MyWeb ちえんじろぐ

2023part2

個人的に気になったWebサービスやツール、ライブラリ、フレームワークを紹介する本

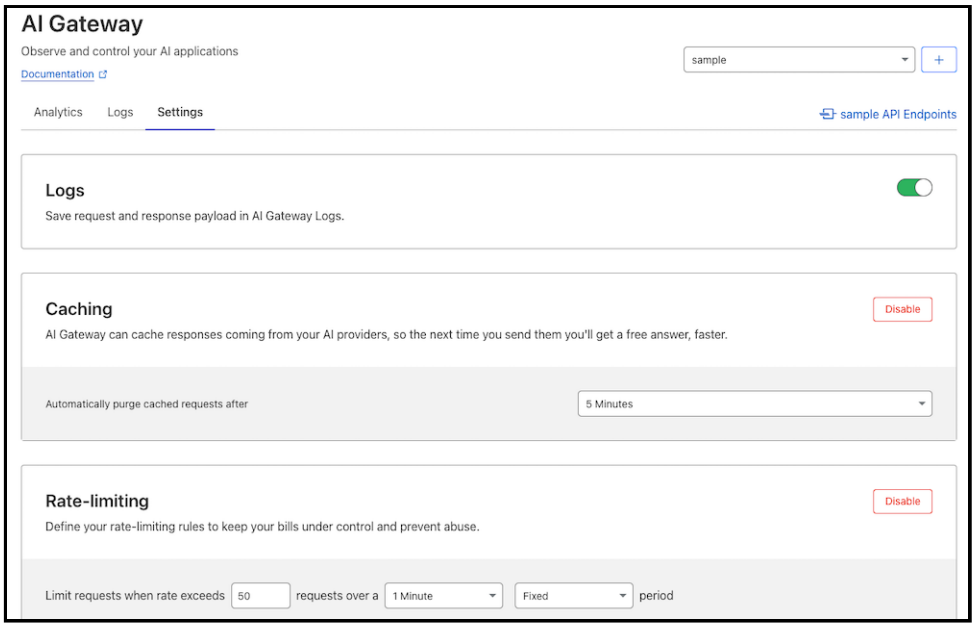
2023年6月 ~ 2023年11月まとめ

AI編

Cloudflare AI Gateway

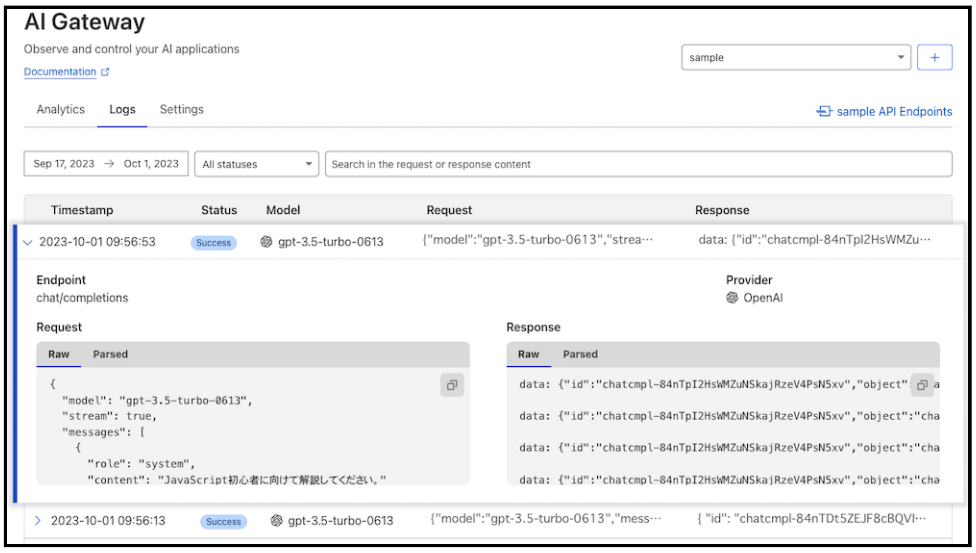
検証日: 2023年10月5日

Cloudflare AI GatewayはHugging FaceやChatGPTへのAPIリクエストを制限したり、キャッシングしたりできるサービスです[1]。



Cloudflare AI Gateway

このサービスを使うと、同じクエリを叩かれたときにキャッシュを返すことができます。クエリの分析は現状できるようになってませんが、リクエスト状況などもモニタリングできます。ログも以下のように、リクエストとレスポンスを確認できるようになってます。



Cloudflare AI Gateway Log

既存のChatGPT APIを使った処理を置き換える際は、Cloudflareにリクエストを送る必要があるため、リクエスト側のパラメーターも変わります。ChatGPTから置き換える時はquery内にパラメーターを移す必要があります。

```
curl -X POST https://gateway.ai.cloudflare.com/v1/xxx/yyy \n
  --header 'Content-Type: application/json' \n
  --data '[
  {
    "provider": "openai",
    "endpoint": "chat/completions",
    "headers": {
      "authorization": "Bearer <OPENAIのAPIキー>",
      "content-type": "application/json"
    },
    "query": {
      "model": "gpt-3.5-turbo",
      "stream": true,
      "messages": [
        {
          "role": "user",
          "content": "What is Cloudflare?"
        }
      ]
    }
  }]'
```

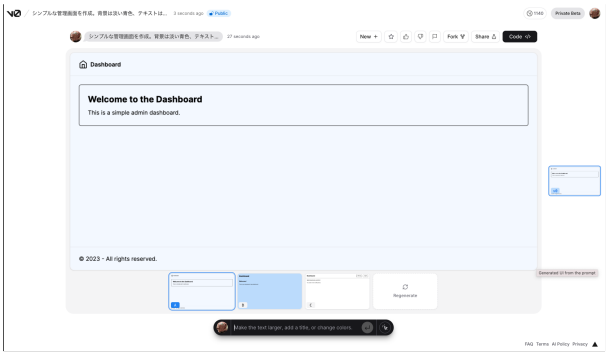
---

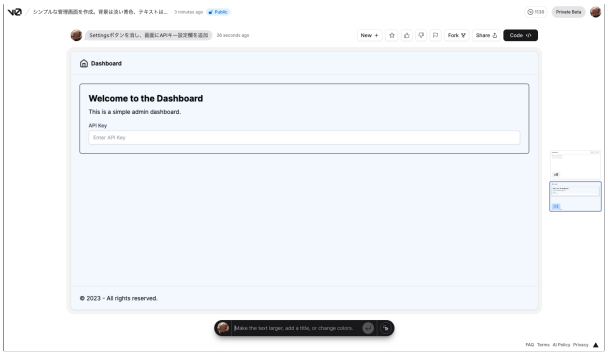
1. https://blog.cloudflare.com/ja-jp/announcing-ai-gateway-ja-jp/↩

自然言語でUIを構築するv0

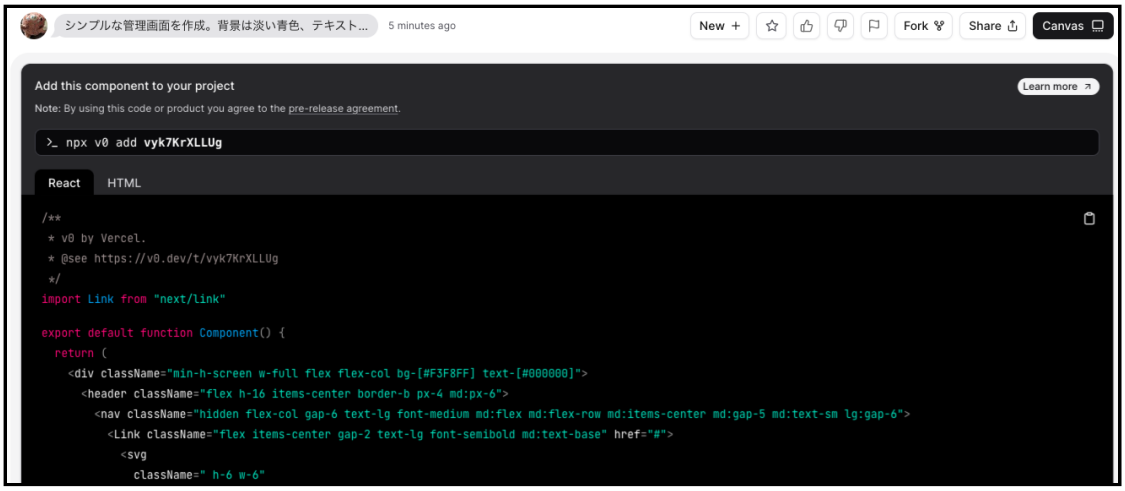
検証日: 2023年10月31日

Vercelが自然言語からUIを構築するサービスを公開しました[1]。例えば、「シンプルな管理画面を作成。背景は淡い青色、テキストは黒色にする。」とプロンプトを入力すると、左下の画像のように3種類ほどUIが生成されます。そこからさらに、プロンプトを入力することで、UIを改修できます (右下の図では、「Settingsボタンを消し、画面にAPIキー設定欄を追加」とプロンプトを入力しました)。





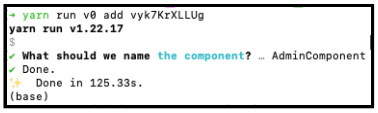
こちらはUIの構築だけでなく、Next.js用のコードとHTMLも生成してくれます。アイコンなどもSVGで生成してくれます。



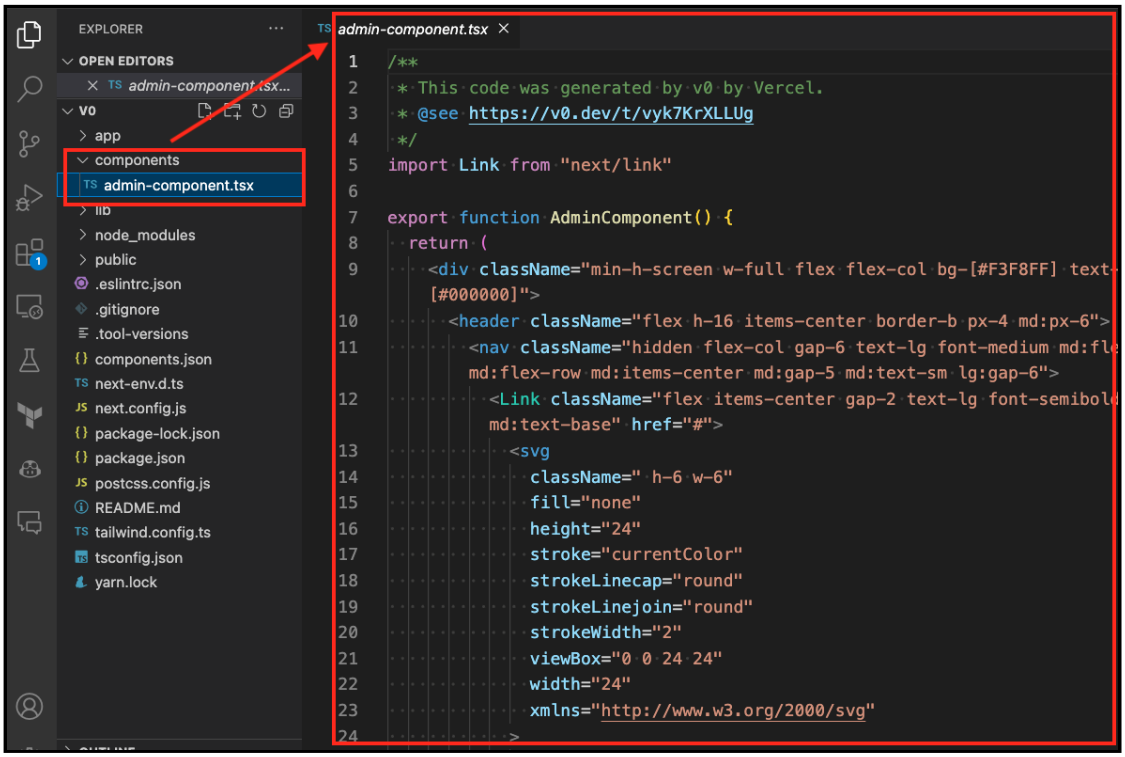
v0で生成されたコード

また、現状 (2023.10.31) はNext.jsでしか利用できませんが、以下のコマンドでコンポーネントの追加ができます。

```
$ yarn add v0
$ yarn run v0 add vyk7KrXLLUg
```



v0で生成されたコードを取り込む



v0が生成したコンポーネント

自然言語からUIとコードを生成。1コマンドでプロジェクトにコンポーネントを追加するというワークフローは真新しく便利に感じますが、現状は制約が多いです。Next.js以外でも利用できるようになったり、自然言語から生成したUIをFigmaのように微調整できるようになったりすると、より便利になりそうです。

---

1. https://v0.dev/↩

WebAssembly編

最近はWebAssemblyを利用したライブラリも増えてきており、画像処理、音声処理などの従来だとできなかったり、負荷が高かった処理がブラウザ上で可能になってきました。また、Can I useを見る限りだと、デスクトップ、モバイル含め、ほぼ全てのブラウザでWebAssemblyが利用できるようになっています (Safari含む)。この本では、WebAssemblyを利用したライブラリを特に紹介しませんが、個人的に面白いと思うものに焦点を当てて、紹介していきます。

Moonbit

検証日: 2023年11月3日

MoonbitはWebAssemblyにコンパイルすることを目的とした言語です。こちらで生成するWebAssemblyはクラウドやCDN Edgeで動作させることを目的としています。言語設計では、GolangやRustの影響を受けており、パッケージシステムはGolang, 型推論やパターンマッチングはRustを参考にしているらしいです[1]。

例えば、フィボナッチ数列の第n項目の値を出す関数は以下のようになります。パターンマッチの書き方がRustと似ています。

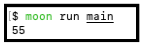
main.mbt

```
fn fib(n : Int) -> Int {
  match n {
    0 => 0
    1 => 1
    _ => fib(n - 1) + fib(n - 2)
  }
}

fn init {
  println(fib(10))
  /// => 55 (フィボナッチ数列の第10項目の値)
}
```

実行は以下のコマンドでできます。

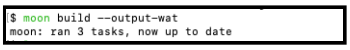
```
$ moon run main
55
```



moonbitの実行結果

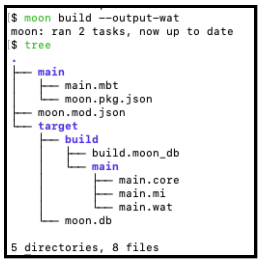
ビルドは以下のコマンドでできます。targetフォルダ配下にWebAssemblyが生成されます。

```
$ moon build --output-wat
moon: ran 2 tasks, now up to date
```



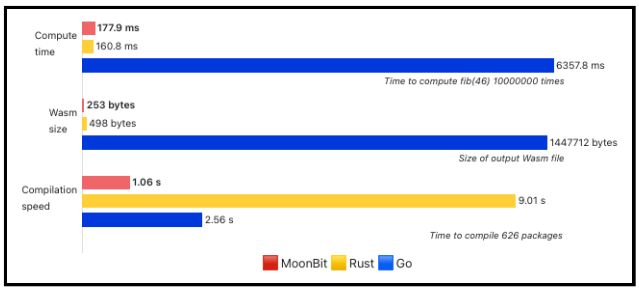
moonbitでビルドした結果

ここでは示してませんが、関数のimportやexportにも対応しているため、適度な粒度でファイル分割もできます。



moonbitのビルド生成物

ちなみに、GolangやRust, Moonbitでそれぞれ WebAssemblyを生成し、そのサイズや実行速度を比較したものがこちらです[2]。実行速度はRustより少し遅いですが、サイズやコンパイル時間は最適化されていることが分かります。個人的にはデバッグのやりやすさはどうなんだろうと思ってますが、その辺りはイマイチ見えませんでした。



moonbitのパフォーマンス (公式ブログより引用)

---

1. https://www.moonbitlang.com/blog/first-announce↩
2. https://github.com/moonbitlang/moonbit-docs/tree/main/benchmark/fibonacci↩

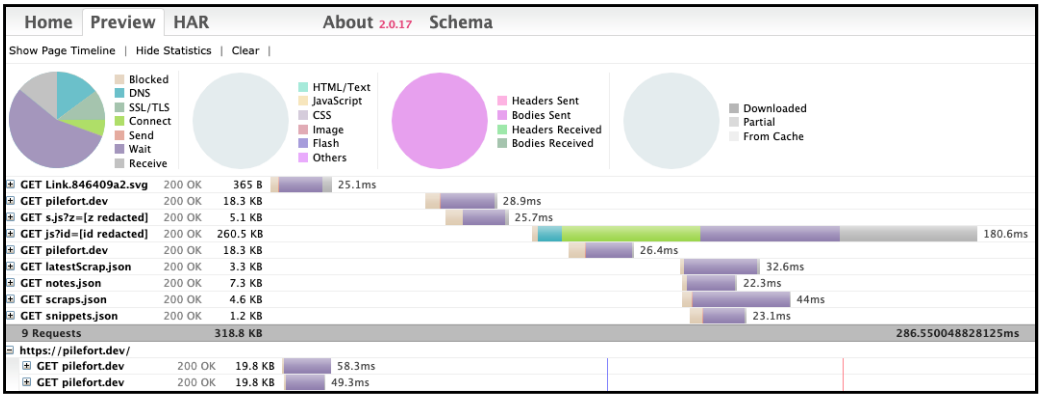
ツール編

har-sanitizer

検証日: 2023年11月4日

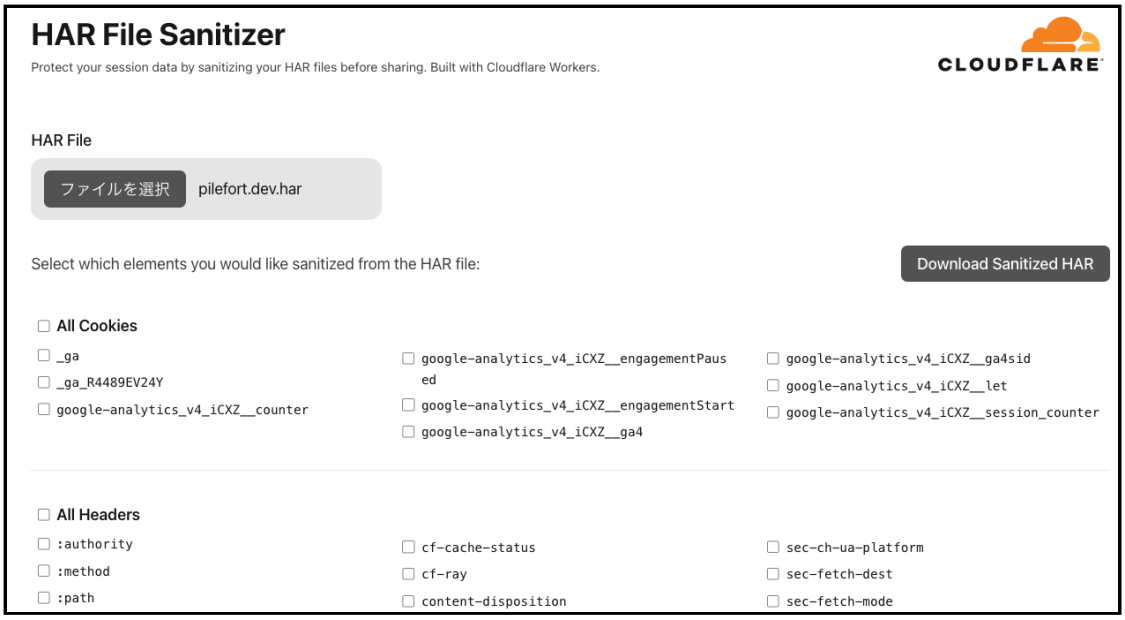
HARファイル (HTTP Archieve formatファイル) はブラウザとサイト間のネットワーク通信の履歴を記録したファイルです。Salesforceが紹介しているサイトを利用すると、以下のように、ログを確認できます[1]。これを確認することで、パフォーマンス改善や不具合調査ができるらしいです (調査のために使ったことはありません)。

こちらのHARファイルはcookieの情報やリクエスト情報、レスポンス情報が含まれており、個人情報が盛り盛りのファイルになります。AWSなどで不具合連絡をすると、共有を依頼されるケースがありますが、そのままでは渡せないため、個人情報のマスクが必要になります。



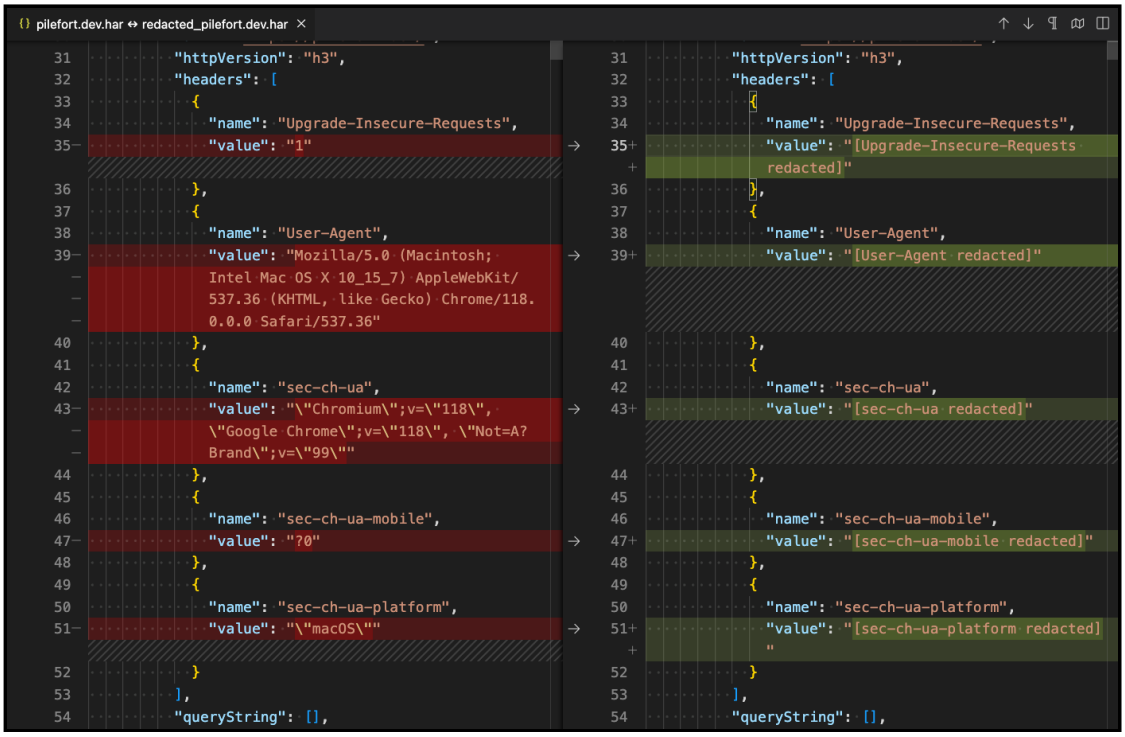
softwareishard.com/har/viewer での表示結果

この度、CloudflareよりHARファイル上の個人情報のマスクをしてくれるツールが提供されました[2]。こちらはCloudflareによりホストもされており、以下のようにHARファイルの項目を読み取り、マスクしたい項目を選択できるようになっています。



HAR File Sanitizer

マスクできる項目はCookies, Headers, Query String Parameters, Post Body Params, Mime Typesとなっています。使用しているブラウザ情報やリクエストヘッダー、レスポンスの中身もマスクできます。マスクすると、以下のようになります。



元のHARファイル (左) とマスクされたHARファイル (右)

ちなみに、Googleが提供するHAR Analyzer[3]だと、マスクした文字列に含まれる`[`が原因でエラーになります。相手が使ってるViewerや解析ツールに依存する部分もありますが、HARファイルのマスクは必要になるケースが多いので、便利なツールではあります。

---

1. https://help.salesforce.com/s/articleView?id=000385988&type=1↩
2. https://github.com/cloudflare/har-sanitizer↩
3. https://toolbox.googleapps.com/apps/har_analyzer/↩

フレームワーク編

Next.js 14

server action 対応がサポートされた React Canaries がマージされた

検証日: 2023年10月31日

Reactは2023年5月頃にReactの準安定機能をReact Canariesとしてリリースするようになりました[1]。こちらは実験機能とは異なり、ほぼ採用が決まった機能が入っています。Next.jsではこちらのReact Canariesを利用しつつ、Reactの先行機能を利用できるようにしてます。

この度、React Canariesでserver actionがリリースされました[2]。このリリースにより、以下のようにNext.jsでもserver actionが利用可能になりました[3]。これを利用すると、`'use server'`で宣言された関数がブラウザではなく、サーバー上で実行されます。

動きとしては、送信ボタンを押すと`<form>...</form>`で設定した項目がサーバーに送信される。サーバーに送信されてから、`'use server'`で宣言された関数が実行され、結果がフロントに描画される。という動きになってそうですが、ドキュメントも少なく詳細不明です。

app/page.tsx

```
export default function Page() {
  async function create(formData: FormData) {
    'use server';
    const id = await createItem(formData);
  }

  return (
    <form action={create}>
      <input type="text" name="name" />
      <button type="submit">Submit</button>
    </form>
  );
}
```

ちなみに、server actionの実行状態を取得するメソッド[4]やserver actionの結果に基づいて画面描画するメソッド[5]も追加されてます (statusとstateはエイリアスっぽい名前なので、すごく紛らわしい)。今後どのような使われ方をするのか分かりませんが、Reactの先行機能が今後もNext.jsに取り込まれて、使えるようになりそうです。

```
function Submit() {
  const status = useFormStatus();
  return <button disabled={status.pending}>Submit</button>
}

export default App() {
  return (
    <form action={action}>
      <Submit />
```

```
    </form>
  );
}
```

```
async function increment(previousState, formData) {
  return previousState + 1;
}

function StatefulForm({}) {
  const [state, formAction] = useFormState(increment, 0);
  return (
    <form>
      {state}
      <button formAction={formAction}>Increment</button>
    </form>
  )
}
```

---

1. https://react.dev/blog/2023/05/03/react-canaries↩
2. https://github.com/facebook/react/blob/main/CHANGELOG-canary.md↩
3. https://nextjs.org/blog/next-14#server-actions-stable↩
4. https://react.dev/reference/react-dom/hooks/useFormStatus↩
5. https://react.dev/reference/react-dom/hooks/useFormState↩

サービス編

Pulumi

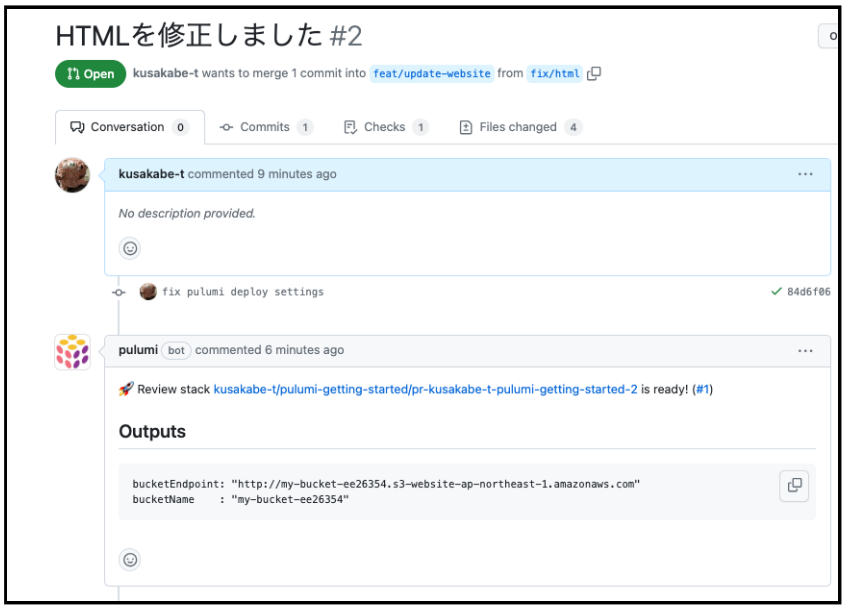
PulumiはTerraformやAWS CloudFormationなどと同じインフラをコード化するためのツールです。Terraformなどと異なり、PulumiではGoやTypeScriptなど型のある言語を使ってリソース作成・管理ができます。

Review Stacks

調査日: 2023年10月23日

Review StacksはPulumiで作成したprスタックの内容をPRごとにコピーして再現する機能です[1]。つまり、PRごとにAWSのリソースを自動作成し、PRを閉じると環境を自動破棄できます。ちなみにスタックとは、pulumiの設定ファイルに記述したAWSリソースを1つにまとめたものです (いわゆるステージング環境、本番環境みたいなもの)。

Review Stackの設定は少々大変ですが、何か差分があるたびに環境を作ってくれるため、実際に触りながら動作確認できます。ちなみに、環境構築はせず、設定ファイルの変更でどのような差分が出るかだけ表示することもできます。



Preview Stackのサンプル

Restore Stacks

調査日: 2023年10月23日

こちらは現状はEnterpriseプランのみとなります。Restore StacksはPulumiで削除したリソースを復元する機能です[2]。Pulumi上でリソースを復元した後に、以下のコマンドを使うことで、設定ファイルの更新もできます。

```
pulumi config refresh
```

---

1. https://www.pulumi.com/blog/restore-stacks↩

番外編

TypeScript 5.2 using

検証日: 2023年10月1日

TypeScript5.2で追加されたusingについて紹介します。usingはECMAScriptに新しく追加予定の変数宣言で、変数がスコープから外れたときの後処理を自動化できます。これにより、一時ファイルを作成して削除する処理やDBに接続してコネクションを閉じる処理などを管理しやすくなります。

usingで後処理を自動化するためには、`[Symbol.dispose]()`に処理を書く必要があります。

```
function usingTest(id: number): Disposable {
  console.log(id);

  return {
    [Symbol.dispose]() {
      console.log(`${id}''`);
      // doWork();
    },
  }
}
```

ちなみに、同期処理をしたい場合は以下のようになります。

```
function usingAsyncTest(id: string): AsyncDisposable {
  console.log(id);

  return {
    async [Symbol.asyncDispose]() {
      console.log(`${id}''`);
      // doWork();
    },
  };
}
```

実行順を確認するために以下のようなコードを書いたとします。これを実行すると、1, 2, 3, 4, 4', 5, 6, 6', 2'となります。usingを使うとスタックのようなイメージでSymbol.disposeの処理が保留されます。スコープから外れたタイミングで直近に宣言されたusingから処理を実行していきます。

```
startUsingTest()

function startUsingTest() {
  console.log(1) // => 1
  using a = usingTest(2) // => 2
  {
    console.log(3) // => 3
    using b = usingTest(4) // => 4, 4'
  }
  console.log(5) // => 5
```

```
  using c = usingTest(6) // 6, 6'
  // => 2'
}
```

現在はまだリリースされて間もないため、usingを使うのに手間がかかります。例えば、viteでusingを使うには、vite.config.tsで以下の設定が必要です。

vite.config.ts

```ts
import { defineConfig } from "vite";
import react from "@vitejs/plugin-react";

export default defineConfig({
  plugins: [
    react({
      babel: {
        plugins: [
          "@babel/plugin-proposal-explicit-resource-management",
        ],
      },
    }),
  ],
});
```

また、usingを使うファイルにおいて、以下の宣言も必要です。まだ手間がかかるため使いづらいですが、今後に期待しています。

```ts
// @ts-ignore
Symbol.dispose ??= Symbol("Symbol.dispose");
// @ts-ignore
Symbol.asyncDispose ??= Symbol("Symbol.asyncDispose");
```