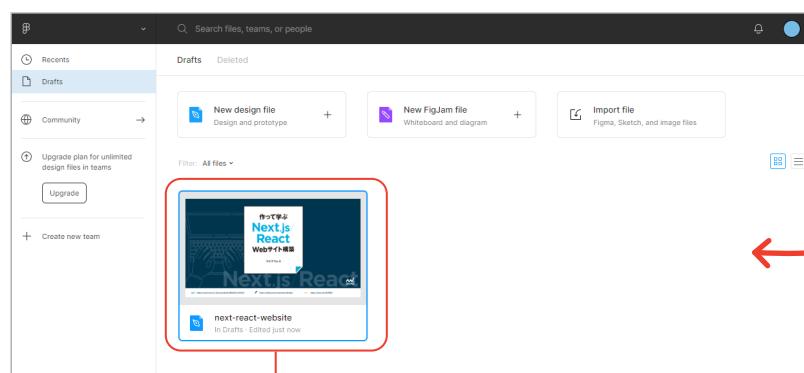
# Next.js / React

# 5.1

Design Data

## Figmaでデザインデータを開く

本書のダウンロードデータには Figma のデザインデータ `next-react-website.fig` を収録しています。このデータを使うためには、Figma のファイルブラウザにドラッグ & ドロップしてインポートします。これでオンラインに保存され、使用できるようになりますので、ダブルクリックして開きます。



The screenshot shows the Figma interface with the sidebar open. A red box highlights the imported file 'next-react-website.fig' in the 'Drafts' section. A red arrow points from the text below to this box.

インポートされたデザインデータをダブルクリックします。



The screenshot shows a web browser window displaying the 'Cover' page of the imported design. The URL in the address bar is `https://book.mosha.jp/react/design-data/next-react-website`.

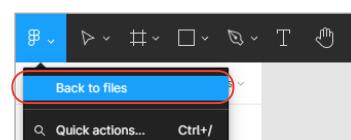
「Cover」ページが開きます。



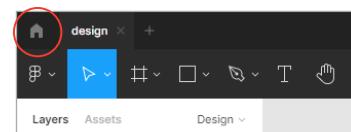
A small icon of the Figma logo is shown next to the file name 'next-react-website.fig'. A red arrow points from the text below to this icon.

Figmaのファイルブラウザにドラッグ & ドロップ。

### ファイルブラウザの開き方

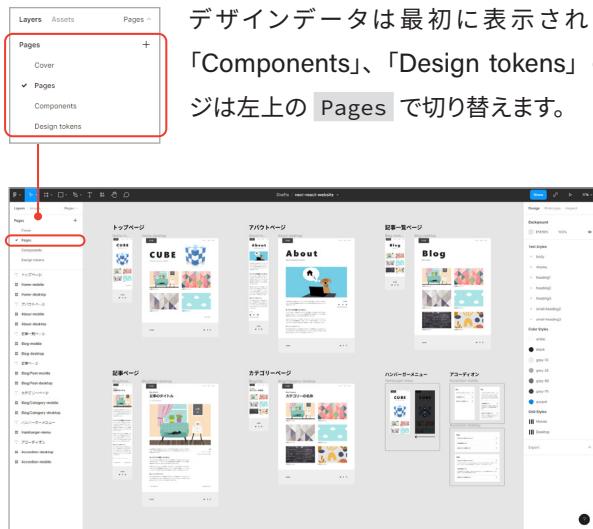


ブラウザの場合:  
メニューから「Back to files」を選択。



デスクトップアプリの場合:  
ホームアイコンをクリック。

## ❖ デザインデータの構成

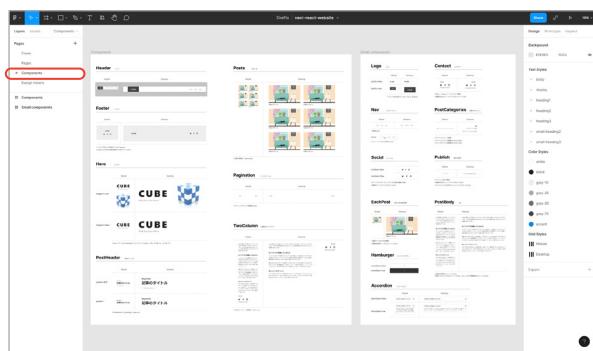


デザインデータは最初に表示される「Cover」ページに続けて、「Pages」、「Components」、「Design tokens」の3つのページで構成しています。各ページは左上の Pages で切り替えます。

### Pages

作成するブログサイトのページ構成とデザインです。

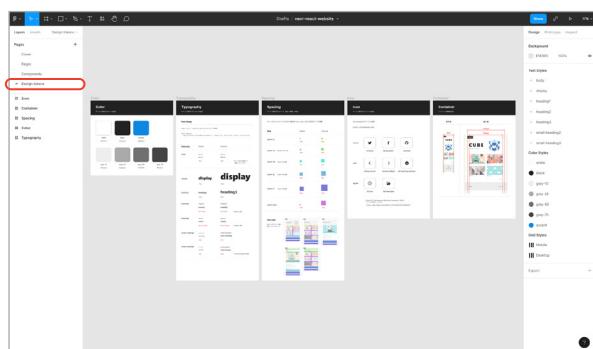
「Components」に用意した Figma のコンポーネントを組み合わせ、ページごとにモバイル版とデスクトップ版の両方を作成しています。



### Components

ブログサイトで繰り返し使う構成要素は Figma のコンポーネントとして作成し、このページにまとめています。

コンポーネントごとに、モバイルとデスクトップでデザインを切り替えるようにしています。



### Design tokens

ブログサイト全体で使用する色やフォントサイズなどの基本的な設定を、デザイントークンとしてまとめています。

フォントサイズとスペースは、モバイルとデスクトップでサイズを変える設計にしています。

## 5.2

Design Data

# デザインの設定を確認する

Figma を使ったデザインデータは、テキスト、画像、図形といったオブジェクトの組み合わせで構成され、グループ化されたオブジェクトは必要に応じてフレームやコンポーネントになっています。

こうしたオブジェクトやグループは「レイヤー」と呼ばれ、左側のレイヤーパネルの **Layers** で構成を確認できます。レイヤーの種類はアイコンで示されており、レイヤーをクリックするか、**Move** ツールでデザインを直接クリックして選択できます。

選択したレイヤーのデザインの設定は、右側のプロパティパネルの **Design** で確認します。

The screenshot shows the Figma application interface. On the left is the **Layers** panel, which lists various components and their sub-elements. A red circle highlights the **Move** tool icon at the top of the toolbar. In the center, a layer named "CUBE" is selected, indicated by a purple selection box around its text and image. A red arrow points from the "Layers" label to the Layers panel. Another red arrow points from the "Moveツール" (Move tool) label to the Move tool icon in the toolbar. To the right is the **Design** panel, which displays the properties for the selected "CUBE" layer. A red arrow points from the "Design" label to the Design panel. At the bottom, a legend identifies the icons used in the Layers panel:

#	フレーム (Frame)	T	テキスト (Text)
□	グループ (Group)	☒	画像 (Image)
❖	コンポーネント (Component)	□	図形 (Shape)
◇	インスタンス (Instance)	=	オートレイアウト (Auto layout)

A red box highlights the "Design" panel, and a red arrow points from the "選択したレイヤーのデザインの設定" (Selected layer's design settings) label to it.

## ❖ グループ化された個々のレイヤーの選択

グループ化された個々のレイヤーはダブルクリックで選択していくことができます。たとえば、ヒーローを構成するタイトルは次のように選択できます。なお、Ctrl + クリック / ⌘+ クリック (Deep select) でダイレクトに選択することも可能です。



タイトルをダブルクリック。グループ化されたヒーロー全体が選択されます。

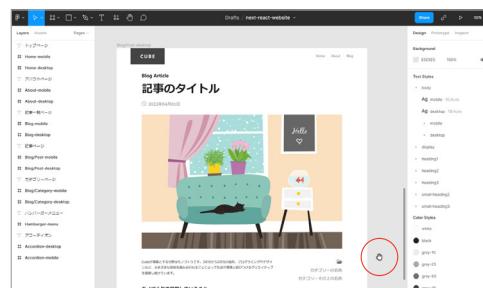
続けて、タイトルをダブルクリック。グループ化されたテキスト全体が選択されます。

さらに、タイトルをダブルクリック。タイトルが選択されます。

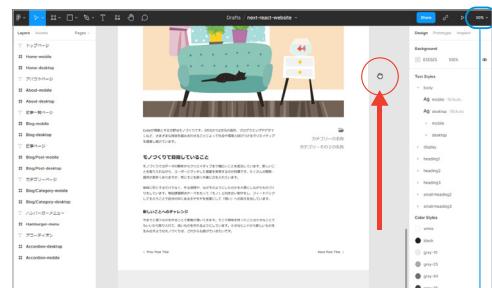
- ! グループ化を解除する場合、グループを選択した状態で、左上の メニューから [Object > Ungroup selection] を選択します。

## ❖ 表示範囲と倍率の変更

デザインの表示範囲はハンドツールで変更します。スペースキーを押すとポインターが手のアイコンになりますので、その状態でドラッグします。表示倍率は右上のズームオプションまたは + / - キーで変更できます。



スペースキーを押すとポインターが手のアイコンになります。



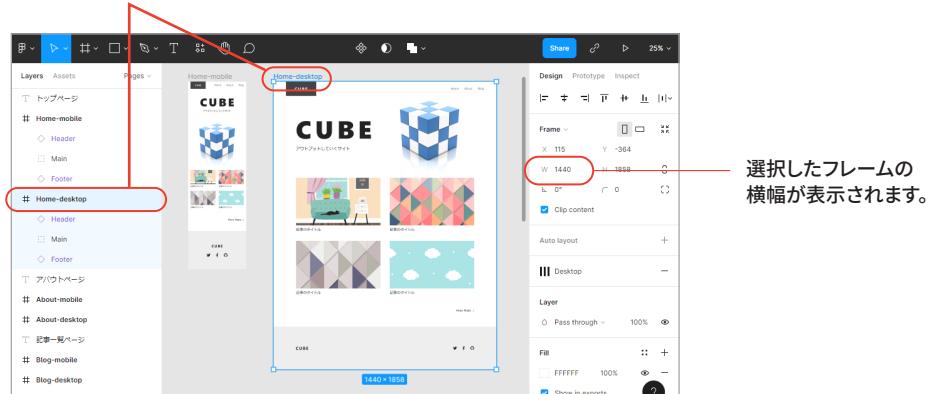
そのままドラッグすると表示範囲が変わります。

ズームオプション

## ❖ Webページの横幅

Web ページの横幅は、トップレベルのフレームを選択して確認します。モバイル版は 375 ピクセル、デスクトップ版は 1440 ピクセルの横幅に設定されています。

トップレベルのフレームを選択。



! Figma の「フレーム」レイヤーは複数のレイヤーをグループ化したものです。ただし、「グループ」レイヤーと異なり、レイアウトや位置指定の基準として扱われます。キャンバス上に直接配置されたトップレベルのフレームはアートボードとして機能し、ページを構成します。

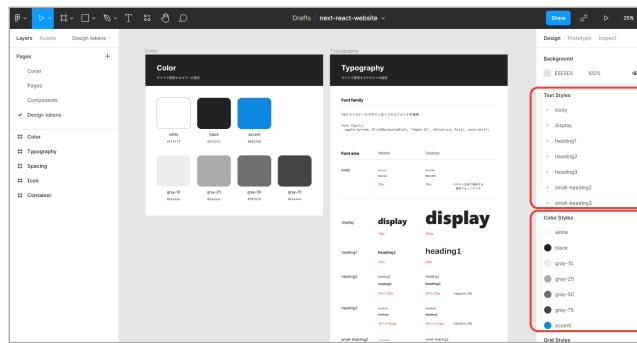
## ❖ レイヤーの間隔(スペースのサイズ)

レイヤーの間隔や距離を計測するためには、計測の起点となるレイヤーを選択した状態で、Alt キー (Windows) または option キー (macOS) を押しながら他のレイヤーにポインターを重ねます。



## ❖ テキストと色のスタイル

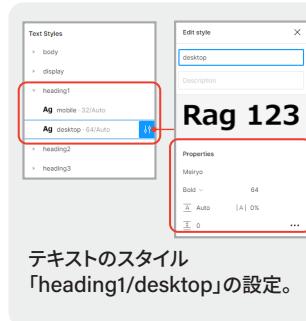
デザイントークンとしてまとめた、サイト全体で使うテキスト（タイポグラフィ）と色の設定は、Figma の Styles（スタイル）として管理しています。Esc キーを押してすべてのレイヤーの選択を解除すると、右側のパネルにすべてのスタイルが表示され、設定を確認できます。



The screenshot shows the Figma interface with the 'Design tokens' panel open. It displays two main sections: 'Color' and 'Typography'. The 'Color' section shows a grid of color swatches with names like 'white', 'black', 'gray-10', etc. The 'Typography' section shows font families, font sizes, and various heading styles. A red box highlights the 'Text Styles' section under 'Typography', which lists 'body', 'display', 'heading1', 'heading2', 'heading3', and 'small-heading'. Another red box highlights the 'Color Styles' section, which lists 'white', 'black', 'gray-10', 'gray-25', 'gray-50', 'gray-75', and 'accent'.

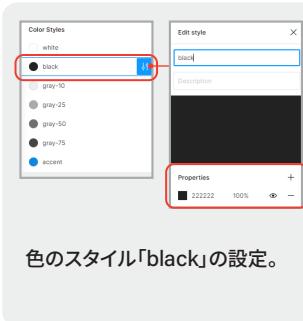
テキストのスタイル

色のスタイル



This screenshot shows the 'Text Styles' panel for the 'heading1/desktop' style. It lists properties like 'Font Family' (Meiryo), 'Font Weight' (Bold), and 'Font Size' (64). A red box highlights the 'Properties' section where the color is set to 'black'.

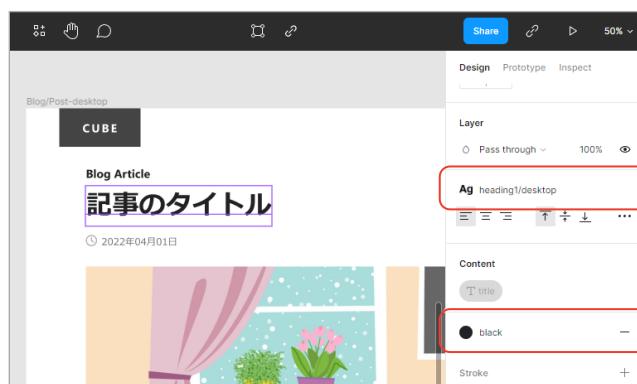
テキストのスタイル  
「heading1/desktop」の設定。



This screenshot shows the 'Color Styles' panel for the 'black' style. It lists other styles like 'white', 'gray-10', 'gray-25', etc. A red box highlights the 'Properties' section where the color is set to '#222222'.

色のスタイル「black」の設定。

スタイルにカーソルを重ねて  をクリックすると、左のようにテキストや色の設定を確認できます。



The screenshot shows the Figma interface with the 'Design' panel open. A specific layer is selected, and its properties are shown in the right-hand sidebar. The 'Content' section shows the text '記事のタイトル' and its style 'Ag heading1/desktop'. The 'Stroke' section shows the color is set to 'black'. A red box highlights the 'Content' section.

レイヤーを選択すると、レイヤーが使用しているテキストと色のスタイル名を確認できます。

たとえば、記事ページのタイトルを選択すると、テキストのスタイルは「heading1/desktop」、色のスタイルは「black」を使用していることがわかります。

## ❖ コンテンツの横幅を示すレイアウトグリッドのスタイル

デザイントークンのコンテナ（サイト全体で使うコンテンツの横幅）の設定は、Figma のレイアウトグリッドのスタイルとして管理しています。テキストや色のスタイルと同じように、**Esc** キーを押してすべてのレイヤーの選択を解除すると、右側のパネルでグリッドのスタイルを確認できます。

レイアウトグリッドのスタイル

モバイル用のグリッドスタイル。

デスクトップ用のグリッドスタイル。

スタイルにカーソルを重ねて をクリックすると、左のようにグリッドの設定を確認できます。

デスクトップ用のスタイルでは 2 つの最大幅のグリッドが登録してあります。

グリッドのスタイルは各ページを構成するフレームに適用してあります。そのため、左上の メニューから [View > Layout grids] を選択すると、レイアウトグリッドの赤色と青色のガイドライン（レイアウト用の補助線）が表示されます。

ページに適用された  
レイアウトグリッドのスタイル

## ❖ コンポーネントとプロパティ

ブログサイトで使用するパートは、繰り返し使えるように Figma のコンポーネントとして作成してあります。コンポーネントを元に作成したレイヤーは  で示され、「インスタンス」と呼ばれます。インスタンスはメインコンポーネント（元のコンポーネント）の設定を共有しますが、メインコンポーネントが用意したプロパティを使うと、インスタンスごとにテキストの内容を変えたり、画像表示をオン・オフするといったことが可能です。

たとえば、トップページ、アバウトページ、記事一覧ページに配置したヒーローは、Hero コンポーネントで作成したインスタンスです。用意された `title` と `subtitle` プロパティでテキストの内容を変え、`imageOn` プロパティで画像表示をオン・オフしています。

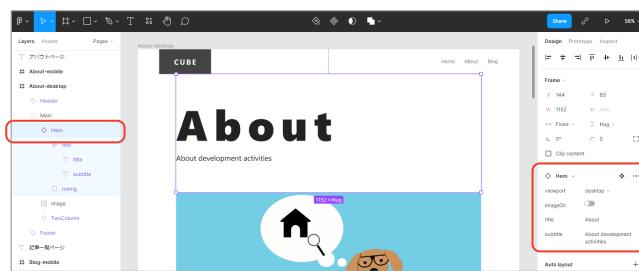


Heroコンポーネントで作成したインスタンス

The screenshot shows the Figma interface with the Home page selected. A red box highlights the 'Hero' component in the left sidebar. On the right, the properties panel shows the configuration for the Hero instance on the Home page:

- `viewport`: desktop
- `imageOn`:
- `title`: CUBE
- `subtitle`: アップロットしていく  
サイト

トップページのHeroインスタンスのプロパティの設定。

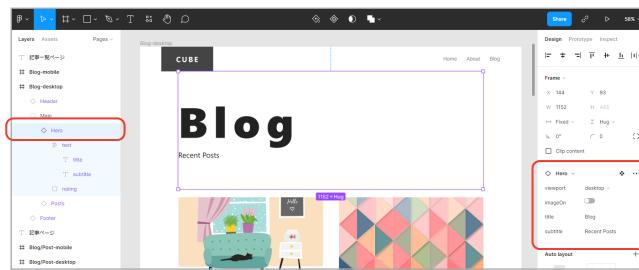


AboutページのHeroインスタンスのプロパティの設定。

The screenshot shows the Figma interface with the About page selected. A red box highlights the 'Hero' component in the left sidebar. On the right, the properties panel shows the configuration for the Hero instance on the About page:

- `viewport`: desktop
- `imageOn`:
- `title`: About
- `subtitle`: About development  
activities

アバウトページのHeroインスタンスのプロパティの設定。



BlogページのHeroインスタンスのプロパティの設定。

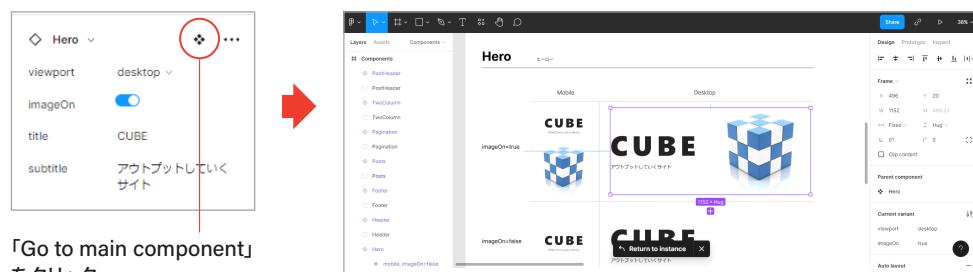
The screenshot shows the Figma interface with the Blog page selected. A red box highlights the 'Hero' component in the left sidebar. On the right, the properties panel shows the configuration for the Hero instance on the Blog page:

- `viewport`: desktop
- `imageOn`:
- `title`: Blog
- `subtitle`: Recent Posts

記事一覧ページのHeroインスタンスのプロパティの設定。

## Setup 5 デザインデータの利用

なお、テキストや色などのデザインの設定はインスタンス側で確認できますが、プロパティの右上に表示された✖をクリックすると、元のメインコンポーネントを開くこともできます。

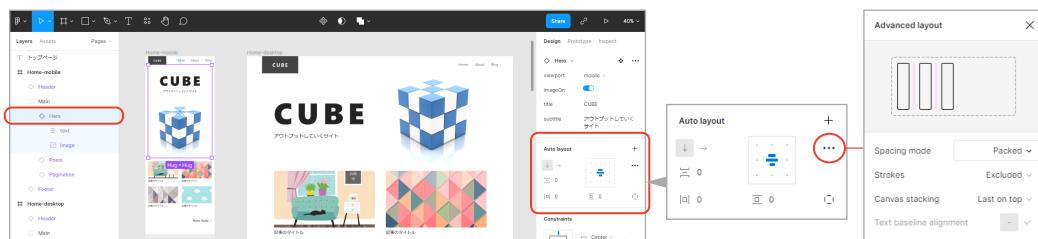


「Go to main component」  
をクリック。

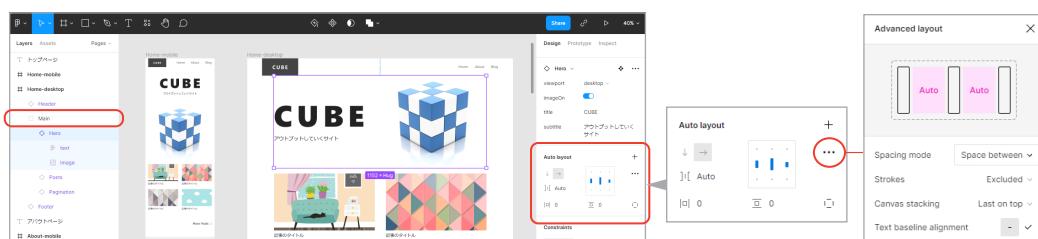
## ❖ オートレイアウト

オートレイアウト (Auto layout) を利用したレイヤーでは、Flexbox を適用したときと同じように子レイヤーが縦または横並びになり、間隔や位置揃えが調整されています。

たとえば、Hero コンポーネントではオートレイアウトを利用して、子レイヤーのテキストと画像をモバイル版では縦並びに、デスクトップ版では横並びにしています。



モバイル版のHeroコンポーネントのオートレイアウトの設定。



デスクトップ版のHeroコンポーネントのオートレイアウトの設定。

■著者紹介

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  
『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』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・ブラックティスブック 1～8』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

## 作って学ぶ Next.js/React Web サイト構築

# セットアップ PDF

---

2022年7月30日 ver.1.0 発行