

Azure 静的 Web アプリを試してみる

Next.js でコンテンツのソースは CMS の場合

概要

Azure 静的 Web アプリを試してみたのでメモ。

- 背景
- テスト用サイトと目標
- 初期設定
- 設定の調整
- デプロイを試す
- その他
- 所感

背景

Azure 静的 Web アプリを試すことにした経緯

GitHub Actions + GitHub Pages を使っているのだが

[mardock\(このサイトをビルドしているウェブアプリ\)](#)では「Next.js アプリのソースとビルドされたコンテンツが同一リポジトリにある」ため、いろいろと取り回しが悪いと感じている。

現在、上記の問題や「プレビューモード」「ステージング環境」等への対応で GitHub Actions + Pages でのビルドを試行錯誤している。

その一環として、今回は Azure 静的 Web アプリを試してみた。

現状の課題(余分な作りこみ)

mardock は PDF の生成等を行うためビルド処理が重い。よって、以下のように対応している。

- GitHub でパブリックリポジトリ + Actions を利用してビルド
- ビルド結果を GitHub Pages へエクスポート

この方法では「パブリックリポジトリを使っているので制限はきつくない」のだが「ステージング」等は別途仕組みを用意する必要がある。

Azure 静的 Web アプリの利用

Azure 静的 Web アプリではソースを GitHub リポジトリとすることで、「デプロイのワークフローに GitHub Actions」が利用可能。そのことから、以下の点を期待し試すことにした。

- 既存のビルド環境を利用できる
 - 現状では [ghcr](#) 等も使ったりしているので、その辺は継続したい
- 運用、ステージング環境は Azure 側が受け持ってくれる

テスト用サイトと目標

テストに使うサイト(Next.js アプリ)

mardock(このサイト)でいきなり試すのは少しハードルが高いので、テスト用として「以前に Next.js + GitHub Pages の組み合わせを試してみたサイト」を利用する。

- Next.js アプリのソース: [GitHub - hankei6km/test-nextjs-az-swa-01](https://github.com/hankei6km/test-nextjs-az-swa-01)
- フレームワーク: React(Next.js)
- コンテンツのソース: microCMS

目標

静的 Web アプリはできることが多いので、とりあえず今回は以下を目標とする。

- GitHub Pages と Azure 静的 Web アプリに同一の内容でデプロイ
- プルリクエストでステージング環境へデプロイされることを確認
- ルートとロールは簡単に試すのみ
- カスタムドメイン、API ルート(Azure Functions)までは確認しない

なお、テストに使うリポジトリはすでに静的サイトとしてデプロイ(エクスポート)できるようになっているので、その辺の設定などについては割愛する。

初期設定

Next.js アプリの設定と静的 Web アプリの作成

Next.js アプリ側での設定

GitHub Pages 用の設定が流用できるが、以下の点には注意が必要。

- 静的にビルドされるようにしておく (`fallback` 等は指定しない)
- `assetPath` にリポジトリ名を設定していたら解除
- npm スクリプトの `build:azure` で `next export` を実行する (`build` の後に実行される)

```
"scripts": {  
  "build": "aspida && aspida-mock && next build",  
  "build:azure": "next export",  
},
```

Azure で静的 Web アプリを作成

- Azure ポータル「すべてのサービス」等から「静的 Web アプリ / 新規」を選択
- 「プロジェクトの詳細」などは状況にあわせて入力
- 「デプロイの詳細」で「GitHub アカウントでサインイン」を選択
 - 前述のリポジトリを選択
 - 「分岐」は `main` (他の値でも可)
- 「ビルドの詳細」が入力できるようになるので
 - 「プリセット」は「React」を選択

ビルドの詳細項目

「プリセット」で「React」を選択すると以下の項目を入力することになる。

- アプリの場所: Next.js であれば `next.config.js` の場所でもいいもよう(今回は `/`)
- API の場所: Azure Functions の API があれば指定(Next.js の API ルートではない)
- 出力先: `next export` されるディレクトリ(今回は `out`)

新規追加画面のスクリーンショット

Microsoft Azure

リソース、サービス、ドキュメントの検索 (G+)

ホーム > 静的 Web アプリ > 静的 Web アプリ

既定のディレクトリ

+ 新規 設定 ビューの管理

任意のフィールドのフィルター...

名前 ↑

- draftlint
- test-nextjs-azswa-01

静的 Web アプリの作成

組織 * hankei6km

リポジトリ * test-nextjs-az-swa-01

分岐 * main

ビルドの詳細

ビルドおよびリリース用の GitHub Actions ワークフロー ファイルを作成するための値を入力します。ワークフロー ファイルは、後で GitHub リポジトリで変更できます。

ビルドのプリセット React

アプリの場所 * /

API の場所 例: "api", "functions" など...

出力先 out

これらのフィールドには、アプリの種類 of 既定のプロジェクト構造が反映されます。アプリに合わせて値を変更してください。

GitHub リポジトリに加えられる変更

Azure 上で新規追加の処理を実行すると、ソースとなったりリポジトリには以下のような変更が加えられる。

- デプロイトークンがリポジトリシークレットとして追加される
- デプロイ用のワークフローが追加される(
`.github/workflow/azure-static-web-app-XXXXX.yaml` など)

GitHub Pages へデプロイしていたなら多くの場合は上記ワークフローでデプロイできるが、今回は CMS の API シークレット等を利用するための調整が必要となる。

設定の調整

CMSのシークレットと Webhook の利用

CMS API シークレットの利用

コンテンツのソースを CMS から取得する場合、リポジトリや Environment に登録したシークレットを利用することが多い。

今回のアプリでも `env` 経由でビルドステップへ設定する。

```
- name: Build And Deploy
  id: builddeploy
  uses: Azure/static-web-apps-deploy@v1
  with:
    azure_static_web_apps_api_token: ${{ secrets.AZURE_STATIC_WEB_APPS_API_TOKEN_CALM_BAY_089CD8B10 }}
    repo_token: ${{ secrets.GITHUB_TOKEN }} # Used for Github integrations (i.e. PR comments)
  # << snip >>
  env:
    API_BASE_URL: ${{ secrets.API_BASE_URL }}
    GET_API_KEY: ${{ secrets.GET_API_KEY }}
```

Webhook の利用(コンテンツ更新対応)

CMS 側からの更新通知は `repository_dispatch` 経由になっているので、必要であれば Azure のワークフローにイベントを追加する。なお、デフォルトブランチで起動されるので、「分岐」に `main` を選択したアプリで定義すると混乱が少ない(かな)。

```
on:
  push:
    branches:
      - main
  pull_request:
    types: [opened, synchronize, reopened, closed]
    branches:
      - main
  repository_dispatch:
    types: [ghp_az-swa]

jobs:
  build_and_deploy_job:
    if: github.event_name == 'push' || (github.event_name == 'pull_request' && github.event.action != 'closed') || github.event_name == 'repository_dispatch'
    runs-on: ubuntu-latest
    name: Build and Deploy Job
```

デプロイを試す

環境別のデプロイ

デプロイするには

デプロイはリポジトリ上でプルリクエストを操作することで行われる。

The screenshot shows a GitHub Actions workflow for the repository 'hankei6km / test-nextjs-az-swa-01'. The workflow is titled 'Use dark theme Azure Static Web Apps CI/CD #6'. The 'Jobs' section lists 'Build and Deploy Job' as completed, with a 'Close Pull Request Job' pending. A detailed view of the 'Build and Deploy Job' shows it succeeded 11 minutes ago in 2m 51s, with the following steps:

- > ✓ Set up job
- > ✓ Build Azure/static-web-apps-deploy@v1
- > ✓ Run actions/checkout@v2
- > ✓ Build And Deploy
- > ✓ Post Run actions/checkout@v2
- > ✓ Complete job

どの環境へデプロイされるか

プルリクエストへの操作によりデプロイされる「環境」が変化する。

- 「作成(オープン)」 「プッシュ」などは「ステージング環境にデプロイ」される
- 「クローズ」は「ステージング環境が削除され」「運用環境に新しいビルドがデプロイ」される

なお、今回はプルリクエストとは別に Webhook にも対応させているが、この場合は「運用環境にデプロイ」される。

各環境にデプロイされたサイトのアドレス

アドレス(URL)は Azure ポータルからアプリの「環境」を開くことで確認できる(Bot による PR へのコメントでも個別に確認可能)。



ホーム > test-nextjs-az-swa01

test-nextjs-az-swa01 | 環境 ...
静的 Web アプリ

検索 (Ctrl+F) << 最新の情報に更新 | 削除

概要
アクセス制御 (IAM)
タグ
設定
構成
Application Insights
カスタムドメイン
関数
環境
ロール管理
ID
ホスティングプラン
ロック

ステージング環境は、pull request が生成されたときに自動的に作成され、pull request のマージ後に運用環境に昇格されます。

運用
アプリごとに 1 つの運用環境があります。その環境の状態と最終更新時刻をここで確認できます。

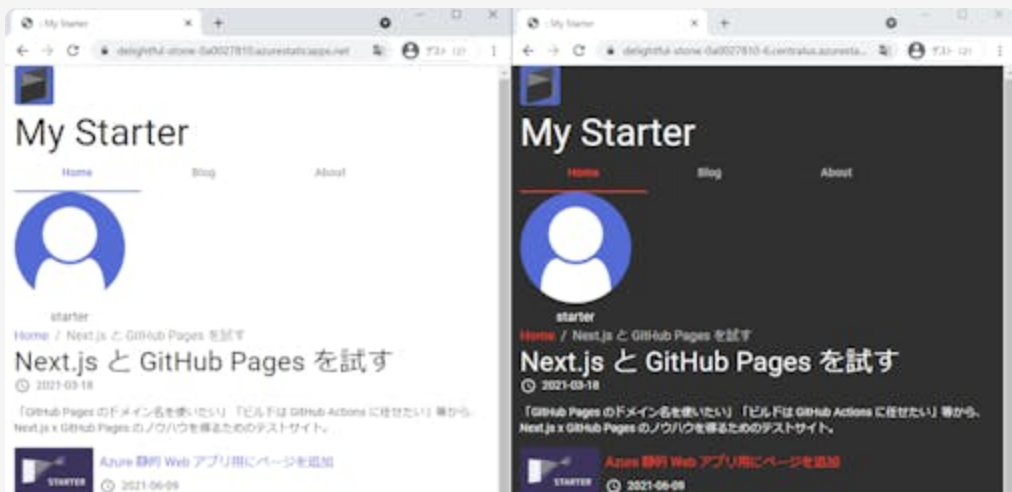
分岐	最終更新日時	状態	参照
main	2021年6月9日 16:24:41 JST	Ready	参照

ステージング
ステージング環境を作成するため、リンクされたリポジトリに対して pull request を開きます。

<input type="checkbox"/> タイトル	分岐	最終更新日時	状態	参照
<input type="checkbox"/> Use dark theme	csb-nv824	2021年6月9日 16:51:02 JST	Ready	参照
<input type="checkbox"/> Routes wuth Roles	topic/routes-with-roles	2021年6月10日 12:15:12 JST	Ready	参照

デプロイされたアプリの表示(目標達成)

- GitHub Pages: <https://hankei6km.github.io/test-nextjs-az-swa-01/>
- Azure 静的 Web アプリ
 - 運用: <https://delightful-stone-0a0027810.azurestaticapps.net/>
 - ステージング: <https://delightful-stone-0a0027810-6.centralus.azurestaticapps.net/>



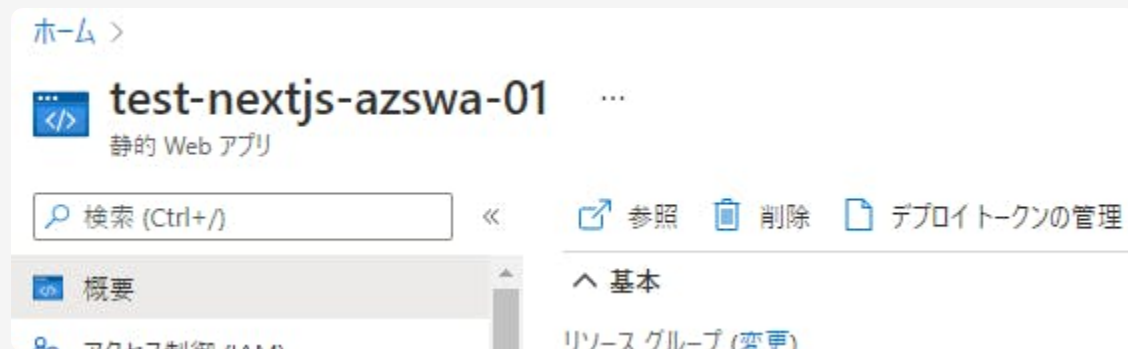
その他

デプロイトークン、ルートとロール

デプロイトークンの確認

デプロイトークンはリポジトリのシークレットへ自動的に登録されているため通常は確認しないが、「Environment シークレットへ登録したい」等の場合もある。

アプリの「概要」を開き「デプロイトークンの管理」で確認できる。



ルートとロール

せっかくなのでと少し試した感じでは「思っていたより柔軟にできる = それなりに設定を考える必要がある」ので、箇条書き程度に。




- ルートは [Next.js の Shallow ルーティング](#)には対応していない(まあそうですよね)
- ログイン(プロバイダーへの接続)はデフォルトで有効になっている
 - 無効にするにはルート設定で **404** を返すようにする
- ログイン時のクライアントは「Azure Static Web Apps(<https://identity.azurestaticapps.net/>)」

- ロール(とユーザー)は静的 Web アプリ内での独自管理
 - ロールは事前に作成しない
 - ユーザーの招待時に指定したロールがタグ付けされるイメージ
 - ユーザーが招待に応じると一覧に登録される
 - ユーザーを削除してもログイン中はロールが付与されたまま
- ユーザーは招待されていなくてもアプリの `.auth/login/***` を開くことで(静的 Wevb アプリに)ログインできる
 - 組み込みロールのみが割り当てられる
 - アプリの「ロール管理」に該当ユーザーは表示されない

- [Custom authentication in Azure Static Web Apps | Microsoft Docs](#)
ではまた様子が異なるもよう
 - 参考: [Azure Static Web Apps を Azure AD B2C で認証 - BEACHSIDE BLOG](#)

ユーザーの招待

| ロール管理 ...

 招待  最新の情報に更新 |  削除

ユーザー ID によるフィルター処理...

認証プロバイダー

ユーザー ID ↑↓

認証プロバイ

結果がありません。

招待リンクの作成



招待リンクを使用すると、特定のユーザーにドメインへのアクセス権を付与できます。リンクの有効期限を指定できます。

認証プロバイダー *

GitHub



ユーザー ハンドル *

hankei6km



ドメイン *

delightful-stone-0a0027810.azurestaticap...



ロール *

az_swa01_test






招待の有効期限 (時間) * ⓘ

1

ロール管理内のユーザー

ユーザーが招待に応じた時点で自動的に登録される。

1 | ロール管理 ...

 招待  最新の情報に更新 |  削除

ユーザー ID によるフィルター処理...

認証プロバイダー == すべて

ロール == すべて

ユーザー ID ↑↓

認証プロバイダー ↑↓

ロール ↑↓

hankei6km

GitHub

az_swa01_test

anonymous

authenticated

所感

良い

- 自動生成されたワークフローを元にカスタマイズしやすい
 - `next build & next export` 以外でのデプロイも比較的簡単にできそう
- アプリは「リポジトリのブランチ」と関連付いている
 - ブランチ別に「独立した運用とステージング」環境を作成できる
 - ただし上限には注意
- ルートとロールによるアクセス制御
 - 簡単なアクセス制御は静的 Web アプリだけで対応できそう

良くない

- ワークフローをどこまでカスタマイズしてよいのか分かりにくい
 - ブランチのフィルター変えても動くけどよいの？等
- 管理する対象が [Azure と GitHub に分かれる](#)
 - パブリックリポジトリでゆるくやってる分にはよいのだけど
- 環境の履歴が残らない
 - クローズしたプルリクエストの環境は確認できない等
- Next.js で利用するには少し工夫が必要

今後

全般的には好感触(「良くない」も「Vercel と比べたら」という部分が多い)なので、以下のようなことも実験してく予定。

- ghcr + サービスコンテナでビルドしているサイトのデプロイ
- 他サービスと組み合わせたプレビューモード対応

参考

- [Azure Static Web Apps のドキュメント | Microsoft Docs](#)