ユーザーガイド

# AWS Amplify ホスティング

# AWS Amplify ホスティング: ユーザーガイド

# Table of Contents

# AWS Amplify ホスティングへようこそ

Amplify ホスティングは、継続的なデプロイでフルスタックのサーバーレスウェブアプリケーションをホストするための Git ベースのワークフローを提供します。Amplify は、アプリケーションを AWS グローバルコンテンツ配信ネットワーク (CDN) にデプロイします。このユーザーガイドでは、Amplify ホスティングを使い始めるために必要な情報を提供します。

## サポートされるフレームワーク

Amplify ホスティングは、次のような多くの一般的な SSR フレームワーク、シングルページアプリケーション (SPA) フレームワーク、静的サイトジェネレーターをサポートしています。

SSR フレームワーク

- Next.js
- Nuxt
- コミュニティアダプター付き Astro
- コミュニティアダプター付き SvelteKit
- カスタムアダプターを使用する任意の SSR フレームワーク

SPA フレームワーク

- React
- Angular
- Vue.js
- Ionic
- Ember

静的サイトジェネレーター

- Eleventy
- Gatsby
- Hugo
- Jekyll

- VuePress

# Amplify ホスティングの機能

機能ブランチ

新しいブランチを接続して、フロントエンドとバックエンドの本番稼働環境とステージング環境を管理します。

カスタムドメイン

アプリケーションをカスタムドメインに接続。

プルリクエストのプレビュー

コードレビュー中に変更をプレビューします。

エンドツーエンドテスト

エンドツーエンドのテストでアプリの品質を向上させます。

パスワードで保護されたブランチ

パブリックアクセス可能にせずに新しい機能を使用できるように、ウェブアプリをパスワードで保護します。

リダイレクトと書き換え

リライトとリダイレクトを設定して SEO ランキングを維持し、クライアントのアプリの要件に基づいてトラフィックをルーティングします。

アトミックデプロイ

アトミックデプロイにより、お使いのウェブアプリはすべてのデプロイが完了したときに更新されるようになり、メンテナンスウィンドウが必要なくなります。これにより、ファイルが正しくアップロードされないというシナリオが解消されます。

# Amplify ホスティングの使用開始

Amplify ホスティングを始めるには、Amplify ホスティングへのアプリのデプロイの概要 のチュートリアルをご覧ください。チュートリアルを完了すると、Git リポジトリ (GitHub、BitBucket、GitLab、または AWS CodeCommit) にウェブアプリを接続し、継続的デプロイで Amplify ホスティングにデプロイする方法がわかります。

# バックエンドの構築

AWS Amplify Gen 2 では、バックエンドを定義するための TypeScript ベースのコードファースト開発者エクスペリエンスが導入されています。Amplify Gen 2 を使用して、バックエンドを構築し、アプリに接続する方法については、「Amplify ドキュメント」の「バックエンドの構築と接続」を参照してください。

Amplify Gen 2 のコードファーストアプローチの詳細については、ワークショップ「AWS スタジオ」ウェブサイトの「Amplify Gen 2 Workshop」を参照してください。この包括的チュートリアルでは、React と Next.js を使用してサーバーレスアプリケーションを構築し、Amplify Gen 2 Data and Auth ライブラリと Amplify UI ライブラリを使用してアプリケーションに機能を追加する方法について説明します。

CLI と Amplify Studio を使用して Gen 1 アプリのバックエンドを構築するためのドキュメントが必要な場合は、「Gen 1 Amplify ドキュメント」の「バックエンドの構築と接続」を参照してください。

# Amplify ホスティングの料金

AWS Amplify は、使用した分に対してのみ課金します。詳細については、「AWS Amplify 料金」を参照してください。

# Amplify ホスティングへのアプリのデプロイの概要

Amplify ホスティングの仕組みを理解できるように、以下のチュートリアルでは、Amplify がサポートする一般的な SSR フレームワークを使用して作成されたアプリケーションの構築とデプロイについて説明します。

チュートリアル

- Next.js アプリを Amplify ホスティングにデプロイする
- Nuxt.js アプリを Amplify ホスティングにデプロイする
- Astro.js アプリを Amplify ホスティングにデプロイする
- SvelteKit アプリを Amplify ホスティングにデプロイする

# Next.js アプリを Amplify ホスティングにデプロイする

このチュートリアルでは、Git リポジトリから Next.js アプリケーションを構築およびデプロイする方法について説明します。

このチュートリアルを始める前に、次の前提条件を完了してください。

にサインアップする AWS アカウント

まだ AWS のお客様でない場合は、オンラインの手順に従って を作成 AWS アカウントする必要があります。サインアップすると、Amplify やアプリケーションで使用できるその他の AWS サービスにアクセスできます。

アプリケーションの作成

「Next.js ドキュメント」の create-next-app 手順を使用して、このチュートリアルで使用する基本的な Next.js アプリケーションを作成します。

Git リポジトリを作成する

Amplify は、GitHub、Bitbucket、GitLab、および をサポートしています AWS CodeCommit。`create-next-app` アプリケーションを Git リポジトリにプッシュします。

## ステップ 1: Git リポジトリの接続

このステップでは、Git リポジトリの Next.js アプリケーションを Amplify ホスティングに接続します。

アプリを GitHub リポジトリに接続するには

1. Amplify コンソールを開きます。
2. 現在のリージョンに最初のアプリをデプロイしている場合は、デフォルトで [AWS Amplify] サービスページから開始できます。

   ページの上部で、[アプリケーションをデプロイする] を選択します。
3. [Amplify で構築を開始] ページで、自分の Git リポジトリプロバイダーを選択し、[次へ] を選択します。

   GitHub リポジトリの場合、Amplify は GitHub Apps の機能を使用して Amplify へのアクセスを承認するようになります。GitHub App のインストールと認証の詳細については、GitHub リポジトリへの Amplify アクセスの設定 をご参照ください。

   > **Note**
   >
   > Bitbucket、GitLab、または で Amplify コンソールを承認すると AWS CodeCommit、Amplify はリポジトリプロバイダーからアクセストークンを取得しますが、 AWS サーバーにはトークンを保存しません。Amplify は、特定のリポジトリにのみインストールされているデプロイキーを使用してリポジトリにアクセスします。
4. 「リポジトリブランチを追加」ページで、以下の操作を行います。
   a. 接続するリポジトリの名前を選択します。
   b. 接続するリポジトリブランチの名前を選択します。
   c. [次へ] を選択します。

## ステップ 2: ビルド設定を確認する

Amplify は、デプロイしているブランチに対して実行するビルドコマンドのシーケンスを自動的に検出します。このステップでは、ビルド設定を確認して承認します。

アプリのビルド設定を確認するには

1. [アプリ設定] ページで、[ビルド設定] セクションを見つけます。

[フロントエンドのビルドコマンド] と [ビルド出力ディレクトリ] が正しいことを確認します。この Next.js サンプルアプリケーションの場合、[ビルド出力ディレクトリ] は `.next` に設定されます。

2. サービスロールを追加する手順は、新しいロールを作成するか、既存のロールを使用するかによって異なります。
   - 新規ロールを作成するには:
     - [新しいサービスロールの作成と使用] を選択します。
   - 既存のサービスロールを使用するには:
     a. [既存のロールを使用する] を選択します。
     b. サービスロールリストで、使用するロールを選択します。
3. [次へ] を選択します。

## ステップ 3: アプリケーションをデプロイする

このステップでは、 AWS グローバルコンテンツ配信ネットワーク (CDN) にアプリケーションをデプロイします。

アプリを保存してデプロイするには

1. [レビュー] ページで、リポジトリの詳細とアプリの設定が適切であることを確認します。
2. [保存してデプロイ] を選択します。フロントエンドのビルドには通常 1〜2 分かかりますが、アプリのサイズによって変わります。
3. デプロイが完了したら、`amplifyapp.com` デフォルトドメインへのリンクを使用してアプリを表示できます。

> **Note**
>
> Amplify のアプリケーションのセキュリティを強化するために、amplifyapp.com ドメインはパブリックサフィックスリスト (PSL) に登録されています。セキュリティを強化するために、Amplify アプリケーションのデフォルトドメイン名に機密性の高いCookieを設定する必要がある場合は、__Host-プレフィックスの付いたCookieを使用することをお勧めします。このプラクティスは、クロスサイトリクエストフォージェリ (CSRF) 攻撃からドメインを防

ぐ際に役立ちます。詳細については、Mozilla 開発者ネットワークの「Set-Cookie」ページを参照してください。

## ステップ 4: (省略可) リソースをクリーンアップする

チュートリアルでデプロイしたアプリが不要になった場合は、削除できます。このステップにより、使用していないリソースに対して課金されることがなくなります。

アプリを削除するには

1. ナビゲーションペインの [アプリの設定] メニューから、[全般設定] を選択します。
2. [全般設定] ページで、[アプリの削除] を選択します。
3. 確認ウィンドウで、**delete** と入力します。その後、[アプリの削除] を選択します。

## アプリに機能を追加する

これで、Amplify にアプリがデプロイされたので、ホストされたアプリで利用可能な機能の一部を以下で確認できます。

環境変数

多くの場合、アプリケーションは実行時に構成情報を必要とします。この構成には、データベース接続の詳細、API キー、パラメータなどがあります。環境変数は、ビルド時にこれらの構成を公開する方法として使用できます。詳細については、「環境変数」を参照してください。

カスタムドメイン

このチュートリアルでは、Amplify は `https://branch-name.d1m7bkiki6tdw1.amplifyapp.com` などの URL を使用して、デフォルトの `amplifyapp.com` ドメインでアプリをホストします。アプリをカスタムドメインに接続すると、ユーザーは、アプリがカスタム URL (`https://www.example.com` など) でホストされていることを理解できます。詳細については、「カスタムドメイン名の設定」を参照してください。

プルリクエストのプレビュー

Web プルリクエストプレビューは、コードを本番ブランチや統合ブランチにマージする前に、プルリクエスト (PR) からの変更をプレビューする方法をチームに提供します。詳細については、「プルリクエストのウェブプレビュー」を参照してください。

複数の環境を管理する

Amplify が機能ブランチと GitFlow ワークフローと連携して複数のデプロイをサポートする方法については、「機能ブランチのデプロイとチームワークフロー」を参照してください。

# Nuxt.js アプリを Amplify ホスティングにデプロイする

以下の手順に従って、Nuxt.js アプリケーションを Amplify ホスティングにデプロイします。Nuxt は Nitro サーバーを使用してプリセットアダプターを実装しています。これにより、追加の設定なしで Nuxt プロジェクトをデプロイできます。

Nuxt アプリを Amplify ホスティングにデプロイするには

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
2. [すべてのアプリ] ページで、[アプリの新規作成] を選択します。
3. [Amplify で構築を開始] ページで、自分の Git リポジトリプロバイダーを選択し、[次へ] を選択します。
4. [リポジトリブランチを追加] ページで、次の操作を行います。
   a. 接続するリポジトリの名前を選択します。
   b. 接続するリポジトリブランチの名前を選択します。
   c. [次へ] を選択します。
5. Amplify がアプリログを Amazon CloudWatch Logs に送信できるようにするには、コンソールで明示的に有効にする必要があります。[詳細設定] セクションを開いて、[サーバーサイドレンダリング (SSR) のデプロイ] セクションで [SSR アプリのログを有効にする] を選択します。
6. [次へ] を選択します。
7. [レビュー]ページで、[保存してデプロイ] を選択します。

# Astro.js アプリを Amplify ホスティングにデプロイする

以下の手順に従って、Astro.js アプリケーションを Amplify Hosting にデプロイします。既存のアプリケーションを使用することもできるし、Astro が提供する公式例のいずれかを使用してスターターアプリケーションを作成できます。スターターアプリケーションを作成するには、「Astro ドキュメント」の「テーマまたはスターターテンプレートを使用する」を参照してください。

SSR を使用して Astro サイトを Amplify ホスティングにデプロイするには、お使いのアプリケーションにアダプターを追加する必要があります。Astro フレームワーク用の Amplify 所有アダプターは維持されません。このチュートリアルでは、コミュニティのメンバーによって作成された `astro-aws-amplify` アダプターを使用します。このアダプターは、GitHub ウェブサイトの github.com/alexnguyennz/astro-aws-amplify で入手できます。このアダプターはメンテナンス AWS されません。

Astro アプリを Amplify ホスティングにデプロイするには

1. ローカルコンピュータで、デプロイする Astro アプリケーションに移動します。
2. このアダプターをインストールするには、ターミナルウィンドウを開いて以下のコマンドを実行します。この例では、github.com/alexnguyennz/astro-aws-amplify で利用可能なコミュニティアダプターを使用します。*`astro-aws-amplify`* は、使用しているアダプターの名前に置き換えることができます。

   ```
   npm install astro-aws-amplify
   ```

3. Astro アプリのプロジェクトフォルダで、`astro.config.mjs` ファイルを開きます。ファイルを更新してアダプターを追加します。ファイルは次のようになっているはずです。

   ```
   import { defineConfig } from 'astro/config';
   import mdx from '@astrojs/mdx';
   import awsAmplify from 'astro-aws-amplify';

   import sitemap from '@astrojs/sitemap';

   // https://astro.build/config
   export default defineConfig({
     site: 'https://example.com',
     integrations: [mdx(), sitemap()],
     adapter: awsAmplify(),
     output: 'server',
   });
   ```

4. 変更をコミットし、プロジェクトを Git リポジトリにプッシュします。

   これで、Astro アプリを Amplify にデプロイする準備が整いました。

5. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
6. [すべてのアプリ] ページで、[アプリの新規作成] を選択します。

7. [Amplify で構築を開始] ページで、自分の Git リポジトリプロバイダーを選択し、[次へ] を選択します。
8. [リポジトリブランチを追加] ページで、次の操作を行います。
   a. 接続するリポジトリの名前を選択します。
   b. 接続するリポジトリブランチの名前を選択します。
   c. [次へ] を選択します。
9. [アプリ設定] ページで、[ビルド設定] セクションを見つけます。[ビルド出力ディレクトリ] には、**.amplify-hosting** を入力します。
10. また、ビルド仕様でアプリのフロントエンドのビルドコマンドを更新する必要があります。ビルド仕様を開くには、[YML ファイルの編集] を選択します。
11. amplify.yml ファイルで、フロントエンドのビルドコマンドセクションを検索します。**mv node_modules ./.amplify-hosting/compute/default** と入力します。

    ビルド設定ファイルは以下のようになっている必要があります。

```
version: 1
frontend:
    phases:
        preBuild:
            commands:
                - 'npm ci --cache .npm --prefer-offline'
        build:
            commands:
                - 'npm run build'
                - 'mv node_modules ./.amplify-hosting/compute/default'
    artifacts:
        baseDirectory: .amplify-hosting
        files:
            - '**/*'
    cache:
        paths:
            - '.npm/**/*'
```

12. [保存] を選択します。
13. Amplify がアプリログを Amazon CloudWatch Logs に送信できるようにするには、コンソールで明示的に有効にする必要があります。[詳細設定] セクションを開いて、[サーバーサイドレンダリング (SSR) のデプロイ] セクションで [SSR アプリのログを有効にする] を選択します。

14. [次へ] を選択します。
15. [レビュー]ページで、[保存してデプロイ] を選択します。

# SvelteKit アプリを Amplify ホスティングにデプロイする

以下の手順に従って、SvelteKit アプリケーションを Amplify ホスティングにデプロイします。独自のアプリケーションを使用することも、スターターアプリケーションを作成することもできます。詳細については、「SvelteKit ドキュメント」の「プロジェクトの作成」を参照してください。

SSR を使用して SvelteKit アプリを Amplify ホスティングにデプロイするには、プロジェクトにアダプターを追加する必要があります。SvelteKit フレームワークの Amplify 所有アダプターは維持されません。この例では、コミュニティのメンバーが作成した `amplify-adapter` を使用しています。アダプターはGitHub ウェブサイトの github.com/gzimbron/amplify-adapter で入手できます。このアダプターはメンテナンス AWS されません。

SvelteKit アプリを Amplify ホスティングにデプロイするには

1. ローカルコンピュータで、デプロイする SvelteKit アプリケーションに移動します。
2. このアダプターをインストールするには、ターミナルウィンドウを開いて以下のコマンドを実行します。この例では、github.com/gzimbron/amplify-adapter で利用可能なコミュニティアダプターを使用します。別のコミュニティアダプターを使用している場合は、*amplify-adapter* をお使いのアダプターの名前に置き換えます。

```
npm install amplify-adapter
```

3. SvelteKit アプリのプロジェクトフォルダで、`svelte.config.js` ファイルを開きます。ファイルを編集して `amplify-adapter` を使用するか、*[amplify-adapter]* をお使いのアダプター名に置き換えます。ファイルは次のようになっているはずです。

```
import adapter from 'amplify-adapter';
import { vitePreprocess } from '@sveltejs/vite-plugin-svelte';


/** @type {import('@sveltejs/kit').Config} */
const config = {
        // Consult https://kit.svelte.dev/docs/integrations#preprocessors
        // for more information about preprocessors
        preprocess: vitePreprocess(),
```

```
        kit: {
                // adapter-auto only supports some environments, see https://
kit.svelte.dev/docs/adapter-auto for a list.
                // If your environment is not supported, or you settled on a
 specific environment, switch out the adapter.
                // See https://kit.svelte.dev/docs/adapters for more information
 about adapters.
                adapter: adapter()
        }
};

export default config;
```

4. 変更をコミットし、アプリケーションを Git リポジトリにプッシュします。
5. これで、SvelteKit アプリを Amplify にデプロイする準備が整いました。

   にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
6. [すべてのアプリ] ページで、[アプリの新規作成] を選択します。
7. [Amplify で構築を開始] ページで、自分の Git リポジトリプロバイダーを選択し、[次へ] を選択します。
8. [リポジトリブランチを追加] ページで、次の操作を行います。

   a. 接続するリポジトリの名前を選択します。

   b. 接続するリポジトリブランチの名前を選択します。

   c. [次へ] を選択します。
9. [アプリ設定] ページで、[ビルド設定] セクションを見つけます。[ビルド出力ディレクトリ] には、**build** を入力します。
10. また、ビルド仕様でアプリのフロントエンドのビルドコマンドを更新する必要があります。ビルド仕様を開くには、[YML ファイルの編集] を選択します。
11. amplify.yml ファイルで、フロントエンドのビルドコマンドセクションを検索します。**- cd build/compute/default/** と **- npm i --production** を入力します。

    ビルド設定ファイルは以下のようになっている必要があります。

```
version: 1
frontend:
    phases:
        preBuild:
            commands:
```

```
                - 'npm ci --cache .npm --prefer-offline'
        build:
            commands:
                - 'npm run build'
                - 'cd build/compute/default/'
                - 'npm i --production'

    artifacts:
        baseDirectory: build
        files:
            - '**/*'
    cache:
        paths:
            - '.npm/**/*'
```

12. [保存] を選択します。
13. Amplify がアプリログを Amazon CloudWatch Logs に送信できるようにするには、コンソールで明示的に有効にする必要があります。[詳細設定] セクションを開いて、[サーバーサイドレンダリング (SSR) のデプロイ] セクションで [SSR アプリのログを有効にする] を選択します。
14. [次へ] を選択します。
15. [レビュー]ページで、[保存してデプロイ] を選択します。

# Amplify ホスティングでサーバーサイドレンダリングされたアプリのデプロイ

を使用して AWS Amplify 、サーバー側のレンダリング (SSR) を使用するウェブアプリケーションをデプロイおよびホストできます。Amplify ホスティングは Next.js フレームワークを使用して、作成されたアプリケーションを自動的に検出するため、 AWS マネジメントコンソールで手動設定を行う必要はありません。

Amplify は、アプリケーションのビルド出力を Amplify ホスティングが想定するディレクトリ構造に変換するオープンソースのビルドアダプターを備えた Javascript ベースの SSR フレームワークもサポートしています。例えば、使用可能なアダプターをインストールすることで、Nuxt、Astro、SvelteKit フレームワークで作成されたアプリをデプロイできます。

上級ユーザーは、デプロイ仕様を使用してビルドアダプターを作成したり、ビルド後のスクリプトを設定したりできます。

最小限の設定で、次のフレームワークを Amplify ホスティングにデプロイできます。

Next.js

- Amplify は、アダプターを必要とせずに Next.js 15 アプリケーションをサポートします。開始するには、「Next.js 向けの Amplify サポート」を参照してください。

Nuxt.js

- Amplify は、プリセットアダプターを使用して Nuxt.js アプリケーションのデプロイをサポートします。開始するには、「Next.js SSR 用の Amplify サポート」を参照してください。

Astro.js

- Amplify は、コミュニティアダプターを使用して Astro.js アプリケーションのデプロイをサポートします。開始するには、「Astro.js 用の Amplify サポート」を参照してください。

SvelteKit

- Amplify は、コミュニティアダプターを使用して SvelteKit アプリケーションのデプロイをサポートします。開始するには、「SvelteKit 用の Amplify サポート」を参照してください。

オープンソースのアダプターを開く

- オープンソースアダプターを使用する – 前述のリストにないアダプターの使用方法については、「SSR フレームワークのオープンソースアダプターの使用」を参照してください。
- フレームワークアダプターを構築する – フレームワークが提供する機能を統合したいフレームワーク作成者は、Amplify ホスティングのデプロイ仕様を使用して、Amplify が想定する構造に

準拠するようにビルド出力を設定できます。詳細については、「Amplify ホスティングのデプロイの仕様を用いたビルド出力の設定」を参照してください。

- ビルド後のスクリプトを構成する – Amplify ホスティングのデプロイ仕様を使用して、特定のシナリオの必要に応じてビルド出力を操作できます。詳細については、「Amplify ホスティングのデプロイの仕様を用いたビルド出力の設定」を参照してください。例については、デプロイマニフェストを使用した Express サーバーのデプロイを参照してください。

トピック



# Next.js 向けの Amplify サポート

Amplify は、Next.js を使用して作成されたサーバーサイドレンダリング (SSR) ウェブアプリのデプロイとホスティングをサポートします。Next.js は、JavaScript を使って SPA を開発するための React フレームワークです。画像の最適化やミドルウェアなどの機能を備えた Next.js 15 までの Next.js バージョンでビルドされたアプリをデプロイできます。

開発者は Next.js を使用して、静的サイト生成 (SSG) と SSR を 1 つのプロジェクトに組み合わせることができます。SSG ページはビルド時に、SSR ページはリクエスト時に事前にレンダリングされます。

事前レンダリングを行うと、パフォーマンスと検索エンジンの最適化が向上します。Next.js はサーバー上のすべてのページを事前にレンダリングするので、各ページの HTML コンテンツはクライアントのブラウザーに到達した時点で準備完了です。また、このコンテンツの読み込みも速くなります。読み込み時間が短くなると、エンドユーザーのウェブサイトの体験が向上し、サイトの SEO ランキングにプラスの影響を与えます。また、事前レンダリングを行うと、検索エンジンのボットがウェブサイトの HTML コンテンツを簡単に見つけてクローリングできるようになり、SEO も向上します。

Next.js には、最初のバイトまでの時間 (TTFB) や最初のコンテンツの描画 (FCP) など、さまざまなパフォーマンス指標を測定するための分析サポートが組み込まれています。Next.js の詳細については、Next.js のウェブサイトの「ご利用開始にあたって」を参照してください。

## Next.js 機能のサポート

Amplify ホスティングコンピューティングは、Next.js バージョン 12 から 15 で構築されたアプリケーションのサーバーサイドレンダリング (SSR) を完全に管理します。

2022 年 11 月にAmplify ホスティングコンピューティングがリリースされる前に Next.js アプリを Amplify にデプロイした場合、アプリは Amplify の以前の SSR プロバイダーであるClassic (Next.js 11 のみ) を使用しています。Amplify ホスティングコンピューティングは、Next.js バージョン 11 以前を使用して作成されたアプリをサポートしていません。Next.js 11 アプリを Amplify ホスティングコンピューティングマネージド SSR プロバイダーに移行することを強くお勧めします。

以下のリストでは、Amplify ホスティングコンピューティング SSR プロバイダーがサポートする特定の機能について説明しています。

サポートされている機能

- サーバーサイドレンダリングのページ (SSR)
- 静的ページ
- API ルート
- ダイナミックルート
- 全ルートをキャッチ
- SSG (静的生成)
- インクリメンタル・スタティック・リジェネレーション (ISR)
- 国際化 (i18n) サブパスルーティング
- 国際化 (i18n) ドメインルーティング
- 国際化 (i18n) 自動ロケール検出
- ミドルウェア
- 環境変数
- 画像の最適化
- Next.js 13 のアプリケーションディレクトリ

サポートされていない 機能

- エッジ API ルート (エッジミドルウェアはサポートされていません)
- オンデマンドインクリメンタル・スタティック・リジェネレーション (ISR)
- Next.js ストリーミング
- 静的アセットと最適化されたイメージでのミドルウェアの実行
- `unstable_after` による応答後にコードを実行する (Next.js 15 でリリースされた実験機能)

### Next.js の画像

画像の最大出力サイズは 4.3 MB を超えることはできません。より大きな画像ファイルをどこかに保存し、Next.js Image コンポーネントを使用してサイズを変更して、Webp または AVIF 形式に最適化してから、小さいサイズとして提供することができます。

Next.js のドキュメントでは、Sharp 画像処理モジュールをインストールして、画像の最適化を本番環境で正しく動作させることを推奨していることに留意してください。ただし、Amplify のデプロイにはこれは必須ではありません。Amplify はお客様に代わって Sharp 画像処理を自動的にデプロイします。

## Amplify への Next.js SSR アプリケーションのデプロイ

デフォルトでは、Amplify は Next.js バージョン 12 から 15 をサポートする Amplify ホスティングのコンピューティングサービスを使用して新しい SSR アプリケーションをデプロイします。Amplify ホスティングコンピューティングは、SSR アプリケーションのデプロイに必要なリソースを完全に管理します。2022 年 11 月 17 日より以前にデプロイした Amplify アカウントの SSR アプリは、クラシック (Next.js 11 のみ) の SSR プロバイダーを使用しています。

クラシック (Next.js 11 のみ) SSR を使用するアプリをAmplify ホスティングコンピューティングSSR プロバイダーに移行することを強く推奨します。Amplify は自動移行を行いません。更新を完了するには、アプリを手動で移行してから新しいビルドを開始する必要があります。手順については、「Next.js 11 SSR アプリをAmplify ホスティングコンピューティングに移行する」を参照してください。

以下の手順で新しい Next.js SSR アプリをデプロイします。

Amplify ホスティングコンピューティングSSR プロバイダーを使用して SSR アプリを Amplify にデプロイするには

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。

2. [すべてのアプリ] ページで、[アプリの新規作成] を選択します。
3. [Amplify で構築を開始] ページで、自分の Git リポジトリプロバイダーを選択し、[次へ] を選択します。
4. [リポジトリブランチを追加] ページで、次の操作を行います。
   a. 「最近更新されたリポジトリ」リストで、接続するリポジトリの名前を選択します。
   b. ブランチリストで、接続するリポジトリブランチの名前を選択します。
   c. [次へ]をクリックします。
5. アプリには、、お客様に代わって他のサービスを呼び出すときに Amplify が引き受ける IAM サービス ロールが必要です。Amplify ホスティングコンピューティングにサービスロールを自動的に作成させることも、作成したロールを指定することもできます。
   - Amplify が自動的にロールを作成してアプリにアタッチできるようにするには:
     - [新しいサービスロールの作成と使用] を選択します。
   - 以前に作成したサービスロールをアタッチするには:
     a. [既存のサービスロールを使用する] を選択します。
     b. リストから使用するロールを選択します。
6. [次へ] を選択します。
7. [レビュー]ページで、[保存してデプロイ] を選択します。

## パッケージ.json ファイルの設定

Next.js アプリをデプロイすると、Amplify は `package.json` ファイル内のアプリのビルドスクリプトを調べて、アプリの種類を判断します。

Next.js アプリのビルドスクリプトの例を次に示します。ビルドスクリプトは`"next build"`、アプリがSSGページとSSRページの両方をサポートしていることを示しています。このビルドスクリプトは、Next.js 14 以降の SSG 専用アプリケーションにも使用されます。

```
"scripts": {
  "dev": "next dev",
  "build": "next build",
  "start": "next start"
},
```

Next.js 13 以前の SSG アプリのビルドスクリプトの例を次に示します。ビルドスクリプトは"next build && next export"、アプリが SSG ページのみをサポートしていることを示しています。

```
"scripts": {
  "dev": "next dev",
  "build": "next build && next export",
  "start": "next start"
},
```

## Next.js SSR アプリケーション用の Amplify のビルド設定

アプリの package.json ファイルを検査すると、Amplify はアプリのビルド設定をチェックします。ビルド設定は、Amplify コンソールまたはリポジトリのルートにあるamplify.ymlファイルに保存できます。詳細については、「Amplify アプリケーションのビルド設定の構成」を参照してください。

Amplify が Next.js SSR アプリをデプロイしていることを検出し、amplify.ymlファイルが存在しない場合、アプリのビルドスペックが生成され、baseDirectoryが.nextに設定されます。ファイルが存在するアプリをデプロイする場合、amplify.ymlファイル内のビルド設定はコンソールのビルド設定よりも優先されます。そのため、ファイル内のbaseDirectoryは手動で.nextに設定する必要があります。

以下は、baseDirectoryが.nextに設定されているアプリのビルド設定の例です。これは、ビルドアーティファクトが SSG ページと SSR ページをサポートする Next.js アプリ用であることを示しています。

```
version: 1
frontend:
  phases:
    preBuild:
      commands:
        - npm ci
    build:
      commands:
        - npm run build
  artifacts:
    baseDirectory: .next
    files:
      - '**/*'
  cache:
```

```
    paths:
      - node_modules/**/*
```

## Next.js 13 以前の SSG アプリケーション用の Amplify のビルド設定

Amplify が Next.js 13 以前の SSG アプリをデプロイしていることを検出すると、そのアプリのビルド使用が生成され、baseDirectory が out に設定されます。ファイルが存在するアプリをデプロイする場合は、amplify.ymlファイル内でbaseDirectoryを手動でoutに設定する必要があります。out ディレクトリは、エクスポートされた静的アセットを保存するために Next.js が作成するデフォルトのフォルダです。アプリのビルド仕様設定を構成するときは、アプリの設定と一致するように baseDirectory フォルダの名前を変更します。

次の例が示すのは、baseDirectory を out に設定することで、ビルドアーティファクトが SSG ページのみをサポートする Next.js 13 以前のアプリケーション用であることを示すアプリのビルド設定です。

```
version: 1
frontend:
  phases:
    preBuild:
      commands:
        - npm ci
    build:
      commands:
        - npm run build
  artifacts:
    baseDirectory: out
    files:
      - '**/*'
  cache:
    paths:
      - node_modules/**/*
```

## Next.js 14 以降の SSG アプリケーション向けの Amplify ビルド設定

Next.js バージョン 14 では、静的エクスポートを有効にするために、next export コマンドは非推奨になり、next.config.js ファイルで output: 'export' に置き換えられました。Next.js 14 SSG 専用アプリケーションをコンソールにデプロイする場合、Amplify はアプリの buildspec を生成し、baseDirectory を .next に設定します。ファイルが存在するアプリをデプロイする場合は、amplify.ymlファイル内でbaseDirectoryを手動で.nextに設定する必要があります。こ

れは、SSG ページと SSR ページの両方をサポートする Next.js WEB_COMPUTE アプリケーションで Amplify が使用するのと同じ baseDirectory 設定です。

以下は、baseDirectory が .next に設定された Next.js 14 SSG 専用アプリケーションのビルド設定の例です。

```
version: 1
frontend:
  phases:
    preBuild:
      commands:
        - npm ci
    build:
      commands:
        - npm run build
  artifacts:
    baseDirectory: .next
    files:
      - '**/*'
  cache:
    paths:
      - node_modules/**/*
```

## Next.js 11 SSR アプリをAmplify ホスティングコンピューティングに移行する

新しい Next.js アプリをデプロイすると、デフォルトで Amplify はサポートされている最新バージョンの Next.js を使用します。現在、Amplify ホスティングコンピューティング SSR プロバイダーは Next.js バージョン 15 をサポートしています。

Amplify コンソールは、Next.js バージョン 12 から 15 を完全にサポートする Amplify ホスティングコンピューティングサービスの 2022 年 11 月のリリース前にデプロイされたアカウント内のアプリケーションを検出します。コンソールには、Amplify の以前の SSR プロバイダーである Classic (Next.js 11 のみ) を使用してデプロイされた、ブランチのあるアプリを識別する情報バナーが表示されます。Amplify ホスティングコンピューティングSSR プロバイダーにアプリを移行することを強く推奨します。

ホストされている Next.js 11 アプリケーションを Next.js 12 以降に更新すると、デプロイがトリガーされるときに "target" property is no longer supported エラーが生じることがあります。この場合、Amplify ホスティングコンピューティングに移行する必要があります。

アプリとそのすべての運用ブランチを同時に手動で移行する必要があります。アプリにはクラシック (Next.js 11 のみ) ブランチと Next.js 12 以降のブランチの両方を入れることはできません。

以下の手順を使用して、アプリをAmplify ホスティングコンピューティングSSR プロバイダーに移行します。

アプリをAmplify ホスティングコンピューティングSSR プロバイダーに移行するには

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
2. 移行する Next.js アプリを選択します。

> **Note**
>
> Amplify コンソールでアプリを移行する前に、まず Next.js バージョン 12 以降を使用するようにアプリの package.json ファイルを更新する必要があります。

3. ナビゲーションペインで [アプリの設定] の [一般] を選択します。
4. アプリに Classic (Next.js 11のみ) SSRプロバイダーを使用してデプロイされたブランチがある場合は、アプリのホームページにバナーが表示されます。バナーで [移行] を選択します。
5. 移行確認ウィンドウで3つのステートメントを選択し、[移行] を選択します。
6. Amplify はアプリをビルドして再デプロイし、移行を完了します。

## SSR 移行を元に戻す

Next.js アプリをデプロイすると、Amplify ホスティングはアプリの設定を検出し、アプリの内部プラットフォーム値を設定します。有効なプラットフォーム値は3 つあります。SSG アプリはプラットフォーム値に設定されます`WEB`。Next.js バージョン 11 を使用する SSR アプリはプラットフォーム値に設定されます`WEB_DYNAMIC`。Next.js 12 以降の SSR アプリはプラットフォーム値に設定されます`WEB_COMPUTE`。

前のセクションの手順を使用してアプリを移行すると、Amplify はアプリのプラットフォーム値を`WEB_DYNAMIC`から`WEB_COMPUTE`に変更します。Amplify ホスティングコンピューティングへの移行が完了したら、コンソールで移行を元に戻すことはできません。移行を元に戻すには、 AWS Command Line Interface を使用してアプリのプラットフォームを`WEB_DYNAMIC`に戻す必要があります。ターミナルウィンドウを開いて次のコマンドを入力し、アプリID とリージョンを独自の情報で更新します。

```
aws amplify update-app --app-id abcd1234 --platform WEB_DYNAMIC --region us-west-2
```

# 静的 Next.js アプリに SSR 機能を追加します。

Amplify でデプロイされた既存の静的 (SSG) Next.js アプリに SSR 機能を追加できます。SSG アプリを SSR に変換するプロセスを開始する前に、Next.js バージョン 12 以降を使用するようにアプリを更新し、SSR 機能を追加してください。次に、以下のステップを実行する必要があります。

1. を使用して AWS Command Line Interface 、アプリケーションのプラットフォームタイプを変更します。
2. アプリにサービスロールを追加します。
3. アプリのビルド設定で出力ディレクトリを更新します。
4. アプリが SSR を使用していることを示すようにアプリの`package.json`ファイルを更新します。

## プラットフォームを更新する

プラットフォームタイプには 3 つの値があります。SSG アプリはプラットフォームタイプ`WEB`に設定されます。Next.js バージョン 11 を使用する SSR アプリはプラットフォームタイプ`WEB_DYNAMIC`に設定されます。Amplify ホスティングコンピューティングが管理する SSR を使用して Next.js 12 以降にデプロイされたアプリの場合、プラットフォームタイプは `WEB_COMPUTE` に設定されます。

アプリを SSG アプリとしてデプロイしたとき、Amplify はプラットフォームタイプを`WEB`に設定しました。を使用して AWS CLI 、アプリケーションのプラットフォームを に変更します`WEB_COMPUTE`。ターミナルウィンドウを開いて次のコマンドを入力し、赤色のテキストを固有のアプリ ID とリージョンで更新します。

```
aws amplify update-app --app-id abcd1234 --platform WEB_COMPUTE --region us-west-2
```

## サービスロールの追加

サービスロールは、Amplify がユーザーに代わって他のサービスを呼び出すときに引き受ける AWS Identity and Access Management (IAM) ロールです。以下の手順に従って、Amplifyで すでにデプロイされている SSG アプリにサービスロールを追加します。

サービスロールを追加するには

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
2. Amplify アカウントでまだサービスロールを作成していない場合は、「サービスロールの追加」を参照してこの前提条件の手順を完了してください。
3. サービスロールを追加する静的 Next.js アプリを選択します。
4. ナビゲーションペインで [アプリの設定] の [一般] を選択します。
5. [アプリの詳細]ページで、[編集] を選択します。
6. サービスロール で、既存のサービスロールの名前、またはステップ 2 で作成したサービスロールの名前を選択します。
7. [保存] を選択します。

## ビルド設定の更新

SSR 機能を使用してアプリを再デプロイする前に、アプリのビルド設定を更新して出力ディレクトリを`.next`に設定する必要があります。ビルド設定は、Amplify コンソールまたは`amplify.yml`リポジトリに保存されているファイルで編集できます。詳細については、「Amplify アプリケーションのビルド設定の構成」を参照してください。

以下は、`baseDirectory`が`.next`に設定されているアプリのビルド設定の例です。

```
version: 1
frontend:
  phases:
    preBuild:
      commands:
        - npm ci
    build:
      commands:
        - npm run build
  artifacts:
    baseDirectory: .next
    files:
      - '**/*'
  cache:
    paths:
      - node_modules/**/*
```

### package.json ファイルの更新

サービスロールを追加してビルド設定を更新したら、アプリのpackage.jsonファイルを更新します。次の例のように、Next.js アプリが SSG ページと SSR ページの両方をサポートすることを示すようにビルドスクリプト"next build"を設定します。

```
"scripts": {
  "dev": "next dev",
  "build": "next build",
  "start": "next start"
},
```

Amplify はリポジトリ内のpackage.jsonファイルへの変更を検出し、SSR 機能を使用してアプリを再デプロイします。

## 環境変数をサーバーサイドランタイムからアクセスできるようにします。

Amplify ホスティングは、Amplify コンソールのプロジェクト構成で環境変数を設定することにより、アプリケーションのビルドに環境変数を追加することをサポートしています。

ただし、Next.js サーバーコンポーネントはデフォルトではこれらの環境変数にアクセスできません。この動作は、ビルドフェーズでアプリケーションが使用する環境変数に保存されているシークレットを保護するためのものです。

特定の環境変数を Next.js にアクセスできるようにするには、Amplify ビルド仕様ファイルを変更すると Next.js が認識する環境ファイルに設定できます。これにより、Amplify はアプリケーションをビルドする前にこれらの環境変数をロードできます。

> **Important**
>
> デプロイアーティファクトにアクセスできるユーザーは読み取ることができるため、認証情報、シークレット、または機密情報を環境変数に保存しないことを強くお勧めします。SSR コンピューティング関数に AWS リソースへのアクセスを許可するには、IAM ロールを使用することをお勧めします。

以下のビルド仕様例は、ビルドコマンドセクションに環境変数を追加する方法を示しています。

```
version: 1
frontend:
```

```
  phases:
    preBuild:
      commands:
        - npm ci
    build:
      commands:
        - env | grep -e API_BASE_URL >> .env.production
        - env | grep -e NEXT_PUBLIC_ >> .env.production
        - npm run build
  artifacts:
    baseDirectory: .next
    files:
      - '**/*'
  cache:
    paths:
      - node_modules/**/*
      - .next/cache/**/*
```

この例では、ビルドコマンドセクションには、アプリケーションのビルドを実行する前に環境変数を.env.productionファイルに書き込む 2 つのコマンドが含まれています。Amplify ホスティングを使用すると、アプリケーションがトラフィックを受信したときに、アプリケーションがこれらの変数にアクセスできます。

前の例のビルドコマンドセクションの次の行は、ビルド環境から特定の変数を取得して.env.productionファイルに追加する方法を示しています。

```
- env | grep -e API_BASE_URL -e APP_ENV >> .env.production
```

ビルド環境に変数が存在する場合、.env.productionファイルには以下の環境変数が含まれます。

```
API_BASE_URL=localhost
APP_ENV=dev
```

前の例のビルドコマンドセクションの次の行は、特定のプレフィックスを付けた環境変数を.env.productionファイルに追加する方法を示しています。この例では、プレフィックスNEXT_PUBLIC_の付いた変数がすべて追加されます。

```
- env | grep -e NEXT_PUBLIC_ >> .env.production
```

プレフィックスNEXT_PUBLIC_の付いた変数がビルド環境に複数存在する場合、.env.productionファイルは次のようになります。

```
NEXT_PUBLIC_ANALYTICS_ID=abcdefghijk
NEXT_PUBLIC_GRAPHQL_ENDPOINT=uowelalsmlsadf
NEXT_PUBLIC_FEATURE_FLAG=true
```

## モノレポの SSR 環境変数

SSR アプリをモノレポにデプロイしていて、特定の環境変数が Next.js にアクセスできるようにする場合は、.env.production ファイルのプレフィックスにアプリケーションルートを付ける必要があります。次の Nx モノレポ内の Next.js アプリのビルド仕様例は、ビルドコマンドセクションに環境変数を追加する方法を示しています。

```
version: 1
applications:
  - frontend:
      phases:
        preBuild:
          commands:
            - npm ci
        build:
          commands:
            - env | grep -e API_BASE_URL -e APP_ENV >> apps/app/.env.production
            - env | grep -e NEXT_PUBLIC_ >> apps/app/.env.production
            - npx nx build app
      artifacts:
        baseDirectory: dist/apps/app/.next
        files:
          - '**/*'
      cache:
        paths:
          - node_modules/**/*
      buildPath: /
    appRoot: apps/app
```

前述の例のビルドコマンドセクションの次の行は、ビルド環境から特定の変数を取得して、アプリケーションルート apps/app を持つモノレポのアプリの .env.production ファイルに追加する方法を示しています。

```
- env | grep -e API_BASE_URL -e APP_ENV >> apps/app/.env.production
- env | grep -e NEXT_PUBLIC_ >> apps/app/.env.production
```

## Next.js アプリをモノリポジトリにデプロイする

Amplify は、一般的なモノレポのアプリだけでなく、npm ワークスペース、pnpm ワークスペース、Yarn ワークスペース、Nx、および Turborepo を使用して作成されたモノレポのアプリもサポートします。アプリをデプロイすると、Amplify は使用しているモノレポジトリのビルドフレームワークを自動的に検出します。Amplify は、npmワークスペース、Yarnワークスペース、またはNxのアプリにビルド設定を自動的に適用します。Turborepo と pnpm アプリには、追加設定が必要です。詳細については、「モノレポビルド設定の構成」を参照してください。

詳細なNxの例については、「AWS Amplify ホスティングの Next.js のアプリケーション間で Nxを使用してコードを共有する」というブログ投稿を参照してください。

## Next.js SSR 用の Amplify サポート

Nuxt は、Vue.js を使用してフルスタックのウェブアプリケーションを作成するためのフレームワークです。

アダプター

Nuxt.js アプリケーションを Amplify にデプロイするには、設定不要のプリセットアダプターを使用します。アダプターの使用に関する詳細については、「Nuxt のドキュメント」を参照してください。

チュートリアル

Nuxt.js アプリケーションを Amplify にデプロイする方法については、「Nuxt.js アプリを Amplify ホスティングにデプロイする」を参照してください。

デモ

動画のデモンストレーションについては、YouTube の「Nuxt Hosting with ZERO Configuration In Minutes (With AWS)」を参照してください。

## Astro.js 用の Amplify サポート

Astro は、コンテンツドリブンのウェブアプリケーションを作成するためのウェブフレームワークです。

アダプター

コミュニティアダプターを使用して、Astro.js アプリケーションを Amplify にデプロイできます。Astro フレームワーク用の Amplify 所有アダプターは維持されません。ただし、アダプターは GitHub ウェブサイトの github.com/alexnguyennz/astro-aws-amplify で入手できます。このアダプターはコミュニティのメンバーによって作成されたため、 AWSによって維持されません。

チュートリアル

Astro アプリを Amplify にデプロイする方法については、「Astro.js アプリを Amplify ホスティングにデプロイする」を参照してください。

デモ

デモ動画は、Amazon Web Services YouTube チャンネルで「How to deploy an Astro Website to AWS」をご覧ください。

# SvelteKit 用の Amplify サポート

SvelteKit は、Svelte でフルスタックのウェブアプリケーションを作成するためのフレームワークです。

アダプター

SvelteKit アプリケーションは、コミュニティアダプターを使用して Amplify にデプロイできます。SvelteKit フレームワークの Amplify 所有アダプターは維持されません。ただし、アダプターは GitHub ウェブサイトの github.com/gzimbron/amplify-adapter で入手できます。このアダプターはコミュニティのメンバーによって作成されたため、 AWSによって維持されません。

チュートリアル

SvelteKit アプリを Amplify にデプロイする方法については、「SvelteKit アプリを Amplify ホスティングにデプロイする」を参照してください。

デモ

デモ動画については、Amazon Web Services YouTube チャンネルで「How to deploy a SvelteKit website (with API) to AWS」を参照してください。

# SSR アプリケーションの Amplify へのデプロイ

この手順に従って、Amplify が想定するビルド出力に適合するデプロイバンドルを使用して、任意のフレームワークで作成されたアプリをデプロイできます。Next.js アプリケーションをデプロイする場合、アダプターは必要ありません。

フレームワークアダプターを使用する SSR アプリケーションをデプロイする場合は、最初にアダプターをインストールして設定する必要があります。手順については、「SSR フレームワークのオープンソースアダプターの使用」を参照してください。

SSR アプリケーションを Amplify ホスティングにデプロイするには

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
2. [すべてのアプリ] ページで、[アプリの新規作成] を選択します。
3. [Amplify で構築を開始] ページで、自分の Git リポジトリプロバイダーを選択し、[次へ] を選択します。
4. 「リポジトリブランチを追加」ページで、以下の操作を行います。

   a. 接続するリポジトリの名前を選択します。

   b. 接続するリポジトリブランチの名前を選択します。

   c. [次へ] を選択します。
5. [アプリの設定] ページで、Amplify は Next.js SSR アプリを自動的に検出します。

   別のフレームワークのアダプターを使用する SSR アプリケーションをデプロイする場合は、Amazon CloudWatch Logs を明示的に有効にする必要があります。[詳細設定] セクションを開いて、[サーバーサイドレンダリング (SSR) のデプロイ] セクションで [SSR アプリのログを有効にする] を選択します。
6. アプリケーションには、Amplify がログを AWS アカウントに配信するために引き受ける IAM サービスロールが必要です。

   サービスロールを追加する手順は、新しいロールを作成するか、既存のロールを使用するかによって異なります。

   - 新規ロールを作成するには:
     - [新しいサービスロールの作成と使用] を選択します。
   - 既存のサービスロールを使用するには:

a. [既存のロールを使用する] を選択します。

b. サービスロールリストで、使用するロールを選択します。

7. [次へ] を選択します。

8. [レビュー]ページで、[保存してデプロイ] を選択します。

# SSR でサポートされている機能

このセクションでは、Amplify の SSR 機能のサポートについて説明します。

Amplify は、アプリの構築に使用された Node.js のバージョンと一致する Node.js バージョンサポートを提供します。

Amplify は、すべての SSR アプリをサポートする組み込みの画像の最適化機能を提供します。デフォルトの画像最適化機能を使用しない場合は、カスタム画像最適化ローダーを実装できます。

トピック

- Next.js アプリケーション向けの Node.js バージョンのサポート
- SSR アプリケーション向けの画像の最適化
- SSR アプリの Amazon CloudWatch Logs
- Amplify Next.js 11 SSR のサポート

## Next.js アプリケーション向けの Node.js バージョンのサポート

Amplify が Next.js コンピューティングアプリケーションを構築してデプロイする際、アプリケーションの構築に使用された Node.js のメジャーバージョンと一致する Node.js ランタイムバージョンが使用されます。

> **Note**
>
> 2025 年 9 月 15 日以降、Amplify ホスティングは Node.js 14、Node.js 16、Node.js 18 ランタイムをサポートしなくなります。サポートされるランタイムは、Node.js 20 や Node.js 22 などです。

Amplify コンソールの [ライブパッケージオーバーライド] 機能で使用する Node.js バージョンを指定できます。ライブパッケージアップデートの設定の詳細については、「ビルドイメージでの特定の

パッケージバージョンと依存関係バージョンの使用」を参照してください。nvm コマンドなどの他のメカニズムを使用して Node.js バージョンを指定することもできます。バージョンを指定しない場合、Amplify はデフォルトで、Amplify ビルドコンテナによって使用されている現在のバージョンを使用します。

## SSR アプリケーション向けの画像の最適化

Amplify ホスティングは、すべての SSR アプリケーションをサポートする組み込みの画像の最適化機能を提供します。Amplify の画像の最適化を使用すると、ファイルサイズを可能な限り最小限に抑えながら、アクセス先のデバイスにとって適切な形式、次元、解像度で質の高い画像を配信できます。

現在、Next.js Image コンポーネントを使用してオンデマンドで画像を最適化することも、カスタム画像ローダーを実装することもできます。Next.js 13 以降を使用している場合、Amplify の画像の最適化機能を使用するためにそれ以上必要なアクションはありません。カスタムローダーを実装する場合は、次の「カスタムイメージローダーの使用」トピックを参照してください。

### カスタム画像ローダーの使用

カスタム画像ローダーを使用する場合、Amplify はアプリケーションの `next.config.js` ファイル内のローダーを検出し、組み込みの画像の最適化機能を利用しません。Next.js がサポートするカスタムローダーの詳細については、Next.js 画像のドキュメントを参照してください。

## SSR アプリの Amazon CloudWatch Logs

Amplify は SSR ランタイムに関する情報を、 AWS アカウントの Amazon CloudWatch Logs に送信します。SSR アプリをデプロイする場合、アプリには、ユーザーの代わりに他のサービスを呼び出す際に Amplify が引き受けるIAMサービスロールが必要です。Amplify ホスティングコンピューティングにサービスロールを自動的に作成させることも、作成したロールを指定することもできます。

Amplify に IAM ロールの作成を許可することを選択した場合、そのロールにはすでに CloudWatch Logs を作成する権限が付与されています。独自の IAM ロールを作成する場合、Amplify が Amazon CloudWatch Logs にアクセスできるようにするには、ポリシーに次のアクセス権限を追加する必要があります。

```
logs:CreateLogStream
logs:CreateLogGroup
logs:DescribeLogGroups
logs:PutLogEvents
```

サービスロールの詳細については、「バックエンドリソースをデプロイするアクセス許可を持つサービスロールの追加」を参照してください。

## Amplify Next.js 11 SSR のサポート

2022 年 11 月 17 日にAmplify ホスティングコンピューティングがリリースされる前に Next.js アプリをAmplify にデプロイした場合、アプリは Amplify の以前の SSR プロバイダーであるClassic (Next.js 11 のみ) を使用しています。このセクションのドキュメントは、Classic (Next.js 11 のみ) SSR プロバイダーを使用してデプロイされたアプリにのみ適用されます。

> **Note**
>
> Next.js 11 アプリをAmplify ホスティングコンピューティングマネージド SSR プロバイダーに移行することを強くお勧めします。詳細については、「Next.js 11 SSR アプリをAmplify ホスティングコンピューティングに移行する」を参照してください。

以下のリストでは、Amplify Classic (Next.js 11 のみ) SSR プロバイダーがサポートする特定の機能について説明しています。

サポートされている機能

- サーバーサイドレンダリングのページ (SSR)
- 静的ページ
- API ルート
- ダイナミックルート
- 全ルートをキャッチ
- SSG (静的生成)
- インクリメンタル・スタティック・リジェネレーション (ISR)
- 国際化 (i18n) サブパスルーティング
- 環境変数

サポートされていない 機能

- イメージの最適化

- オンデマンドインクリメンタル・スタティック・リジェネレーション (ISR)
- 国際化 (i18n) ドメインルーティング
- 国際化 (i18n) 自動ロケール検出
- ミドルウェア
- エッジ ミドルウェア
- エッジ API ルート¶

## Next.js 11 SSR アプリケーションの価格設定

Next.js 11 SSR アプリをデプロイすると、Amplify は AWS アカウントに次のような追加のバックエンドリソースを作成します。

- アプリの静的アセットのリソースを格納するAmazon Simple Storage Service (Amazon S3) バケット。Amazon S3 の料金に関する詳細については、「Amazon S3 の料金」を参照してください。
- アプリを提供する Amazon CloudFront ディストリビューション。CloudFront の料金の詳細については、「Amazon CloudFront 料金表」を参照してください。
- CloudFront が配信するコンテンツをカスタマイズする4つのLambda @Edge 関数。

## AWS Identity and Access Management Next.js 11 SSR アプリケーションの アクセス許可

Amplify では、SSR アプリケーションをデプロイするために AWS Identity and Access Management (IAM) アクセス許可が必要です。SSR アプリケーションの場合、Amplify は Amazon S3 バケット、CloudFront ディストリビューション、Lambda@Edge 関数、Amazon SQS キュー (ISR を使用している場合)、IAM ロールなどのリソースをデプロイします。必要最小限の権限がなければ、SSR アプリをデプロイしようとしたときに `Access Denied` エラーが発生します。Amplify に必要な権限を付与するには、サービスロールを指定する必要があります。

お客様に代わって他のサービスを呼び出すときに Amplify が引き受けるIAM サービスロールを作成するには、バックエンドリソースをデプロイするアクセス許可を持つサービスロールの追加 を参照してください。以下の手順では、`AdministratorAccess-Amplify`管理ポリシーをアタッチするロールを作成する方法を示しています。

`AdministratorAccess-Amplify` マネージドポリシーは、IAM アクションを含む複数の AWS サービスへのアクセスを提供します。 および は、`AdministratorAccess`ポリシーとして強力であ

ると見なす必要があります。このポリシーでは、SSR アプリのデプロイに必要な権限よりも多くの権限が付与されます。

最小特権を認めるというベストプラクティスに従い、サービスロールに付与するアクセス許可を減らすことを推奨します。サービスロールに管理者アクセス権限を付与する代わりに、SSR アプリのデプロイに必要な権限のみを付与する独自のカスタマーマネージド IAM ポリシーを作成できます。カスタマー管理ポリシーを作成する手順については、「IAM ユーザーガイド」の「IAM ポリシーの作成」を参照してください。

独自のポリシーを作成する場合は、SSR アプリのデプロイに必要な最低限の権限を以下に示します。

```
acm:DescribeCertificate
acm:DescribeCertificate
acm:ListCertificates
acm:RequestCertificate
cloudfront:CreateCloudFrontOriginAccessIdentity
cloudfront:CreateDistribution
cloudfront:CreateInvalidation
cloudfront:GetDistribution
cloudfront:GetDistributionConfig
cloudfront:ListCloudFrontOriginAccessIdentities
cloudfront:ListDistributions
cloudfront:ListDistributionsByLambdaFunction
cloudfront:ListDistributionsByWebACLId
cloudfront:ListFieldLevelEncryptionConfigs
cloudfront:ListFieldLevelEncryptionProfiles
cloudfront:ListInvalidations
cloudfront:ListPublicKeys
cloudfront:ListStreamingDistributions
cloudfront:UpdateDistribution
cloudfront:TagResource
cloudfront:UntagResource
cloudfront:ListTagsForResource
iam:AttachRolePolicy
iam:CreateRole
iam:CreateServiceLinkedRole
iam:GetRole
iam:PutRolePolicy
iam:PassRole
lambda:CreateFunction
lambda:EnableReplication
lambda:DeleteFunction
```

```
lambda:GetFunction
lambda:GetFunctionConfiguration
lambda:PublishVersion
lambda:UpdateFunctionCode
lambda:UpdateFunctionConfiguration
lambda:ListTags
lambda:TagResource
lambda:UntagResource
route53:ChangeResourceRecordSets
route53:ListHostedZonesByName
route53:ListResourceRecordSets
s3:CreateBucket
s3:GetAccelerateConfiguration
s3:GetObject
s3:ListBucket
s3:PutAccelerateConfiguration
s3:PutBucketPolicy
s3:PutObject
s3:PutBucketTagging
s3:GetBucketTagging
lambda:ListEventSourceMappings
lambda:CreateEventSourceMapping
iam:UpdateAssumeRolePolicy
iam:DeleteRolePolicy
sqs:CreateQueue           // SQS only needed if using ISR feature
sqs:DeleteQueue
sqs:GetQueueAttributes
sqs:SetQueueAttributes
amplify:GetApp
amplify:GetBranch
amplify:UpdateApp
amplify:UpdateBranch
```

## Next.js 11 SSR デプロイのトラブルシューティング

Amplify で Classic (Next.js 11 のみ) SSR アプリをデプロイする際に予期しない問題が発生した場合は、以下のトラブルシューティングトピックを確認してください。

トピック

- アプリケーションの出力ディレクトリが上書きされる
- SSR サイトをデプロイすると 404 エラーが発生します。
- アプリに CloudFront SSR ディストリビューションの書き換えルールがありません

- アプリケーションが大きすぎてデプロイできません
- ビルドがメモリ不足エラーで失敗します
- アプリケーションに SSR と SSG の両方のブランチがあります。
- アプリが静的ファイルを予約パスのあるフォルダに保存します。
- アプリが CloudFront の制限に達しました
- Lambda@Edge 関数は米国東部 (バージニア北部)リージョンで作成されます。
- Next.js アプリではサポートされていない機能が使用されています。
- Next.js アプリケーションにある画像が読み込まれない
- サポートされていないリージョン

### アプリケーションの出力ディレクトリが上書きされる

Amplify でデプロイされた Next.js アプリの出力ディレクトリは `.next` に設定する必要があります。アプリの出力ディレクトリが上書きされている場合は、`next.config.js` ファイルを確認してください。ビルド出力ディレクトリのデフォルトを `.next` にするには、ファイルから次の行を削除します。

```
distDir: 'build'
```

ビルド設定で出力ディレクトリが `.next` に設定されていることを確認します。アプリのビルド設定を表示する方法については、Amplify アプリケーションのビルド設定の構成を参照してください。

以下は、`baseDirectory` が `.next` に設定されているアプリのビルド設定の例です。

```
version: 1
frontend:
  phases:
    preBuild:
      commands:
        - npm ci
    build:
      commands:
        - npm run build
  artifacts:
    baseDirectory: .next
    files:
      - '**/*'
```

```
  cache:
    paths:
      - node_modules/**/*
```

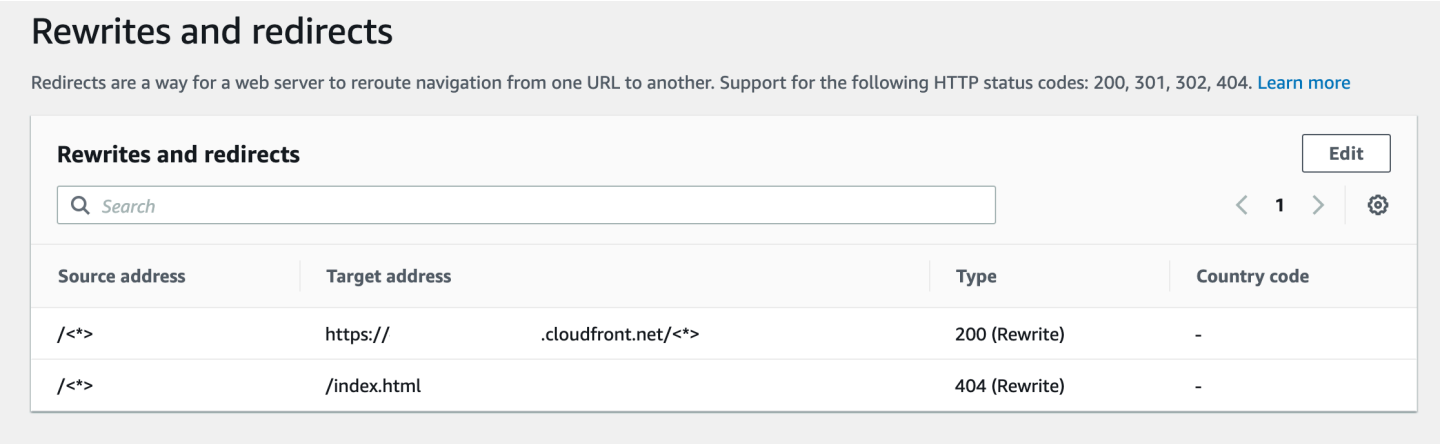
SSR サイトをデプロイすると 404 エラーが発生します。

サイトをデプロイした後に 404 エラーが発生した場合、出力ディレクトリが上書きされたことが問題の原因である可能性があります。`next.config.js`ファイルを確認し、アプリのビルドスペック内のビルド出力ディレクトリが正しいことを確認するには、前のトピックの[アプリケーションの出力ディレクトリが上書きされる]の手順に従ってください。

アプリに CloudFront SSR ディストリビューションの書き換えルールがありません

SSR アプリをデプロイすると、Amplify は CloudFront SSR ディストリビューションの書き換えルールを作成します。ウェブブラウザでアプリにアクセスできない場合は、Amplify コンソールでアプリの CloudFront 書き換えルールが存在することを確認してください。見つからない場合は、手動で追加するか、アプリを再デプロイできます。

Amplify コンソールでアプリの書き換えルールとリダイレクトルールを表示または編集するには、ナビゲーションペインで、[アプリ設定]、[書き換えとリダイレクト] の順に選択します。次のスクリーンショットは、SSR アプリをデプロイするときに Amplify が作成するリライトルールの例を示しています。この例では、CloudFront の書き換えルールが存在することに注意してください。




アプリケーションが大きすぎてデプロイできません

Amplify は、SSR デプロイのサイズを50 MB に制限しています。Next.js SSR アプリを Amplify にデプロイしようとして`RequestEntityTooLargeException`エラーが発生した場合は、アプリが大きすぎるためデプロイできません。この問題を回避するには、キャッシュクリーンアップコードを`next.config.js`ファイルに追加してください。

キャッシュクリーンアップを実行するnext.config.jsファイル内のコードの例を次に示します。

```
module.exports = {
    webpack: (config, { buildId, dev, isServer, defaultLoaders, webpack }) => {
        config.optimization.splitChunks.cacheGroups = { }
        config.optimization.minimize = true;
        return config
      },
}
```

ビルドがメモリ不足エラーで失敗します

Next.js では、ビルドアーティファクトをキャッシュして、以降のビルドのパフォーマンスを向上させることができます。さらに、Amplify の AWS CodeBuild コンテナは、後続のビルドパフォーマンスを向上させるために、ユーザーに代わってこのキャッシュを圧縮して Amazon S3 にアップロードします。これにより、ビルドがメモリ不足エラーで失敗する可能性があります。

ビルドフェーズ中にアプリがメモリ制限を超えないようにするには、次のアクションを実行します。まず、ビルド設定の cache.paths セクションから.next/cache/**/*を削除します。次に、ビルド設定ファイルから環境変数NODE_OPTIONSを削除します。代わりに、Amplify コンソールで環境変数NODE_OPTIONSを設定して、ノードの最大メモリ制限を定義します。Amplify コンソールを使用した環境変数を設定する方法の詳細については、「環境変数の設定」を参照してください。

これらの変更を加えたら、ビルドをやり直してください。成功したら、ビルド設定ファイルの cache.paths セクションに.next/cache/**/*を追加し直してください。

ビルドパフォーマンスを向上させるための Next.js キャッシュ設定の詳細については、Next.js のウェブサイトの「AWS CodeBuild」を参照してください。

アプリケーションに SSR と SSG の両方のブランチがあります。

SSR と SSG の両方のブランチを持つアプリはデプロイできません。SSR ブランチと SSG ブランチの両方をデプロイする必要がある場合は、SSR ブランチのみを使用するアプリと SSG ブランチのみを使用する別のアプリをデプロイする必要があります。

アプリが静的ファイルを予約パスのあるフォルダに保存します。

Next.js は、プロジェクトのルートディレクトリに保存されているpublicという名前のフォルダから静的ファイルを提供できます。Amplify で Next.js アプリをデプロイしてホストする場合、プロジェクトにpublic/staticパスのフォルダーを含めることはできません。Amplify は、アプリを配

布するときに使用するpublic/staticパスを予約します。アプリにこのパスが含まれている場合は、Amplify でデプロイする前にstaticフォルダーの名前を変更する必要があります。

アプリが CloudFront の制限に達しました

CloudFront サービスクォータは、Lambda@Edge 関数がアタッチされた 25 のディストリビューションに AWS アカウントを制限します。このクォータを超える場合は、未使用の CloudFront ディストリビューションをアカウントから削除するか、クォータの増額をリクエストできます。詳細については、「Service Quotas ユーザーガイド」の「クォータ引き上げのリクエスト」を参照してください。

Lambda@Edge 関数は米国東部 (バージニア北部)リージョンで作成されます。

Next.js アプリをデプロイすると、Amplify は CloudFront が配信するコンテンツをカスタマイズする Lambda @Edge 関数を作成します。Lambda @Edge 関数は、アプリがデプロイされているリージョンではなく、米国東部 (バージニア北部) リージョンで作成されます。これは Lambda @Edge の制限です。Lambda @Edge 関数の詳細については、「Amazon CloudFront デベロッパーガイド」の「エッジ関数の制限」を参照してください。

Next.js アプリではサポートされていない機能が使用されています。

Amplify でデプロイされたアプリは、バージョン 11 までの Next.js のメジャーバージョンをサポートします。Amplify でサポートされている、またはサポートされていない Next.js 機能の詳細なリストについては、supported featuresを参照してください。

新しい Next.js アプリをデプロイすると、Amplify はデフォルトでサポートされている最新バージョンの Next.js を使用します。古いバージョンの Next.js で Amplify にデプロイした既存の Next.js アプリがある場合は、そのアプリを Amplify ホスティングコンピューティングSSRプロバイダーに移行できます。手順については、「Next.js 11 SSR アプリをAmplify ホスティングコンピューティングに移行する」を参照してください。

Next.js アプリケーションにある画像が読み込まれない

next/imageコンポーネントを使用して Next.js アプリに画像を追加する場合、画像のサイズは 1 MB を超えることはできません。アプリを Amplify にデプロイすると、1 MB を超える画像は 503 エラーを返します。これは、ヘッダーと本文を含む、Lambda 関数によって生成されたレスポンスのサイズをLambda@Edge が 1 MB に制限しているためです。

1 MB の制限は、PDF やドキュメントファイルなど、アプリ内の他のアーティファクトにも適用されます。

サポートされていないリージョン

Amplify は、Amplify が利用可能なすべての AWS 地域で、クラシック (Next.js 11 のみ) SSR アプリのデプロイをサポートしているわけではありません。クラシック (Next.js 11 のみ) SSR は、欧州 (ミラノ) eu-south-1、中東 (バーレーン) me-south-1、およびアジアパシフィック (香港) ap-east-1 の各地域ではサポートされていません。

# SSR デプロイのトラブルシューティング

Amplify ホスティングコンピューティングで SSR アプリをデプロイする際に予期しない問題が発生した場合は、Amplify トラブルシューティングの章で「サーバーサイドレンダリングされたアプリケーションのトラブルシューティング」を参照してください。

# 詳細: オープンソースアダプター

フレームワークの作成者は、ファイルシステムベースのデプロイ仕様を使用して、特定のフレームワーク用にカスタマイズされたオープンソースのビルドアダプターを開発できます。これらのアダプターは、アプリケーションのビルド出力を、Amplify ホスティングの想定されるディレクトリ構造に準拠するデプロイバンドルに変換します。このデプロイバンドルには、ルーティングルールといったランタイム設定など、アプリケーションをホストするために必要なすべてのファイルとアセットが含まれます。

フレームワークを使用していない場合は、独自のソリューションを開発して、Amplify が想定するビルド出力を生成できます。

トピック

- Amplify ホスティングのデプロイ仕様を使用したビルド出力の設定
- デプロイマニフェストを使用した Express サーバーのデプロイ
- フレームワークの作成者向けの画像の最適化の統合
- SSR フレームワークのオープンソースアダプターの使用

## Amplify ホスティングのデプロイ仕様を使用したビルド出力の設定

Amplify ホスティングのデプロイ仕様は、Amplify ホスティングへのデプロイを容易にするディレクトリ構造を定義するファイルシステムベースの仕様です。フレームワークは、ビルドコマンドの出力としてこの想定されるディレクトリ構造を生成できます。これにより、フレームワークは Amplify ホ

スティングのサービスプリミティブを利用できるようになります。Amplify ホスティングは、デプロイバンドルの構造を理解し、それに応じてデプロイします。

デプロイ仕様の使用方法を説明するデモ動画については、Amazon Web Services YouTube チャンネルで「 AWS Amplifyを使用してウェブサイトをホストする方法」を参照してください。

Amplify がデプロイバンドルについて想定するフォルダ構造の例を次に示します。大まかに言うと、`static` という名前のフォルダ、`compute` という名前のフォルダ、`deploy-manifest.json` という名前のデプロイマニフェストファイルがあります。

```
.amplify-hosting/
### compute/
#   ### default/
#       ### chunks/
#       #   ### app/
#       #       ### _nuxt/
#       #       #   ### index-xxx.mjs
#       #       #   ### index-styles.xxx.js
#       #       ### server.mjs
#       ### node_modules/
#       ### server.js
### static/
#   ### css/
#   #   ### nuxt-google-fonts.css
#   ### fonts/
#   #   ### font.woff2
#   ### _nuxt/
#   #   ### builds/
#   #   #   ### latest.json
#   #   ### entry.xxx.js
#   ### favicon.ico
#   ### robots.txt
### deploy-manifest.json
```

## Amplify SSR プリミティブのサポート

Amplify ホスティングのデプロイ仕様は、次のプリミティブに機密にマッピングされるコントラクトを定義します。

静的アセット

静的ファイルをホストする機能をフレームワークに提供します。

コンピューティング

ポート 3000 で Node.js HTTP サーバーを実行する機能をフレームワークに提供します。

画像の最適化

実行時に画像を最適化するサービスをフレームワークに提供します。

ルーティングルール

着信リクエストのパスを特定のターゲットにマッピングするメカニズムをフレームワークに提供します。

## .amplify-hosting/static ディレクトリ

アプリケーション URL から提供される、パブリックにアクセス可能なすべての静的ファイルを `.amplify-hosting/static` ディレクトリに格納する必要があります。このディレクトリ内のファイルは、静的アセットのプリミティブを介して提供されます。

静的ファイルには、内容、ファイル名、または拡張子の変更なしで、アプリケーション URL のルート (/) でアクセスできます。さらに、サブディレクトリは URL 構造内に保持され、ファイル名の前に表示されます。一例として、`.amplify-hosting/static/favicon.ico` は `https://myAppId.amplify-hostingapp.com/favicon.ico` から提供され、`.amplify-hosting/static/_nuxt/main.js` は `https://myAppId.amplify-hostingapp.com/_nuxt/main.js` から提供されます

フレームワークがアプリケーションのベースパスを変更する機能をサポートしている場合は、`.amplify-hosting/static` ディレクトリ内の静的アセットへのベースパスを先頭に付加する必要があります。例えば、ベースパスが `/folder1/folder2` である場合、`main.css` という静的アセットのビルド出力は `.amplify-hosting/static/folder1/folder2/main.css` になります。

## .amplify-hosting/compute ディレクトリ

単一のコンピューティングリソースは、`.amplify-hosting/compute` ディレクトリ内に含まれる `default`という名前の単一のサブディレクトリによって表されます。パスは `.amplify-hosting/compute/default` です。このコンピューティングリソースは、Amplify ホスティングのコンピューティングプリミティブにマッピングされます。

default サブディレクトリの内容は、次のルールに準拠する必要があります。

- コンピューティングリソースへのエントリポイントとして機能するファイルは、default サブディレクトリのルートに存在する必要があります。
- エントリポイントファイルは Node.js モジュールでなければならず、ポート3000 でリッスンする HTTP サーバーを起動する必要があります。
- 他のファイルを default サブディレクトリに格納し、エントリポイントファイルのコードからそれらのファイルを参照できます。
- サブディレクトリの内容は自己完結型である必要があります。エントリポイントモジュール内のコードは、サブディレクトリの外部のモジュールを参照できません。フレームワークは任意の方法で HTTP サーバーをバンドルできることに留意してください。サブディレクトリ内から node server.js コマンド (ここで server.js is はエントリファイルの名前) を使用してコンピューティングプロセスを開始できる場合、Amplify は、ディレクトリ構造がデプロイ仕様に準拠しているものとみなします。

Amplify ホスティングは、default サブディレクトリ内のすべてのファイルをバンドルし、プロビジョニングされたコンピューティングリソースにデプロイします。各コンピューティングリソースには、512 MB のエフェメラルストレージが割り当てられます。このストレージは実行インスタンス間では共有されませんが、同じ実行インスタンス内での後続の呼び出しの間では共有されます。実行インスタンスの実行時間は最大 15 分に制限されており、実行インスタンス内の唯一の書き込み可能なパスは /tmp ディレクトリです。各コンピューティングリソースバンドルの非圧縮サイズは 220 MB を超えることはできません。例えば、.amplify/compute/default サブディレクトリは圧縮されていないときに、220 MB を超えることはできません。

## .amplify-hosting/deploy-manifest.json ファイル

デプロイの設定の詳細とメタデータを保存するには deploy-manifest.json ファイルを使用します。deploy-manifest.json ファイルには少なくとも、version 属性、キャッチオールルートが指定された routes 属性、およびフレームワークメタデータが指定された framework 属性が含まれている必要があります。

次のオブジェクト定義は、デプロイマニフェストの設定を示しています。

```
type DeployManifest = {
  version: 1;
  routes: Route[];
  computeResources?: ComputeResource[];
  imageSettings?: ImageSettings;
```

```
  framework: FrameworkMetadata;
};
```

次のトピックでは、デプロイマニフェストの各属性の詳細と使用法について説明します。

バージョン属性の使用

version 属性は、実装しようとしているデプロイ仕様のバージョンを定義します。現在、Amplify ホスティングのデプロイ仕様の唯一のバージョンはバージョン 1 です。次の JSON の例は、version 属性の使用法を示しています。

```
"version": 1
```

ルート属性の使用

routes 属性により、フレームワークは Amplify ホスティングのルーティングルールのプリミティブを利用できるようになります。ルーティングルールは、着信リクエストのパスをデプロイバンドル内の特定のターゲットにルーティングするメカニズムを提供します。ルーティングルールは着信リクエストの宛先のみを決定し、リクエストが書き換えルールとリダイレクトルールによって変換された後に適用されます。Amplify ホスティングが書き換えとリダイレクトを処理する方法の詳細については、「Amplify アプリケーションのリダイレクトと書き換えの設定」を参照してください。

ルーティングルールは、リクエストを書き換えたり、変換したりしません。着信リクエストがルートのパスパターンと一致する場合、リクエストはそのままルートのターゲットにルーティングされます。

routes 配列で指定されたルーティングルールは、次のルールに準拠する必要があります。

- キャッチオールルートが指定されている必要があります。キャッチオールルートには、すべての着信リクエストに一致する /* パターンがあります。
- routes 配列には最大 25 個の項目を含めることができます。
- Static ルートまたは Compute ルートのいずれかを指定する必要があります。
- Static ルートを指定する場合は、.amplify-hosting/static ディレクトリが存在している必要があります。
- Compute ルートを指定する場合は、.amplify-hosting/compute ディレクトリが存在している必要があります。
- ImageOptimization ルートを指定する場合は、Compute ルートも指定する必要があります。画像の最適化は純粋に静的なアプリケーションではまだサポートされていないため、これは必須です。

次のオブジェクト定義は、Route オブジェクトの設定を示しています。

```
type Route = {
  path: string;
  target: Target;
  fallback?: Target;
}
```

次の表では、Route オブジェクトのプロパティについて説明します。

| Key | タイプ | 必須 | 説明 |
| --- | --- | --- | --- |
| パス | String | はい | 着信リクエストのパス (クエリ文字列を除く) に一致するパターンを定義します。<br><br>最大パス長は 255 文字です。<br><br>パスはスラッシュ / で始まる必要があります。<br><br>パスには、[A-Z]、[a-z]、[0-9]、[ _-.*$/~"' @:+] の文字を使用できます。<br><br>パターンマッチングでは、次のワイルドカード文字のみがサポートされます:<br><br>• * (0 個以上の文字に一致)<br>• /* パターンはキャッチオールパターンと呼ばれ、 |

| Key | タイプ | 必須 | 説明 |
| --- | --- | --- | --- |
| | | | すべての着信リクエストに一致します。 |
| target | ターゲット | はい | 一致したリクエストのルーティング先となるターゲットを定義するオブジェクト。<br><br>Compute ルートが指定されている場合は、対応する ComputeResource ルートが存在する必要があります。<br><br>ImageOptimization ルートを指定する場合は、imageSettings も指定する必要があります。 |

| Key | タイプ | 必須 | 説明 |
| --- | --- | --- | --- |
| fallback | ターゲット | いいえ | 元のターゲットが 404 エラーを返した場合にフォールバックするターゲットを定義するオブジェクト。<br><br>指定したルートで target の種類と fallback の種類を同じにすることはできません。例えば、Static から Static へのフォールバックは許可されません。フォールバックは、本文のない GET リクエストでのみサポートされます。リクエストに本文が存在する場合、その本文はフォールバック中にドロップされます。 |

次のオブジェクト定義は、Target オブジェクトの設定を示しています。

```
type Target = {
  kind: TargetKind;
  src?: string;
  cacheControl?: string;
}
```

次の表では、Target オブジェクトのプロパティについて説明します。

| Key | タイプ | 必須 | 説明 |
| --- | --- | --- | --- |
| kind | Targetkind | はい | ターゲットのタイプを定義する enum。有効な値は、Static、Compute、ImageOptimization です。 |
| src | String | Compute: はい<br><br>他のプリミティブ: いいえ | プリミティブの実行可能コードを含むデプロイバンドル内のサブディレクトリの名前を指定する文字列。コンピューティングプリミティブでのみ有効かつ必須です。<br><br>値は、デプロイバンドルに存在するコンピューティングリソースの 1 つをポイントする必要があります。現在、このフィールドでサポートされている値は default のみです。 |
| cacheControl | String | いいえ | 応答に適用する Cache-Control ヘッダーの値を指定する文字列。Static および ImageOptimization プリミティブでのみ有効です。 |

| Key | タイプ | 必須 | 説明 |
| --- | --- | --- | --- |
| | | | 指定された値は、カスタムヘッダーによってオーバーライドされます。Amplify ホスティングのカスタマーヘッダーの詳細については、「Amplify アプリのカスタムヘッダーの設定」を参照してください。<br><br>**Note**<br>このキャッシュコントロールヘッダーは、ステータスコードが 200 (OK) に設定された正常なレスポンスにのみ適用されます。 |

次のオブジェクト定義は、TargetKind 列挙型の使用法を示しています。

```
enum TargetKind {
  Static = "Static",
  Compute = "Compute",
  ImageOptimization = "ImageOptimization"
}
```

次のリストは、TargetKind 列挙型の有効な値を指定します。

静的

リクエストを静的アセットのプリミティブにルーティングします。

コンピューティング

リクエストをコンピューティングプリミティブにルーティングします。

ImageOptimization

リクエストを画像の最適化のプリミティブにルーティングします。

次の JSON の例は、複数のルーティングルールが指定された routes 属性の使用法を示しています。

```
"routes": [
    {
      "path": "/_nuxt/image",
      "target": {
        "kind": "ImageOptimization",
        "cacheControl": "public, max-age=3600, immutable"
      }
    },
    {
      "path": "/_nuxt/builds/meta/*",
      "target": {
        "cacheControl": "public, max-age=31536000, immutable",
        "kind": "Static"
      }
    },
    {
      "path": "/_nuxt/builds/*",
      "target": {
        "cacheControl": "public, max-age=1, immutable",
        "kind": "Static"
      }
    },
    {
      "path": "/_nuxt/*",
      "target": {
        "cacheControl": "public, max-age=31536000, immutable",
        "kind": "Static"
      }
    },
```

```
    {
      "path": "/*.*",
      "target": {
        "kind": "Static"
      },
      "fallback": {
        "kind": "Compute",
        "src": "default"
      }
    },
    {
      "path": "/*",
      "target": {
        "kind": "Compute",
        "src": "default"
      }
    }
  ]
```

デプロイマニフェストでのルーティングルールの指定の詳細については、「ルーティングルールの設定に関するベストプラクティス」を参照してください。

computeResources 属性の使用

computeResources 属性により、フレームワークは、プロビジョニングされたコンピューティングリソースに関するメタデータを提供できるようになります。あらゆるコンピューティングリソースには、対応するルートが関連付けられている必要があります。

次のオブジェクト定義は、ComputeResource オブジェクトの使用法を示しています。

```
type ComputeResource = {
  name: string;
  runtime: ComputeRuntime;
  entrypoint: string;
};

type ComputeRuntime = 'nodejs20.x' | 'nodejs22.x';
```

次の表では、ComputeResource オブジェクトのプロパティについて説明します。

| Key | タイプ | 必須 | 説明 |
| --- | --- | --- | --- |
| 名前 | String | はい | コンピューティングリソースの名前を指定します。この名前は、`.amplify-hosting/compute` directory 内のサブディレクトリの名前と一致する必要があります。<br><br>デプロイ仕様のバージョン 1 である場合、有効な値は `default` のみです。 |
| ランタイム | ComputeRuntime | はい | プロビジョニングされたコンピューティングリソースのランタイムを定義します。<br><br>有効な値は、`nodejs20.x` および `nodejs22.x` です。 |
| entrypoint | String | はい | 指定されたコンピューティングリソースのためにコードが実行される開始ファイルの名前を指定します。このファイルは、コンピューティングリソースを表すサ |

| Key | タイプ | 必須 | 説明 |
| --- | --- | --- | --- |
| | | | ブディレクトリ内に存在している必要があります。 |

次のようなディレクトリ構造があるとします。

```
.amplify-hosting
|---compute
|   |---default
|       |---index.js
```

computeResource 属性の JSON は次のようになります。

```
"computeResources": [
    {
      "name": "default",
      "runtime": "nodejs20.x",
      "entrypoint": "index.js",
    }
  ]
```

### imageSettings 属性の使用

imageSettings 属性を使用すると、フレームワークは画像の最適化のプリミティブの動作をカスタマイズでき、実行時における画像のオンデマンド最適化を提供します。

次のオブジェクト定義は、ImageSettings オブジェクトの使用法を示しています。

```
type ImageSettings = {
  sizes: number[];
  domains: string[];
  remotePatterns: RemotePattern[];
  formats: ImageFormat[];
  minumumCacheTTL: number;
  dangerouslyAllowSVG: boolean;
};

type ImageFormat = 'image/avif' | 'image/webp' | 'image/png' | 'image/jpeg';
```

次の表では、`ImageSettings` オブジェクトのプロパティについて説明します。

| Key | タイプ | 必須 | 説明 |
| --- | --- | --- | --- |
| sizes | Number[] | はい | サポートされている画像の幅の配列。 |
| domains | String[] | はい | 画像の最適化を使用できる許可された外部ドメインの配列。デプロイドメインのみが画像の最適化を使用できるようにするには、配列を空のままにしておきます。 |
| remotePatterns | RemotePattern[] | はい | 画像の最適化を使用できる許可された外部パターンの配列。ドメインに似ていますが、正規表現 (regex) によりさらに詳細なコントロールを提供します。 |
| formats | ImageFormat[] | はい | 許可される出力画像形式の配列。 |
| minimumCacheTTL | Number | はい | 最適化された画像のキャッシュ期間 (秒)。 |
| dangerouslyAllowSVG | ブール値 | はい | SVG 入力画像 URL を許可します。これは、セキュリティ上の理由からデフォルトでは無効になっています。 |

次のオブジェクト定義は、RemotePattern オブジェクトの使用法を示しています。

```
type RemotePattern = {
  protocol?: 'https';
  hostname: string;
  port?: string;
  pathname?: string;
}
```

次の表では、RemotePattern オブジェクトのプロパティについて説明します。

| Key | タイプ | 必須 | 説明 |
| --- | --- | --- | --- |
| protocol | String | いいえ | 許可されるリモートパターンのプロトコル。唯一の有効な値は `https` です。 |
| hostname | String | はい | 許可されるリモートパターンのホスト名。<br><br>リテラルまたはワイルドカードを指定できます。単一の「*」は単一のサブドメインに一致します。二重の「**」は任意の数のサブドメインに一致します。Amplify では、「**」のみが指定されるブランケットワイルドカードを使用できません。 |
| port | String | いいえ | 許可されるリモートパターンのポート。 |

| Key | タイプ | 必須 | 説明 |
| --- | --- | --- | --- |
| pathname | String | いいえ | 許可されるリモートパターンのパス名。 |

次の例は、imageSettings 属性を示しています。

```
"imageSettings": {
    "sizes": [
      100,
      200
    ],
    "domains": [
      "example.com"
    ],
    "remotePatterns": [
      {
        "protocol": "https",
        "hostname": "example.com",
        "port": "",
        "pathname": "/**",
      }
    ],
    "formats": [
      "image/webp"
    ],
    "minumumCacheTTL": 60,
    "dangerouslyAllowSVG": false
  }
```

フレームワーク属性の使用

framework 属性を使用してフレームワークのメタデータを指定します。

次のオブジェクト定義は、FrameworkMetadata オブジェクトの設定を示しています。

```
type FrameworkMetadata = {
  name: string;
  version: string;
}
```

次の表では、FrameworkMetadata オブジェクトのプロパティについて説明します。

| Key | タイプ | 必須 | 説明 |
| --- | --- | --- | --- |
| 名前 | String | はい | フレームワークの名前。 |
| バージョン | String | はい | フレームワークのバージョン。<br><br>有効なセマンティックバージョニング (semver) 文字列である必要があります。 |

## ルーティングルールの設定に関するベストプラクティス

ルーティングルールは、着信リクエストのパスをデプロイバンドル内の特定のターゲットにルーティングするメカニズムを提供します。デプロイバンドルでは、フレームワークの作成者は、次のターゲットのいずれかにデプロイされるファイルをビルド出力に出力できます:

- 静的アセットプリミティブ – ファイルは .amplify-hosting/static ディレクトリに含まれます。
- コンピューティングプリミティブ – ファイルは .amplify-hosting/compute/default ディレクトリに含まれます。

また、フレームワークの作成者は、デプロイマニフェストファイルでルーティングルールの配列も提供します。配列内の各ルールは、一致が見つかるまで、シーケンシャルトラバーサル順序で着信リクエストと照合されます。一致するルールがある場合、リクエストは一致ルールで指定されたターゲットにルーティングされます。オプションで、ルールごとにフォールバックターゲットを指定できます。元のターゲットが 404 エラーを返した場合、リクエストはフォールバックターゲットにルーティングされます。

デプロイ仕様では、トラバーサル順序の最後のルールがキャッチオールルールである必要があります。キャッチオールルールは /* パスで指定されます。着信リクエストがルーティングルールの配列内の以前のルートのいずれとも一致しない場合、リクエストはキャッチオールルールのターゲットにルーティングされます。

Nuxt.js などの SSR フレームワークでは、キャッチオールルールのターゲットはコンピューティングプリミティブである必要があります。これは、SSR アプリケーションには、ビルド時に予測できないルートを含むサーバーサイドレンダリングされたページがあるためです。例えば、Nuxt.js アプリケーションでは /blog/[slug] にページがあるとします (ここで [slug] は動的ルートパラメータです)。キャッチオールルールのターゲットは、リクエストをこれらのページにルーティングする唯一の方法です。

対照的に、特定のパスパターンを使用して、ビルド時に既知のルートをターゲットにすることができます。例えば、Nuxt.js は /_nuxt パスから静的アセットを提供します。これは、リクエストを静的アセットプリミティブにルーティングする特定のルーティングルールによって /_nuxt/* パスをターゲットとすることができることを意味します。

パブリックフォルダのルーティング

ほとんどの SSR フレームワークは、public フォルダから変更可能な静的アセットを提供する機能を提供します。favicon.ico や robots.txt のようなファイルは通常、public フォルダ内に保存され、アプリケーションのルート URL から提供されます。例えば、favicon.ico ファイルは https://example.com/favicon.ico から提供されます。これらのファイルには予測可能なパスパターンがないことに留意してください。それらは、ほぼ完全にファイル名によって決まります。public フォルダ内のファイルをターゲットにする唯一の方法は、キャッチオールルートを使用することです。ただし、キャッチオールルートのターゲットはコンピューティングプリミティブである必要があります。

public フォルダを管理するには、次のいずれかのアプローチをお勧めします。

1. ファイル拡張子を含むリクエストパスをターゲットにするためにパスパターンを使用します。例えば、ファイル拡張子を含むすべてのリクエストパスをターゲットにするために　/*.* を使用できます。

   このアプローチは信頼できない可能性があることに留意してください。例えば、public フォルダ内にファイル拡張子のないファイルが存在する場合、それらのファイルはこのルールのターゲットになりません。このアプローチで留意すべきもう 1 つの問題は、名前にピリオドが含まれるページがアプリケーションに存在する可能性があることです。例えば、/blog/2021/01/01/hello.world のページは /*.*　ルールのターゲットになります。ページは静的アセットではないため、これは理想的ではありません。ただし、このルールにフォールバックターゲットを追加して、静的プリミティブで 404 エラーが発生した場合に、リクエストがコンピューティングプリミティブにフォールバックされるようにすることができます。

   ```
   {
   ```

```
    "path": "/*.*",
    "target": {
        "kind": "Static"
    },
    "fallback": {
        "kind": "Compute",
        "src": "default"
    }
}
```

2. ビルド時に public フォルダ内のファイルを識別し、各ファイルのルーティングルールを出力します。デプロイ仕様によって課されるルールの数が 25 個に制限されているため、このアプローチはスケーラブルではありません。

```
{
    "path": "/favicon.ico",
    "target": {
        "kind": "Static"
    }
},
{
    "path": "/robots.txt",
    "target": {
        "kind": "Static"
    }
}
```

3. フレームワークのユーザーがすべてのミュータブルな静的アセットを public フォルダ内のサブフォルダに保存することを推奨します。

   次の例では、ユーザーはすべてのミュータブルな静的アセットを public/assets フォルダ内に保存できます。その後、パスパターン /assets/* を含むルーティングルールを使用して、public/assets フォルダ内のすべてのミュータブルな静的アセットをターゲットにすることができます。

```
{
    "path": "/assets/*",
    "target": {
        "kind": "Static"
    }
}
```

4. キャッチオールルートの静的フォールバックを指定します。このアプローチには欠点があり、次の キャッチオールフォールバックルーティング セクションで詳しく説明します。

### キャッチオールフォールバックルーティング

コンピューティングプリミティブターゲットにキャッチオールルートが指定されている Nuxt.js などの SSR フレームワークでは、フレームワークの作成者は、public フォルダのルーティングに関する問題を解決するために、キャッチオールルートに静的フォールバックを指定することを検討することがあります。しかし、このタイプのルーティングルールでは、サーバーサイドレンダリングされた 404 ページが壊れます。例えば、存在しないページにエンドユーザーがアクセスすると、アプリケーションはステータスコード 404 で 404 ページを表示します。しかし、キャッチオールルートに静的フォールバックがある場合、404 ページはレンダリングされません。代わりに、リクエストは静的プリミティブにフォールバックし、依然として 404 ステータスコードで終了しますが、404 ページはレンダリングされません。

```
{
    "path": "/*",
    "target": {
        "kind": "Compute",
        "src": "default"
    },
    "fallback": {
        "kind": "Static"
    }
}
```

### ベースパスルーティング

アプリケーションのベースパスを変更する機能を提供するフレームワークは、.amplify-hosting/static ディレクトリ内の静的アセットへのベースパスを先頭に付加することが想定されます。例えば、ベースパスが /folder1/folder2 である場合、main.css という静的アセットのビルド出力は .amplify-hosting/static/folder1/folder2/main.css になります。

これは、ベースパスを反映するためにルーティングルールも更新する必要があることを意味します。例えば、ベースパスが /folder1/folder2 である場合、public フォルダ内の静的アセットのルーティングルールは次のようになります。

```
{
    "path": "/folder1/folder2/*.*",
    "target": {
```

```
        "kind": "Static"
    }
}
```

同様に、サーバー側のルートにもベースパスを先頭に付加する必要があります。例えば、ベースパスが /folder1/folder2 である場合、/api ルートのルーティングルールは次のようになります。

```
{
    "path": "/folder1/folder2/api/*",
    "target": {
        "kind": "Compute",
        "src": "default"
    }
}
```

ただし、ベースパスをキャッチオールルートの先頭に付加しないでください。例えば、ベースパスが /folder1/folder2 である場合、キャッチオールルートは次のようになります。

```
{
    "path": "/*",
    "target": {
        "kind": "Compute",
        "src": "default"
    }
}
```

Nuxt.js ルートの例

ルーティングルールを指定する方法を示す Nuxt アプリケーションのサンプル deploy-manifest.json ファイルを次に示します。

```
{
  "version": 1,
  "routes": [
    {
      "path": "/_nuxt/image",
      "target": {
        "kind": "ImageOptimization",
        "cacheControl": "public, max-age=3600, immutable"
      }
    },
    {
```

```
      "path": "/_nuxt/builds/meta/*",
      "target": {
        "cacheControl": "public, max-age=31536000, immutable",
        "kind": "Static"
      }
    },
    {
      "path": "/_nuxt/builds/*",
      "target": {
        "cacheControl": "public, max-age=1, immutable",
        "kind": "Static"
      }
    },
    {
      "path": "/_nuxt/*",
      "target": {
        "cacheControl": "public, max-age=31536000, immutable",
        "kind": "Static"
      }
    },
    {
      "path": "/*.*",
      "target": {
        "kind": "Static"
      },
      "fallback": {
        "kind": "Compute",
        "src": "default"
      }
    },
    {
      "path": "/*",
      "target": {
        "kind": "Compute",
        "src": "default"
      }
    }
  ],
  "computeResources": [
    {
      "name": "default",
      "entrypoint": "server.js",
      "runtime": "nodejs22.x"
    }
```

```
  ],
  "framework": {
    "name": "nuxt",
    "version": "3.8.1"
  }
}
```

ベースパスを含むルーティングルールを指定する方法を示す Nuxt のサンプル deploy-manifest.json ファイルを次に示します。

```
{
  "version": 1,
  "routes": [
    {
      "path": "/base-path/_nuxt/image",
      "target": {
        "kind": "ImageOptimization",
        "cacheControl": "public, max-age=3600, immutable"
      }
    },
    {
      "path": "/base-path/_nuxt/builds/meta/*",
      "target": {
        "cacheControl": "public, max-age=31536000, immutable",
        "kind": "Static"
      }
    },
    {
      "path": "/base-path/_nuxt/builds/*",
      "target": {
        "cacheControl": "public, max-age=1, immutable",
        "kind": "Static"
      }
    },
    {
      "path": "/base-path/_nuxt/*",
      "target": {
        "cacheControl": "public, max-age=31536000, immutable",
        "kind": "Static"
      }
    },
    {
      "path": "/base-path/*.*",
```

```
      "target": {
        "kind": "Static"
      },
      "fallback": {
        "kind": "Compute",
        "src": "default"
      }
    },
    {
      "path": "/*",
      "target": {
        "kind": "Compute",
        "src": "default"
      }
    }
  ],
  "computeResources": [
    {
      "name": "default",
      "entrypoint": "server.js",
      "runtime": "nodejs22.x"
    }
  ],
  "framework": {
    "name": "nuxt",
    "version": "3.8.1"
  }
}
```

routes 属性の使用に関する詳細については、「ルート属性の使用」を参照してください。

## デプロイマニフェストを使用した Express サーバーのデプロイ

この例では、Amplify ホスティングのデプロイ仕様を使用して基本的な Express サーバーをデプロイする方法を説明します。提供されたデプロイマニフェストを利用して、ルーティング、コンピューティングリソース、および他の設定を指定できます。

Amplify ホスティングにデプロイする前に、Express サーバーをローカルに設定する

1. プロジェクト用に新しいディレクトリを作成し、Express と Typescript をインストールします。

   ```
   mkdir express-app
   ```

```
cd express-app

# The following command will prompt you for information about your project
npm init

# Install express, typescript and types
npm install express --save
npm install typescript ts-node @types/node @types/express --save-dev
```

2. 次の内容を含む tsconfig.json ファイルを、プロジェクトのルートに追加します。

```
{
  "compilerOptions": {
    "target": "es6",
    "module": "commonjs",
    "outDir": "./dist",
    "strict": true,
    "esModuleInterop": true,
    "skipLibCheck": true,
    "forceConsistentCasingInFileNames": true
  },
  "include": ["src/**/*.ts"],
  "exclude": ["node_modules"]
}
```

3. プロジェクトのルートに src という名前のディレクトリを作成します。
4. src ディレクトリ内に index.ts ファイルを作成します。これは、Express サーバーを起動するアプリケーションへのエントリポイントになります。サーバーはポート 3000 でリッスンするように設定する必要があります。

```
// src/index.ts
import express from 'express';

const app: express.Application = express();
const port = 3000;

app.use(express.text());

app.listen(port, () => {
  console.log(`server is listening on ${port}`);
});
```

```
// Homepage
app.get('/', (req: express.Request, res: express.Response) => {
  res.status(200).send("Hello World!");
});

// GET
app.get('/get', (req: express.Request, res: express.Response) => {
  res.status(200).header("x-get-header", "get-header-value").send("get-response-
from-compute");
});

//POST
app.post('/post', (req: express.Request, res: express.Response) => {
  res.status(200).header("x-post-header", "post-header-
value").send(req.body.toString());
});

//PUT
app.put('/put', (req: express.Request, res: express.Response) => {
  res.status(200).header("x-put-header", "put-header-
value").send(req.body.toString());
});

//PATCH
app.patch('/patch', (req: express.Request, res: express.Response) => {
  res.status(200).header("x-patch-header", "patch-header-
value").send(req.body.toString());
});

// Delete
app.delete('/delete', (req: express.Request, res: express.Response) => {
  res.status(200).header("x-delete-header", "delete-header-value").send();
});
```

5. 次のスクリプトを package.json ファイルに追加します。

```
"scripts": {
  "start": "ts-node src/index.ts",
  "build": "tsc",
  "serve": "node dist/index.js"
}
```

6. プロジェクトのルートに public という名前のディレクトリを作成します。その後、次の内容で、hello-world.txt という名前のファイルを作成します。

```
Hello world!
```

7. 次の内容を含む .gitignore ファイルを、プロジェクトルートに追加します。

```
.amplify-hosting
dist
node_modules
```

### Amplify のデプロイマニフェストを設定する

1. プロジェクトのルートディレクトリに、deploy-manifest.json という名前のファイルを作成します。
2. 次のマニフェストをコピーして deploy-manifest.json ファイルに貼り付けます。

```
{
  "version": 1,
  "framework": { "name": "express", "version": "4.18.2" },
  "imageSettings": {
    "sizes": [
      100,
      200,
      1920
    ],
    "domains": [],
    "remotePatterns": [],
    "formats": [],
    "minimumCacheTTL": 60,
    "dangerouslyAllowSVG": false
  },
  "routes": [
    {
      "path": "/_amplify/image",
      "target": {
        "kind": "ImageOptimization",
        "cacheControl": "public, max-age=3600, immutable"
      }
    },
    {
```

```
      "path": "/*.*",
      "target": {
        "kind": "Static",
        "cacheControl": "public, max-age=2"
      },
      "fallback": {
        "kind": "Compute",
        "src": "default"
      }
    },
    {
      "path": "/*",
      "target": {
        "kind": "Compute",
        "src": "default"
      }
    }
  ],
  "computeResources": [
    {
      "name": "default",
      "runtime": "nodejs22.x",
      "entrypoint": "index.js"
    }
  ]
}
```

マニフェストには、Amplify ホスティングがアプリケーションのデプロイを処理する方法が記述されています。主な設定は次のとおりです。

- version – 使用しているデプロイ仕様のバージョンを示します。
- framework – これを調整して Express サーバー設定を指定します。
- imageSettings – 画像の最適化を処理する場合を除き、このセクションは Express サーバー用のオプションです。
- routes – これらは、トラフィックをアプリケーションの適切な部分にルーティングするために重要です。`"kind": "Compute"` ルートはトラフィックをサーバーロジックにルーティングします。
- computeResources – このセクションを使用して、Express サーバーのランタイムとエントリポイントを指定します。

次に、ビルドされたアプリケーションアーティファクトを .amplify-hosting デプロイバンドルに移動するビルド後スクリプトを設定します。ディレクトリ構造は、Amplify ホスティングのデプロイ仕様と整合しています。

ビルド後のスクリプトを設定する

1. プロジェクトのルートに bin という名前のディレクトリを作成します。
2. bin ディレクトリに postbuild.sh という名前のファイルを作成します。postbuild.sh ファイルに次の内容を追加します。

```
#!/bin/bash

rm -rf ./.amplify-hosting

mkdir -p ./.amplify-hosting/compute

cp -r ./dist ./.amplify-hosting/compute/default
cp -r ./node_modules ./.amplify-hosting/compute/default/node_modules

cp -r public ./.amplify-hosting/static

cp deploy-manifest.json ./.amplify-hosting/deploy-manifest.json
```

3. package.json ファイルに postbuild スクリプトを追加します。ファイルは次のようになっているはずです。

```
"scripts": {
  "start": "ts-node src/index.ts",
  "build": "tsc",
  "serve": "node dist/index.js",
  "postbuild": "chmod +x bin/postbuild.sh && ./bin/postbuild.sh"
}
```

4. アプリケーションを構築するには、次のコマンドを実行します。

```
npm run build
```

5. (オプション) Express のルートを調整します。Express サーバーに合わせてデプロイマニフェスト内のルートを変更できます。例えば、public ディレクトリに静的アセットがない場合は、コンピューティングにルーティングするキャッチオールルート "path": "/*" のみが必要になる可能性があります。これはサーバーの設定によって異なります。

最終的なディレクトリ構造は次のようになります。

```
express-app/
### .amplify-hosting/
#   ### compute/
#   #   ### default/
#   #       ### node_modules/
#   #       ### index.js
#   ### static/
#   #   ### hello.txt
#   ### deploy-manifest.json
### bin/
#   ### .amplify-hosting/
#   #   ### compute/
#   #   #   ### default/
#   #   ### static/
#   ### postbuild.sh*
### dist/
#   ### index.js
### node_modules/
### public/
#   ### hello.txt
### src/
#   ### index.ts
### deploy-manifest.json
### package.json
### package-lock.json
### tsconfig.json
```

サーバーをデプロイする

1. コードを Git リポジトリにプッシュし、アプリケーションを Amplify ホスティングにデプロイします。
2. 次のとおり、baseDirectory が .amplify-hosting をポイントするようにビルド設定を更新します。ビルド中に、Amplify は .amplify-hosting ディレクトリ内のマニフェストファイルを検出し、設定されたとおりに Express サーバーをデプロイします。

```
version: 1
frontend:
  phases:
    preBuild:
      commands:
```

```
        - nvm use 18
        - npm install
    build:
      commands:
        - npm run build
  artifacts:
    baseDirectory: .amplify-hosting
    files:
      - '**/*'
```

3. デプロイが成功し、サーバーが正しく実行されていることを検証するには、Amplify ホスティングによって提供されるデフォルトの URL でアプリケーションにアクセスします。

## フレームワークの作成者向けの画像の最適化の統合

フレームワークの作成者は、Amplify ホスティングのデプロイ仕様を使用して、Amplify の画像の最適化機能を統合できます。画像の最適化を有効にするには、画像の最適化サービスをターゲットとするルーティングルールがデプロイマニフェストに含まれている必要があります。次の例は、ルーティングルールを設定する方法を示しています。

```
// .amplify-hosting/deploy-manifest.json

{
  "routes": [
    {
      "path": "/images/*",
      "target": {
        "kind": "ImageOptimization",
        "cacheControl": "public, max-age=31536000, immutable"
      }
    }
  ]
}
```

デプロイ仕様を使用した画像の最適化設定の構成の詳細については、「Amplify ホスティングのデプロイ仕様を使用したビルド出力の設定」を参照してください。

### 画像の最適化 API について

画像の最適化は、ルーティングルールによって定義されたパスで、Amplify アプリケーションのドメイン URL を介して実行時に呼び出すことができます。

```
GET https://{appDomainName}/{path}?{queryParams}
```

画像の最適化では、画像に対して次のルールが適用されます。

- Amplify は、GIF、APNG、SVG 形式を最適化したり、別の形式に変換したりすることはできません。
- `dangerouslyAllowSVG` 設定が有効になっていない限り、SVG 画像は提供されません。
- ソース画像の幅または高さは、11 MB または 9,000 ピクセルを超えることはできません。
- 最適化された画像のサイズ制限は 4 MB です。
- リモート URL を使用して画像を取得するためにサポートされているプロトコルは、HTTPS のみです。

HTTP ヘッダー

Accept リクエスト HTTP ヘッダーは、クライアント (通常はウェブブラウザ) によって許可される、MIME タイプとして表現される画像形式を指定するために使用されます。画像の最適化サービスは、指定された形式への画像の変換を試みます。このヘッダーに指定された値は、形式クエリパラメータよりも優先されます。例えば、Accept ヘッダーの有効な値は `image/png, image/webp, */*` です。Amplify のデプロイマニフェストで指定された形式設定により、形式がリスト内の形式に制限されます。Accept ヘッダーが特定の形式を要求しても、その形式が許可リストに含まれていない場合は無視されます。

URI リクエストパラメータ

次の表は、画像の最適化のための URI リクエストパラメータについて説明したものです。

| クエリパラメーター | タイプ | 必須 | 説明 | 例 |
| --- | --- | --- | --- | --- |
| url | String | はい | ソース画像への相対パスまたは絶対 URL。リモート URL では、https プロトコルのみがサポートされています。値は URL | `?url=https%3A%2F%2Fwww.example.com%2Fbuffalo.png` |

| クエリパラメーター | タイプ | 必須 | 説明 | 例 |
| --- | --- | --- | --- | --- |
| | | | エンコードされている必要があります。 | |
| width | Number | はい | 最適化された画像の幅 (ピクセル)。 | `?width=800` |
| height | Number | いいえ | 最適化された画像の高さ (ピクセル)。指定しない場合、画像は幅に合わせて自動的に調整されます。 | `?height=600` |
| fit | 列挙型の値: `cover`、`contain` | いいえ | 指定された幅と高さに合わせて画像のサイズを変更する方法。 | `?width=800&height=600&fit=cover` |
| position | 列挙型の値: `center`、`top`、`ri` | いいえ | fit が `cover` または `contain` である場合に使用される位置。 | `?fit=contain&position=centre` |
| trim | Number | いいえ | 左上のピクセルの指定された背景色に類似した値を含むピクセルをすべてのエッジからトリミングします。 | `?trim=50` |

| クエリパラメーター | タイプ | 必須 | 説明 | 例 |
| --- | --- | --- | --- | --- |
| 拡張 | オブジェクト | いいえ | 最も近いエッジピクセルから派生した色を使用して、画像のエッジにピクセルを追加します。形式は `{top}_{right}_{bottom}_{left}` です。ここで、各値は追加するピクセル数です。 | `?extend=10_0_5_0` |
| extract | オブジェクト | いいえ | 上、左、幅、高さで区切られた、指定された四角形に画像をトリミングします。形式は {left}_{top}_{width}_{right} です。ここで、各値はトリミングするピクセル数です。 | `?extract=10_0_5_0` |
| format | String | いいえ | 最適化された画像に必要な出力形式。 | `?format=webp` |

| クエリパラメーター | タイプ | 必須 | 説明 | 例 |
| --- | --- | --- | --- | --- |
| quality | Number | いいえ | 画質 (1 ~ 100)。画像の形式を変換する場合にのみ使用されます。 | `?quality=50` |
| rotate | Number | いいえ | 指定した角度 (度) で画像を回転します。 | `?rotate=45` |
| flip | ブール値 | いいえ | X 軸を挟んで垂直 (上下) に画像をミラーリングします。これは、回転する場合には常にその前に発生します。 | `?flip` |
| flop | ブール値 | いいえ | Y 軸を挟んで水平 (左右) に画像をミラーリングします。これは、回転する場合には常にその前に発生します。 | `?flop` |

| クエリパラメーター | タイプ | 必須 | 説明 | 例 |
| --- | --- | --- | --- | --- |
| sharpen | Number | いいえ | シャープニングにより、画像のエッジの鮮明さが向上します。有効な値は 0.000001 ~ 10 です。 | `?sharpen=1` |
| median | Number | いいえ | メディアンフィルターを適用します。これにより、ノイズが除去されたり、画像のエッジが滑らかになったりします。 | `?sharpen=3` |
| blur | Number | いいえ | 指定されたシグマのガウシアンぼかしを適用します。有効な値は 0.3 ~ 1,000 です。 | `?blur=20` |
| gamma | Number | いいえ | ガンマ補正を適用して、サイズ変更された画像の知覚される明るさを改善します。値は 1.0 ~ 3.0 である必要があります。 | `?gamma=1` |

| クエリパラメーター | タイプ | 必須 | 説明 | 例 |
| --- | --- | --- | --- | --- |
| negate | ブール値 | いいえ | 画像の色を反転します。 | `?negate` |
| normalize | ブール値 | いいえ | ダイナミックレンジ全体をカバーするように輝度を広げることにより、画像のコントラストを上げます。 | `?normalize` |
| threshold | Number | いいえ | 明度が指定されたしきい値よりも小さい場合、画像内のピクセルを黒のピクセルに置き換えます。または、しきい値より大きい場合は白いピクセルに置き換えます。有効な値は 0 ~ 255 です。 | `?threshold=155` |
| tint | String | いいえ | 画像の輝度を維持しながら、指定された RGB を使用して画像に色合いを付けます。 | `?tint=#7743CE` |

| クエリパラメーター | タイプ | 必須 | 説明 | 例 |
| --- | --- | --- | --- | --- |
| grayscale | ブール値 | いいえ | 画像をグレースケール (白黒) に変換します。 | `?grayscale` |

レスポンスステータスコード

次のリストでは、画像の最適化の応答ステータスコードについて説明します。

成功 - HTTP ステータスコード 200

リクエストは正常に完了しました。

BadRequest - HTTP ステータスコード 400

- 入力クエリパラメータの指定が間違っています。
- リモート URL は `remotePatterns` の設定で許可されているものとしてリストされていません。
- リモート URL は画像に解決されません。
- リクエストされた幅または高さが、`sizes` の設定で許可されているものとしてリストされていません。
- リクエストされた画像は SVG ですが、`dangerouslyAllowSvg` の設定が無効になっています。

見つかりません - HTTP ステータスコード 404

ソース画像が見つかりませんでした。

コンテンツが大きすぎます - HTTP ステータスコード 413

ソース画像または最適化された画像のいずれかが、最大許容サイズ (バイト) を超えています。

最適化された画像のキャッシュについて

Amplify ホスティングは最適化された画像を CDN にキャッシュするため、同じクエリパラメータを使用した同じ画像に対する後続のリクエストはキャッシュから提供されます。キャッシュの存続時間 (TTL) は `Cache-Control` ヘッダーによって制御されます。次のリストでは、`Cache-Control` ヘッダーを指定するためのオプションについて説明します。

- 画像の最適化をターゲットとするルーティングルール内で Cache-Control キーを使用します。
- Amplify アプリケーションで定義されたカスタムヘッダーを使用します。
- リモート画像の場合、リモート画像によって返された Cache-Control ヘッダーが準拠されます。

画像の最適化の設定で指定された minimumCacheTTL は、Cache-Control max-age ディレクティブの下限を定義します。例えば、リモート画像 URL が Cache-Control s-max-age=10 で応答するが、minimumCacheTTL の値が 60 である場合、60 が使用されます。

## SSR フレームワークのオープンソースアダプターの使用

Amplify ホスティングとの統合用に作成された SSR フレームワークビルドアダプターを使用できます。アダプターを提供する各フレームワークは、どのようにアダプターが設定され、ビルドプロセスに接続されるかを決定します。通常、アダプターは npm 開発の依存関係としてインストールします。

フレームワークを使用してアプリケーションを作成した後、フレームワークのドキュメントを参照して、Amplify ホスティングアダプターをインストールし、アプリケーションの設定ファイルで設定する方法を確認してください。

次に、プロジェクトのルートディレクトリに amplify.yml ファイルを作成します。amplify.yml ファイル内で、baseDirectory をアプリケーションのビルド出力ディレクトリに設定します。フレームワークはビルドプロセス中にアダプターを実行し、出力を Amplify ホスティングデプロイバンドルに変換します。

ビルド出力ディレクトリの名前は任意ですが、.amplify-hosting のファイル名には意味があります。Amplify はまず、baseDirectory として定義されたディレクトリを探します。存在する場合、Amplify はそこにあるビルド出力を探します。ディレクトリが存在しない場合、Amplify は、お客様によって定義されていない場合でも、.amplify-hosting 内のビルド出力を検索します。

アプリケーションのビルド設定の例を次に示します。baseDirectory は、ビルド出力が .amplify-hosting フォルダ内にあることを示すために .amplify-hosting に設定されます。.amplify-hosting フォルダの内容が Amplify ホスティングのデプロイ仕様と一致している限り、アプリケーションは正常にデプロイされます。

```
version: 1
frontend:
  preBuild:
```

```
    commands:
      - npm install
  build:
    commands:
      - npm run build
  artifacts:
    baseDirectory: .amplify-hosting
```

フレームワークアダプターを使用するようにアプリケーションを設定したら、Amplify ホスティングにデプロイできます。詳細な手順については、「SSR アプリケーションの Amplify へのデプロイ」を参照してください。

# Amazon S3 バケットから Amplify への静的ウェブサイトのデプロイ

Amplify ホスティングと Amazon S3 の統合を使用すれば、S3 に保存されている静的ウェブサイトコンテンツを数回クリックするだけでホストできます。Amplify ホスティングにデプロイすると、以下の利点と機能が得られます。

- CloudFront を使用したグローバルに利用可能な AWS コンテンツ配信ネットワーク (CDN) への自動デプロイ
- HTTPS サポート
- Amplify コンソールを使用してウェブサイトをカスタムドメインに簡単に接続する
- 独自のカスタム SSL 証明書の持ち込み
- 組み込みアクセスログと CloudWatch メトリクスを使用してウェブサイトをモニタリングする
- ウェブサイトのパスワード保護を設定する
- Amplify コンソールでリダイレクトと書き換えルールを作成する

デプロイプロセスは、Amplify コンソール、 AWS CLI、または AWS SDKs から開始できます。Amplify にデプロイできるのは、ご自分のアカウントにある Amazon S3 汎用バケットからのみです。Amplify はクロスアカウント S3 バケットアクセスをサポートしていません。

Amazon S3 汎用バケットから Amplify ホスティングにアプリケーションをデプロイする場合、 AWS 料金は Amplify の料金モデルに基づいています。詳細については、「AWS Amplify 料金」を参照してください。

> **Important**
>
> Amplify ホスティングは、Amazon S3 AWS リージョン が利用可能なすべての で利用できるわけではありません。静的ウェブサイトを Amplify ホスティングにデプロイするには、ユーザーのウェブサイトを有する Amazon S3 汎用バケットが Amplify が利用可能なリージョンにある必要があります。Amplify が利用可能なリージョンのリストについては、「Amazon Web Services 全般のリファレンス」の「Amplify endpoints」を参照してください。

Amazon S3 から Amplify ホスティングに静的ウェブサイトをデプロイおよび更新する方法については、以下のトピックを参照してください。

トピック



# Amplify コンソールを使用して S3 から静的ウェブサイトをデプロイする

次の手順に従い、Amplify コンソールを使用して Amazon S3 汎用バケットから新しい静的ウェブサイトをデプロイします。

Amplify コンソールを使用して Amazon S3 汎用バケットから静的ウェブサイトをデプロイするには

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/amplify/ で Amplify コンソールを開きます。
2. [すべてのアプリ] ページで、[アプリの新規作成] を選択します。
3. [Amplify で構築を開始する] ページで、[Git なしでデプロイ] を選択します。
4. [次へ] を選択します。
5. [手動デプロイを開始] ページで、以下を実行します。
   a. [アプリ名] にお客様のアプリの名前を入力します。
   b. [ブランチ名] には、デプロイするブランチの名前を入力します。
6. [方法] で [Amazon S3] を選択します。
7. [ホストするオブジェクトの S3 の場所] については、[参照] を選択します。使用する Amazon S3 汎用バケットを選択し、[プレフィックスを選択] を選択します。
8. [保存してデプロイ] を選択します。

# AWS SDKs S3を使用して から静的ウェブサイトをデプロイするバケットポリシーの作成

AWS SDKs を使用して、Amazon S3 から Amplify ホスティングに静的ウェブサイトをデプロイできます。SDK を使用してウェブサイトをデプロイする場合は、S3 バケット内のオブジェクトを取得す

るためのアクセス許可を Amplify ホスティングに付与する独自のバケットポリシーを作成する必要があります。

バケットポリシーの作成方法の詳細については、「Amazon Simple Storage Service ユーザーガイド」の「Amazon S3 のバケットポリシー」を参照してください。

次のバケットポリシーの例では、指定された、Amplify アプリケーション ID、ブランチのバケットを一覧表示し AWS アカウント、バケットオブジェクトを取得するアクセス許可を Amplify ホスティングに付与します。

この例を使用するには:

- *amzn-s3-demo-website-bucket/prefix* をウェブサイトのバケットとプレフィックスの名前に置き換えます。
- *111122223333* を ID AWS アカウント に置き換えます。
- *region-id* を、Amplify アプリケーション AWS リージョン がある などに置き換えます**us-east-1**。
- *app_id* を Amplify アプリケーション ID に置き換えます。この情報は、Amplify コンソールで入手できます。
- *branch_name* をお使いのブランチ名に置き換えます。

> **Note**
>
> バケットポリシーでは、`aws:SourceArn` が URL エンコード (パーセントエンコード) ブランチ ARN である必要があります。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAmplifyToListPrefix_appid_branch_prefix_",
            "Effect": "Allow",
            "Principal": {
                "Service": "amplify.amazonaws.com"
            },
```

```
            "Action": "s3:ListBucket",
            "Resource": "arn:aws:s3:::amzn-s3-demo-website-bucket/prefix/*",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333",
                    "aws:SourceArn": "arn%3Aaws%3Aamplify%3Aregion-
id%3A111122223333%3Aapps%2Fapp_id%2Fbranches%2Fbranch_name",
                    "s3:prefix": ""
                }
            }
        },
        {
            "Sid": "AllowAmplifyToReadPrefix__appid_branch_prefix_",
            "Effect": "Allow",
            "Principal": {
                "Service": "amplify.amazonaws.com"
            },
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::amzn-s3-demo-website-bucket/prefix/*",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333",
                    "aws:SourceArn": "arn%3Aaws%3Aamplify%3Aregion-
id%3A111122223333%3Aapps%2Fapp_id%2Fbranches%2Fbranch_name"
                }
            }
        },
        {
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:*",
            "Resource": "arn:aws:s3:::amzn-s3-demo-website-bucket/*",
            "Condition": {
                "Bool": {
                    "aws:SecureTransport": "false"
                }
            }
        }
    ]
}
```

# S3 バケットから Amplify にデプロイされた静的ウェブサイトの更新

Amplify でホストされている汎用 S3 バケットにある静的ウェブサイト用オブジェクトのいずれかを更新する場合、その変更を有効にするには、アプリケーションを Amplify ホスティングに再デプロイする必要があります。Amplify ホスティングは、S3 バケットへの変更を自動的に検出しません。AWS Command Line Interface (CLI) を使用してウェブサイトを更新することをお勧めします。

更新を S3 に同期する

ウェブサイトのプロジェクトファイルを変更したら、次の S3 sync コマンドを使用して、ローカルソースディレクトリに加えた変更をターゲットの Amazon S3 汎用バケットと同期します。この例では、*<source>* をローカルディレクトリの名前に置き換え、*<target>* を Amazon S3 バケットの名前に置き換えます。

```
aws s3 sync <source> <target>
```

ウェブサイトを Amplify ホスティングに再デプロイする

次の amplify start-deployment コマンドを使用して、Amazon S3 バケットの更新されたアプリケーションを Amplify ホスティングに再デプロイします。この例では、*<app_id>* を Amplify アプリケーションの ID、*<branch_name>* をブランチの名前、*s3://amzn-s3-demo-website-bucket/prefix* を S3 バケットとプレフィックスに置き換えます。

```
aws amplify start-deployment --app-id <app_id> --branch-name <branch_name> --source-
url s3://amzn-s3-demo-website-bucket/prefix --source-url-type BUCKET_PREFIX
```

# .zip ファイルの代わりにバケットとプレフィックスを使用するように S3 デプロイを更新する

Amazon S3 汎用バケットの .zip ファイルから Amplify ホスティングにデプロイされた既存の静的ウェブサイトがある場合は、アプリケーションデプロイを更新して、ホストするオブジェクトを含むバケット名とプレフィックスを使用できます。このタイプのデプロイでは、ビルド出力の zip 圧縮コンテンツを含むバケットに別のファイルをアップロードする必要はありません。

静的ウェブサイトを .zip ファイルからバケットコンテンツに移行するには

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/amplify/ で Amplify コンソールを開きます。
2. [すべてのアプリ] ページで、手動でデプロイされたアプリケーションの名前を選択し、.zip ファイルの使用からアプリケーションファイルの使用に直接移行します。
3. アプリケーションの [概要] ページで、[アップデートをデプロイ] を選択します。
4. [アップデートをデプロイ] ページで、[方法] について [Amazon S3] を選択します。
5. [ホストするオブジェクトの S3 の場所] については、[参照] を選択します。使用するバケットを選択し、[プレフィックスを選択] を選択します。
6. [保存してデプロイ] を選択します。

# Git リポジトリなしでアプリケーションを Amplify にデプロイする

手動デプロイを使用すると、Git プロバイダーに接続しなくても、Amplify ホスティングでウェブアプリケーションを公開できます。デスクトップから zip フォルダをドラッグアンドドロップすると、数秒でサイトをホストできます。または、Amazon S3 バケット内のアセットを参照するか、ファイルが保存されている場所へのパブリック URL を指定することもできます。

> **Note**
>
> Amazon S3 コピーオペレーションの制約により、手動デプロイの最大 .zip ファイルサイズ制限は 5GB です。いずれかのビルドアーティファクトがこのサイズを超える場合は、小さいアーカイブに分割するか、別のデプロイ方法を使用することを検討してください。

Amazon S3 では、新しいアセットがアップロードされるたびにサイトを更新する AWS Lambda トリガーを設定することもできます。このシナリオの設定の詳細については、ブログ投稿「Amazon S3、Dropbox、またはデスクトップに保存されているファイルを AWS Amplify コンソールにデプロイする」を参照してください。

Amplify ホスティングは、サーバーサイドレンダリング (SSR) されたアプリの手動デプロイをサポートしていません。詳細については、「Amplify ホスティングでサーバーサイドレンダリングされたアプリのデプロイ」を参照してください。

## ドラッグアンドドロップによる手動デプロイ

ドラッグアンドドロップを使用してアプリを手動でデプロイするには

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
2. 右上隅にある [アプリの新規作成] を選択します。
3. [Amplify で構築を開始する] ページで、[Git なしでデプロイ] を選択します。その後、[Next] を選択します。
4. [手動デプロイを開始する] ページの [アプリ名] に、お客様のアプリの名前を入力します。
5. [ブランチ名] には、**`development`** や **`production`** などのわかりやすい名前を入力します。
6. [メソッド] には [ドラッグアンドドロップ] を選択します。

7. デスクトップからドロップゾーンにフォルダーをドラッグアンドドロップするか、[.zip フォルダーを選択] を使用してコンピューターからファイルを選択します。ドラッグアンドドロップまたは選択するファイルは、ビルド出力の内容を含む 圧縮フォルダである必要があります。
8. [保存してデプロイ] を選択します。

# Amazon S3 または URL の手動デプロイ

> **Note**
>
> S3 から静的ウェブサイトをデプロイしている場合、次の手順では、ビルド出力の内容を含む圧縮フォルダを S3 バケットにアップロードする必要があります。バケット名とプレフィックスを使用して、S3 から静的ウェブサイトを直接デプロイすることをお勧めします。この簡易プロセスの詳細については、「Amazon S3 バケットから Amplify への静的ウェブサイトのデプロイ」を参照してください。

Amazon S3 またはパブリック URL からアプリを手動でデプロイするには

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
2. 右上隅にある [アプリの新規作成] を選択します。
3. [Amplify で構築を開始する] ページで、[Git なしでデプロイ] を選択します。その後、[Next] を選択します。
4. [手動デプロイを開始する] ページの [アプリ名] に、お客様のアプリの名前を入力します。
5. [ブランチ名] には、**`development`** や **`production`** などのわかりやすい名前を入力します。
6. [メソッド] には、[Amazon S3] または [任意の URL] を選択します。
7. ファイルをアップロードする手順は、アップロード方法によって異なります。
   - Amazon S3
     a. [S3 location of objects to host] には、[S3 を参照する] を選択します。次に、リストから Amazon S3 バケットの名前を選択します。選択したバケット上でアクセスコントロールリスト (ACL) を有効にする必要があります。詳細については、「手動デプロイの Amazon S3 バケットアクセスのトラブルシューティング」を参照してください。
     b. デプロイする .zip ファイルの名前を選択します。
     c. [プレフィックスの選択] を選択します。
   - 任意の URL

- [リソース URL] には、デプロイする .zip ファイルへの URL を入力します。

8. [保存してデプロイ] を選択します。

> **Note**
>
> 圧縮フォルダーを作成するときは、最上位のフォルダーではなく、ビルド出力の内容を必ず圧縮してください。たとえば、ビルド出力から「build」または「public」という名前のフォルダーが生成される場合は、まずそのフォルダーに移動し、内容をすべて選択して、そこから圧縮します。これを行わないと、サイトのルートディレクトリが正しく初期化されないため、「Access Denied」(アクセスが拒否されました) エラーが表示されます。

## 手動デプロイの Amazon S3 バケットアクセスのトラブルシューティング

Amazon S3 バケットを作成するときは、Amazon S3 オブジェクト所有権設定を使用して、バケット上でアクセスコントロールリスト (ACL) の有効/無効を制御するために使用できます。Amazon S3 バケットから Amplify に手動でアプリをデプロイするには、バケット上でACLを有効にする必要があります。

Amazon S3 バケットからデプロイするときに `AccessControlList` エラーが発生した場合、バケットは ACL を無効にして作成されているため、Amazon S3 コンソールで有効にする必要があります。手順については、「Amazon Simple Storage Service ユーザーガイド」の「既存のバケットにオブジェクト所有権を設定する」を参照してください。

# Amplify アプリケーションのビルド設定の管理

Amplify デプロイのビルド設定と設定をカスタマイズできます。アプリケーションをデプロイすると、Amplify は、Git リポジトリ内のアプリの ファイルを検査することで、フロントエンドフレームワークと関連するビルド設定を自動的に検出します。アプリケーションのビルド仕様 (buildspec) のビルド設定をカスタマイズして、環境変数の追加、ビルドコマンドの実行、ビルドの依存関係の指定を行うことができます。

Amplify のデフォルトのビルドイメージには複数のパッケージと依存関係がプリインストールされていますが、ライブパッケージ更新機能を使用して特定のバージョンを指定するか、常に最新バージョンがインストールされていることを確認することもできます。Amplifyのデフォルトのコンテナを使用して、ビルド中に特定の依存関係をインストールするのに長い時間がかかる場合は、独自の Docker イメージを作成してビルド中に参照することができます。ビルドインスタンスのサイズをカスタマイズして、必要な CPU、メモリ、ディスク容量リソースをアプリケーションデプロイに提供することもできます。

ビルドは、Git リポジトリへのコミットごとに、および新しいデプロイごとに自動的に開始されます。受信ウェブフック機能を設定して、Git リポジトリへのコミットなしでビルドを開始できます。

ビルド通知機能を使用すると、ビルドの成功と失敗に関する情報をチームメンバーと共有できます。

トピック



## Amplify アプリケーションのビルド設定の構成

アプリケーションをデプロイすると、Amplify は、Git リポジトリ内のアプリの `package.json` ファイルを検査することで、フロントエンドフレームワークと関連するビルド設定を自動的に検出します。アプリのビルド設定を保存するには、次のオプションがあります。

- Amplify コンソールでビルド設定を保存する – Amplify コンソールを使用してビルド設定を自動検出および保存することで、Amplify コンソールでアクセスできるようになります。リポジトリに

amplify.yml ファイルが保存されている場合を除き、Amplify はこれらの設定をすべてのブランチに適用します。

- ビルド設定をリポジトリに保存する - amplify.yml ファイルをダウンロードして、リポジトリのルートに追加します。

> **Note**
>
> [ビルド設定] は、アプリが継続的デプロイ用に設定され、git リポジトリに接続されている場合にのみ、Amplify コンソールの [ホスティング] メニューに表示されます。この種類のデプロイの手順については、「概要」を参照してください。

## ビルド仕様に関するリファレンス

Amplify のアプリケーションのビルド仕様は、Amplify がビルドの実行に使用する YAML 設定とビルドコマンドを集めたものです。以下のリストでは、これらの設定とその使用方法について説明しています。

バージョン

Amplify YAML バージョン番号。

appRoot

このアプリケーションが置かれているリポジトリ内のパス。複数のアプリケーションが定義されていない限り無視されます。

env

環境変数をこのセクションに追加します。また、環境変数はコンソールを使用して追加することもできます。

backend

Amplify CLI コマンドを実行して、バックエンドのプロビジョン、Lambda 関数の更新、または継続的なデプロイの一環としての GraphQL スキーマの更新を行います。

frontend

フロントエンドのビルドコマンドを実行します。

test

テストフェーズ中にコマンドを実行します。アプリにテストを追加する方法をご覧ください。

ビルドフェーズ

フロントエンド、バックエンド、およびテストには、ビルドの各シーケンス中に実行されるコマンドを表す 3 つのフェーズがあります。

- preBuild - preBuild スクリプトが、実際のビルドの開始前、Amplify が依存関係をインストールした後に実行されます。
- build - お客様のビルドコマンド。
- postBuild - post-build スクリプトは、ビルドが終了し、Amplify が必要なすべてのアーティファクトを出力ディレクトリにコピーした後に実行されます。

buildpath

ビルドの実行に使用するパス。Amplify はこのパスを使用してビルドアーティファクトを見つけます。パスを指定しない場合、Amplify は、モノレポのアプリルートを使用します。(例: apps/app)

artifacts>base-directory

ビルドアーティファクトが存在するディレクトリ。

artifacts>files

デプロイするアーティファクトからファイルを指定します。すべてのファイルを含めるには **/* を入力します。

cache

node_modules フォルダなどのビルド時の依存関係を指定します。最初のビルドでは、ここで提供されるパスがキャッシュされます。以降のビルドでは、Amplify はコマンドを実行する前にキャッシュを同じパスに復元します。

Amplify は、提供されたすべてのキャッシュパスをプロジェクトルートに対する相対パスと見なします。ただし、Amplify はプロジェクトルート外へのトラバースを許可しません。たとえば、絶対パスを指定すると、ビルドはエラーなしで成功しますが、パスはキャッシュされません。

## ビルド仕様 YAML 構文のリファレンス

以下のビルド仕様例は、基本的な YAML 構文を示しています。

```
version: 1
env:
  variables:
    key: value
backend:
  phases:
    preBuild:
      commands:
        - *enter command*
    build:
      commands:
        - *enter command*
    postBuild:
        commands:
        - *enter command*
frontend:
  buildpath:
  phases:
    preBuild:
      commands:
        - cd react-app
        - npm ci
    build:
      commands:
        - npm run build
  artifacts:
    files:
        - location
        - location
    discard-paths: yes
    baseDirectory: location
  cache:
    paths:
        - path # A cache path relative to the project root
        - path # Traversing outside of the project root is not allowed
test:
  phases:
    preTest:
      commands:
        - *enter command*
    test:
      commands:
        - *enter command*
```

```
    postTest:
      commands:
        - *enter command*
  artifacts:
    files:
        - location
        - location
    configFilePath: *location*
    baseDirectory: *location*
```

## ビルド仕様の編集

Amplify コンソールでビルド仕様 (buildspec) を編集することで、アプリケーションのビルド設定をカスタマイズできます。ビルド設定は、アプリ内のすべてのブランチに適用されます。ただし、Git リポジトリに `amplify.yml` ファイルが保存されたブランチを除きます。

Amplify コンソールでビルド設定を編集するには

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
2. ビルド設定を編集するアプリを選択します。
3. ナビゲーションペインで、[ホスティング] を選択し、[ビルド設定] を選択します。
4. [ビルド設定] ページの [ビルド仕様の追加] セクションで、[編集] を選択します。
5. [ビルド仕様の編集] ウィンドウで、更新を入力します。
6. [保存] を選択します。

以下のトピックの例を使用して、特定のシナリオのビルド設定を更新できます。

トピック

- スクリプトを使用したブランチ固有のビルド設定の設定
- サブフォルダーに移動するコマンドの設定
- Gen 1 アプリのフロントエンドでのバックエンドのデプロイ
- 出力フォルダの設定
- ビルドの一部としてパッケージをインストールする
- プライベート npm レジストリを使用する
- OS パッケージのインストール

- ビルドごとのキー値のストレージの設定
- コミットのビルドのスキップ
- コミットごとの自動ビルドのオフ
- 差分ベースのフロントエンドビルドとデプロイの設定
- Gen 1 アプリケーションの diff ベースのバックエンドビルドの設定

## スクリプトを使用したブランチ固有のビルド設定の設定

bash シェルスクリプトを使用してブランチ固有のビルドを設定できます。たとえば、次のスクリプトは、システム環境変数 $AWS_BRANCH を使用して、ブランチ名が main の場合はあるコマンドセットを実行し、ブランチ名が dev の場合は別のコマンドセットを実行します。

```
frontend:
  phases:
    build:
      commands:
        - if [ "${AWS_BRANCH}" = "main" ]; then echo "main branch"; fi
        - if [ "${AWS_BRANCH}" = "dev" ]; then echo "dev branch"; fi
```

## サブフォルダーに移動するコマンドの設定

モノレポの場合、ユーザーは、cd を実行してフォルダに移動して、ビルドを実行できる必要があります。cd コマンドを実行すると、コマンドがビルドのすべてのステージに適用されるため、各フェーズでコマンドを再度実行する必要はありません。

```
version: 1
env:
  variables:
    key: value
frontend:
  phases:
    preBuild:
      commands:
        - cd react-app
        - npm ci
    build:
      commands:
        - npm run build
```

## Gen 1 アプリのフロントエンドでのバックエンドのデプロイ

> **Note**
>
> このセクションは、Amplify Gen 1 アプリケーションにのみ対応します。Gen 1 バックエンドは、Amplify Studio と Amplify コマンドラインインターフェイス (CLI) を使用して作成されます。

amplifyPush コマンドは、バックエンドのデプロイを支援するヘルパースクリプトです。以下のビルド設定によって、現在のブランチにデプロイする上で適切なバックエンド環境が自動的に判別されます。

```
version: 1
env:
  variables:
    key: value
backend:
  phases:
    build:
      commands:
        - amplifyPush --simple
```

### 出力フォルダの設定

次のビルド設定は、出力ディレクトリをパブリックフォルダに設定します。

```
frontend:
  phases:
    commands:
      build:
        - yarn run build
  artifacts:
    baseDirectory: public
```

### ビルドの一部としてパッケージをインストールする

npm または yarn コマンドを使って、ビルド中にパッケージをインストールすることができます。

```
frontend:
```

```
  phases:
    build:
      commands:
        - npm install -g <package>
        - <package> deploy
        - yarn run build
  artifacts:
    baseDirectory: public
```

## プライベート npm レジストリを使用する

プライベートレジストリへのリファレンスは、ビルド設定で追加するか、環境変数として追加することができます。

```
build:
  phases:
    preBuild:
      commands:
        - npm config set <key> <value>
        - npm config set registry https://registry.npmjs.org
        - npm config set always-auth true
        - npm config set email hello@amplifyapp.com
        - yarn install
```

## OS パッケージのインストール

Amplify の AL2023 イメージは、amplify という名前の非特権ユーザーでコードを実行します。Amplify は、Linux sudo コマンドを使用して OS コマンドを実行する権限をこのユーザーに付与します。依存関係が欠落しているため OS パッケージをインストールする場合は、yum や rpm などのコマンドを sudo とともに使用できます。

次のビルドセクションの例は、sudo コマンドを使用して OS パッケージをインストールするための構文を示しています。

```
build:
  phases:
    preBuild:
      commands:
        - sudo yum install -y <package>
```

## ビルドごとのキー値のストレージの設定

envCache はビルド時に key-value ストレージを提供します。envCache に保存された値はビルド中にのみ変更でき、次回のビルドで再利用できます。envCache を使用することで、デプロイされた環境に情報を保存し、それを連続したビルドでビルドコンテナを利用できるようにします。envCache に保存された値とは異なり、ビルド中に環境変数を変更しても、将来のビルドには反映されません。

使用例:

```
envCache --set <key> <value>
envCache --get <key>
```

## コミットのビルドのスキップ

特定のコミットの自動ビルドをスキップするには、コミットメッセージの末尾に [skip-cd] というテキストを含めます。

## コミットごとの自動ビルドのオフ

コードのコミットごとに自動ビルドをオフにするように Amplify を設定できます。セットアップするには、[アプリの設定]、[ブランチ設定] の順に選択し、接続されたすべてのブランチが一覧になっている [ブランチ] セクションまでスクロールします。ブランチを選択してから、[アクション]、[自動構築を無効化] を選択します。そのブランチに新しくコミットしても、新しいビルドは開始されません。

## 差分ベースのフロントエンドビルドとデプロイの設定

差分ベースのフロントエンドビルドを使用するように Amplify を設定できます。有効にすると、Amplify は各ビルドの開始時に、デフォルトで自分の appRoot フォルダー、または /src/ フォルダーの差分を実行しようとします。Amplify で差分が見つからなかった場合、フロントエンドのビルド、テスト (設定されている場合)、およびデプロイのステップはスキップされ、ホストされているアプリは更新されません。

差分ベースのフロントエンドビルドとデプロイを設定するには

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
2. アプリを選択して、差分ベースのフロントエンドビルドとデプロイを設定します。
3. ナビゲーションペインで、[ホスティング]、[環境変数] を選択します。

4. 「環境変数」セクションで、[変数の管理] を選択します。
5. 環境変数を設定する手順は、差分ベースのフロントエンドビルドとデプロイを有効にするか無効にするかによって異なります。
   - 差分ベースのフロントエンドビルドとデプロイを有効化するには
     a. 「変数の管理」セクションの [変数] に、AMPLIFY_DIFF_DEPLOY と入力します。
     b. [値] に「true」と入力します。
   - 差分ベースのフロントエンドビルドとデプロイを無効化するには
     - 次のいずれかを行います。
       - 「変数の管理」セクションで、AMPLIFY_DIFF_DEPLOY を探します。[値] に「false」と入力します。
       - AMPLIFY_DIFF_DEPLOY 環境変数を削除します。
6. [保存] を選択します。

オプションで、デフォルトパスをリポジトリのルートからの相対パス (distなど) でオーバーライドするように環境変数 AMPLIFY_DIFF_DEPLOY_ROOT を設定できます。

## Gen 1 アプリケーションの diff ベースのバックエンドビルドの設定

> **Note**
> このセクションは、Amplify Gen 1 アプリケーションにのみ対応します。Gen 1 バックエンドは、Amplify Studio と Amplify コマンドラインインターフェイス (CLI) を使用して作成されます。

環境変数 AMPLIFY_DIFF_BACKEND を使用して、差分ベースのバックエンドビルドを使用するように Amplify ホスティングを設定できます。差分ベースのバックエンドビルドを有効にすると、各ビルドの開始時に、Amplify はリポジトリ内の amplify フォルダーで差分を実行しようとします。Amplify が差分を見つけられない場合、バックエンドのビルドステップをスキップし、バックエンドリソースを更新しません。プロジェクトのリポジトリに amplify フォルダがない場合、Amplify は環境変数 AMPLIFY_DIFF_BACKEND の値を無視します。

現在、バックエンドフェーズのビルド設定でカスタムコマンドが指定されている場合、条件付きバックエンドビルドは機能しません。これらのカスタムコマンドを実行したい場合は、アプリの amplify.yml ファイルにあるビルド設定のフロントエンドフェーズに移動する必要があります。

差分ベースのバックエンドビルドを設定するには

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
2. 差分ベースのバックエンドビルドを設定するアプリを選択します。
3. ナビゲーションペインで、[ホスティング]、[環境変数] を選択します。
4. 「環境変数」セクションで、[変数の管理] を選択します。
5. 環境変数を設定する手順は、差分ベースのバックエンドビルドを有効化 / 無効化するかによって異なります。
   - 差分ベースのバックエンドビルドを有効にするには
     a. 「変数の管理」セクションの [変数] に、AMPLIFY_DIFF_BACKEND と入力します。
     b. [値] に「true」と入力します。
   - 差分ベースのバックエンドビルドを無効にするには
     - 次のいずれかを行います。
       - 「変数の管理」セクションで、AMPLIFY_DIFF_BACKEND を探します。[値] に「false」と入力します。
       - AMPLIFY_DIFF_BACKEND 環境変数を削除します。
6. [保存] を選択します。

## モノレポビルド設定の構成

複数のプロジェクトやマイクロサービスを単一のリポジトリに格納することをモノレポと呼びます。Amplify ホスティングを使用すると、複数のビルド構成やブランチ構成を作成しなくても、モノレポにアプリケーションをデプロイできます。

Amplify は、一般的なモノレポのアプリだけでなく、npm ワークスペース、pnpm ワークスペース、Yarn ワークスペース、Nx、および Turborepo を使用して作成されたモノレポのアプリもサポートします。アプリをデプロイすると、Amplify は使用しているモノレポビルドツールを自動的に検出します。Amplify は、npmワークスペース、Yarnワークスペース、またはNxのアプリにビルド設定を自動的に適用します。Turborepo と pnpm アプリには、追加設定が必要です。詳細については、「Turborepo アプリと pnpm モノレポアプリの設定」を参照してください。

モノレポのビルド設定は Amplify コンソールに保存することも、amplify.yml ファイルをダウンロードしてリポジトリのルートに追加することもできます。Amplify は、リポジトリで amplify.yml ファイルを見つけない限り、コンソールに保存された設定をすべてのブランチに適用

します。amplify.yml ファイルが存在する場合、その設定は Amplify コンソールに保存されているビルド設定よりも優先されます。

## モノレポビルド仕様 YAML 構文のリファレンス

モノレポビルド仕様の YAML 構文は、単一のアプリケーションを含むリポジトリの YAML 構文とは異なります。モノレポでは、各プロジェクトをアプリケーションのリストで宣言します。モノレポビルド仕様で宣言するアプリケーションごとに、以下の追加 appRoot キーを指定する必要があります。

appRoot

アプリケーションが起動するリポジトリ内のルート。キーは必ず存在する必要があり、環境変数 AMPLIFY_MONOREPO_APP_ROOT と同じ値となります。この環境変数を設定する手順については、AMPLIFY_MONOREPO_APP_ROOT 環境変数の設定 を参照してください。

以下のモノレポビルド仕様の例は、同じリポジトリで複数の Amplify アプリケーションを宣言する方法を示しています。2 つのアプリ (react-app と angular-app) は applications リストで宣言されます。各アプリの appRoot キーは、そのアプリケーションがリポジトリの apps ルートフォルダーにあることを示しています。

この buildpath 属性は、モノレポプロジェクトルートからアプリを実行してビルドするように / に設定されます。baseDirectory 属性は buildpath の相対パスです。

モノレポビルド仕様 YAML 構文

```
version: 1
applications:
  - appRoot: apps/react-app
    env:
      variables:
        key: value
    backend:
      phases:
        preBuild:
          commands:
            - *enter command*
        build:
          commands:
            - *enter command*
        postBuild:
```

```
            commands:
            - *enter command*
    frontend:
      buildPath: / # Run install and build from the monorepo project root
      phases:
        preBuild:
          commands:
            - *enter command*
            - *enter command*
        build:
          commands:
            - *enter command*
      artifacts:
        files:
            - location
            - location
        discard-paths: yes
        baseDirectory: location
      cache:
        paths:
            - path
            - path
    test:
      phases:
        preTest:
          commands:
            - *enter command*
        test:
          commands:
            - *enter command*
        postTest:
          commands:
            - *enter command*
      artifacts:
        files:
            - location
            - location
        configFilePath: *location*
        baseDirectory: *location*
  - appRoot: apps/angular-app
    env:
      variables:
        key: value
    backend:
```

```
      phases:
        preBuild:
          commands:
            - *enter command*
        build:
          commands:
            - *enter command*
        postBuild:
            commands:
            - *enter command*
    frontend:
      phases:
        preBuild:
          commands:
            - *enter command*
            - *enter command*
        build:
          commands:
            - *enter command*
      artifacts:
        files:
            - location
            - location
        discard-paths: yes
        baseDirectory: location
      cache:
        paths:
            - path
            - path
    test:
      phases:
        preTest:
          commands:
            - *enter command*
        test:
          commands:
            - *enter command*
        postTest:
          commands:
            - *enter command*
      artifacts:
        files:
            - location
            - location
```

```
        configFilePath: *location*
        baseDirectory: *location*
```

次のビルド仕様例を使用するアプリは、プロジェクトルートの下にビルドされ、ビルドアーティファクトは /packages/nextjs-app/.next に配置されます。

```
applications:
  - frontend:
      buildPath: '/'  # run install and build from monorepo project root
      phases:
        preBuild:
          commands:
            - npm install
        build:
          commands:
            - npm run build --workspace=nextjs-app
      artifacts:
        baseDirectory: packages/nextjs-app/.next
        files:
          - '**/*'
      cache:
        paths:
          - node_modules/**/*
    appRoot: packages/nextjs-app
```

## AMPLIFY_MONOREPO_APP_ROOT 環境変数の設定

モノレポに保存されたアプリをデプロイする場合、アプリの環境変数 AMPLIFY_MONOREPO_APP_ROOT は、リポジトリのルートを基準にしたアプリルートのパスと同じ値でなければなりません。たとえば、ExampleMonorepo という名前のモノレポに、app1、app2 を含む apps という名前のルートフォルダがあり、app3 は次のようなディレクトリ構造を持つとします。

```
ExampleMonorepo
  apps
    app1
    app2
    app3
```

この例では、app1 の環境変数 AMPLIFY_MONOREPO_APP_ROOT の値は apps/app1 です。

Amplify コンソールを使用してモノレポアプリをデプロイすると、コンソールはアプリのルートへのパスに指定した値を使用して環境変数 AMPLIFY_MONOREPO_APP_ROOT を自動的に設定します。ただし、モノレポアプリが既に Amplify に存在するか、 を使用してデプロイされている場合は AWS CloudFormation、Amplify コンソールのAMPLIFY_MONOREPO_APP_ROOT環境変数セクションで環境変数を手動で設定する必要があります。

デプロイ時に AMPLIFY_MONOREPO_APP_ROOT 環境変数を自動的に設定する

以下の手順は、Amplify コンソールでモノレポアプリをデプロイする方法を示しています。Amplify は、コンソールで指定したアプリのルートフォルダーを使用して環境変数 AMPLIFY_MONOREPO_APP_ROOT を自動的に設定します。

Amplify コンソールでモノレポアプリをデプロイするには

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
2. 右上隅の [アプリの新規作成] を選択します。
3. [Amplify で構築を開始する] ページで、お使いの Git プロバイダーを選択し、[次へ] を選択します。
4. [リポジトリブランチを追加] ページで、次の操作を行います。
   a. リストからリポジトリの名前を選択します。
   b. 使用するブランチの名前を選択します。
   c. [My アプリがモノレポです] を選択する
   d. モノレポにアプリへのパスを入力します (例:**apps/app1**)。
   e. [次へ] を選択します。
5. [アプリ設定] ページでは、デフォルト設定を使用するか、アプリのビルド設定をカスタマイズできます。[環境変数] セクションで、Amplify はステップ 4d で指定したパスに AMPLIFY_MONOREPO_APP_ROOT を設定します。
6. [次へ] を選択します。
7. [確認]ページ で、[保存してデプロイ]を選択します。

既存のアプリケーションの AMPLIFY_MONOREPO_APP_ROOT 環境変数を設定する

以下の手順を使用して、Amplify にすでにデプロイされているアプリ、または CloudFormation を使用して作成されたアプリの環境変数 AMPLIFY_MONOREPO_APP_ROOT を手動で設定します。

既存のアプリケーションの AMPLIFY_MONOREPO_APP_ROOT 環境変数を設定するには

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
2. 環境変数を設定するアプリの名前を選択します。
3. ナビゲーションペインで、[ホスティング]、[環境変数] の順で選択します。
4. 「環境変数」ページで、[変数の管理] を選択します。
5. [変数の管理]セクションで、次の操作を行います。
   a. [新規追加] を選択します。
   b. [変数] にはキー AMPLIFY_MONOREPO_APP_ROOT を入力します。
   c. [値] には、アプリへのパスを入力します (例:**apps/app1**)。
   d. [ブランチ] の場合、Amplify はデフォルトで環境変数をすべてのブランチに適用します。
6. [保存] を選択します。

## Turborepo アプリと pnpm モノレポアプリの設定

Turborepo と pnpm ワークスペースのモノレポビルドツールは .npmrc ファイルから構成情報を取得します。これらのツールのいずれかで作成したモノレポアプリをデプロイする場合、プロジェクトのルートディレクトリに .npmrc ファイルを置く必要があります。

この .npmrc ファイルで、ノードパッケージをインストールするためのリンカーを hoisted に設定します。以下の行をファイルにコピーできます。

```
node-linker=hoisted
```

.npmrc ファイルと設定について詳しくは、「pnpm ドキュメント」の「pnpm .npmrc」を参照してください。

pnpm は Amplify のデフォルトビルドコンテナには含まれていません。pnpm ワークスペースと Turborepo アプリの場合、アプリのビルド設定の preBuild 段階で pnpm をインストールするコマンドを追加する必要があります。

次のビルド仕様からの抜粋例は、pnpm をインストールするコマンドを含む preBuild フェーズを示しています。

```
version: 1
applications:
  - frontend:
```

```
    phases:
      preBuild:
        commands:
          - npm install -g pnpm
```

# ビルドイメージのカスタマイズ

カスタムビルドイメージを使用すると、カスタマイズされた Amplify アプリのビルド環境を提供できます。Amplifyのデフォルトのコンテナを使用して、ビルド中に特定の依存関係をインストールするのに長い時間がかかる場合は、独自の Docker イメージを作成してビルド中に参照することができます。イメージは Docker Hub、またはパブリックの Amazon Elastic Container Registry Public でホストできます。

カスタムビルドイメージを Amplify ビルドイメージとして動作させるには、次の要件を満たしている必要があります。

カスタムビルドイメージの要件

1. x86-64 アーキテクチャ用にコンパイルされた Amazon Linux などの GNU C ライブラリ (glibc) をサポートする Linux 配信。
2. cURL: カスタムイメージを起動すると、ビルドランナーがコンテナにダウンロードされるため、cURL が必要です。この依存関係が欠落している場合は、build-runner で出力を生成できないため、ビルドは何も出力することなく即座に失敗します。
3. Git: Git リポジトリのクローンを作成するには、Git をイメージにインストールする必要があります。この依存関係が欠落している場合、「リポジトリのクローンを作成する」ステップは失敗します。
4. OpenSSH: リポジトリのクローンを安全に作成するには、OpenSSH を使用して、ビルド中に一時的に SSH キーを設定する必要があります。OpenSSH パッケージは、ビルドランナーがこれを行うために必要なコマンドを提供します。
5. bash と Bourne シェル: これらの 2 つのユーティリティは、ビルド時にコマンドを実行するために使用されます。これらがインストールされていない場合、ビルドが開始する前に失敗する可能性があります。
6. Node.JS+NPM: ビルドランナーは Node をインストールしません。代わりに、Node (ノード) と NPM がイメージにインストールされていることを前提としています。これは、NPM パッケージまたはノード固有のコマンドを必要とするビルドにのみ必要です。ただし、これらが存在している場合、Amplify ビルドランナーがこれらのツールを使用してビルドの実行を改善できるため、インストールすることを強くお勧めします。Amplify のパッケージオーバーライド機能は、Hugo

用にオーバーライドを設定する際に、NPM を使用して Hugo 拡張パッケージをインストールします。

次のパッケージは必須ではありませんが、インストールすることを強くお勧めします。

1. NVM (Node Version Manager): Node の異なるバージョンを処理する必要がある場合は、このバージョンマネージャーをインストールすることをお勧めします。オーバーライドを設定すると、Amplify のパッケージオーバーライド機能は NVM を使用して、各ビルドの前に Node.js のバージョンを変更します。
2. Wget: Amplify は、ビルドプロセス中に Wget ユーティリティを使用してファイルをダウンロードできます。カスタムイメージにインストールすることをお勧めします。
3. Tar: Amplify は、ビルドプロセス中に Tar ユーティリティを使用して、ダウンロードされたファイルを解凍できます。カスタムイメージにインストールすることをお勧めします。

## アプリのカスタムビルドイメージの設定

次の手順で、Amplify コンソールでアプリケーションのカスタムビルドイメージを設定します。

Amazon ECR でホストされているカスタムビルドイメージを設定するには

1. Docker イメージを使用して Amazon ECR パブリックリポジトリをセットアップするには、「Amazon ECR パブリックユーザーガイド」の「はじめに」を参照してください。
2. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
3. カスタムビルドイメージを設定したいアプリを選択します。
4. ナビゲーションペインで [ホスティング]、[ビルド設定] を選択します。
5. 「ビルド設定」 ページの「ビルドイメージ設定」セクションで、「編集」を選択します。
6. [ビルドイメージ設定の編集] のページで、[ビルドイメージ] メニューを展開し、[カスタムビルドイメージ] を選択します。
7. ステップ 1で作成した Amazon ECR パブリックリポジトリの名前を入力します。ビルドイメージはここでホストされます。例えば、リポジトリの名前が ecr-examplerepo の場合、「**`public.ecr.aws/xxxxxxxx/ecr-examplerepo`**」と入力します。
8. [保存] を選択します。

# ビルドイメージでの特定のパッケージバージョンと依存関係バージョンの使用

ライブパッケージのアップデートを使用すると、Amplify のデフォルトのビルドイメージで使用するパッケージと依存関係のバージョンを指定できます。デフォルトのビルドイメージには、いくつかのパッケージと依存関係がプリインストールされています (例: Hugo、Amplify CLI、Yarn)。ライブパッケージのアップデートを使用すると、これらの依存関係のバージョンを上書きして特定のバージョンを指定するか、常に最新バージョンがインストールされていることを確認できます。

ライブパッケージのアップデートが有効になっている場合は、ビルドが実行される前に、ビルドランナーは最初に指定された依存関係を更新 (またはダウングレード) します。これにより、依存関係の更新にかかる時間に比例してビルド時間が長くなりますが、同じバージョンの依存関係を使用してアプリをビルドできるというメリットがあります。

> **Warning**
>
> Node.js バージョンを [最新] に設定すると、ビルドが失敗します。代わりに、`18`、`21.5`、`v0.1.2` などの特定の Node.js バージョンを指定する必要があります。

ライブパッケージアップデートを設定するには

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
2. ライブパッケージアップデートを設定したいアプリを選択します。
3. ナビゲーションペインで [ホスティング]、[ビルド設定] を選択します。
4. 「ビルド設定」 ページの「ビルドイメージ設定」セクションで、「編集」を選択します。
5. [ビルドイメージ設定の編集] ページの [ライブパッケージのアップデート] リストで、[新規追加] を選択します。
6. [パッケージ] で、上書きする依存関係を選択します。
7. Version には、デフォルトを最新のままにするか、依存関係の特定のバージョンを入力します。最新を使用すると、依存関係は利用可能なバージョンに常にアップグレードされます。
8. [保存] を選択します。

# Amplify アプリケーションのビルドインスタンスの設定

Amplify ホスティングには、アプリケーションのビルドインスタンスに必要な CPU、メモリ、ディスク容量のリソースを提供する設定可能なビルドインスタンスサイズが用意されています。この機能のリリース前、Amplify は 8 GiB のメモリと 4 つの vCPU から成る固定サイズのビルドインスタンス構成を提供していました。

Amplify は、Standard、Large、XLarge という 3 つのビルドインスタンスタイプをサポートしています。インスタンスタイプを指定しなければ、Amplify はデフォルトの Standard インスタンスを使用します。Amplify コンソール、、または SDKs を使用して AWS CLI、アプリケーションのビルドインスタンスタイプを設定できます。

各ビルドインスタンスタイプのコストは、ビルド時間 (分) ごとに計算されます。料金の詳細については、「AWS Amplify の料金」を参照してください。

次の表に、各ビルドインスタンスタイプのコンピューティング仕様を示します。

| ビルドインスタンスタイプ | vCPU | メモリ | ディスク容量 |
|---|---|---|---|
| Standard | 4 vCPU | 8 GiB | 128 GB |
| Large | 8 vCPUs | 16 GiB | 128 GB |
| XLarge | 36 vCPUs | 72 GiB | 256 GB |

トピック



## ビルドインスタンスタイプについて

ビルドインスタンスタイプの設定はアプリケーションレベルで設定され、アプリケーションのすべてのブランチに拡張されます。ビルドインスタンスタイプには、次のキーの詳細が適用されます。

- アプリケーション用に設定したビルドインスタンスタイプは、自動作成されたブランチとプルリクエストのプレビューに自動的に適用されます。
- 同時ジョブサービスクォータは、 のすべてのビルドインスタンスタイプに適用されます AWS アカウント。たとえば、同時ジョブの制限が 5 の場合、 AWS アカウントのすべてのインスタンスタイプで最大 5 つのビルドを実行できます。
- 各ビルドインスタンスタイプのコストは、ビルド時間 (分) ごとに計算されます。ビルドインスタンスの割り当てプロセスでは、ビルドの開始前に追加のオーバーヘッド時間が必要になる場合があります。大規模なインスタンス、特に XLarge の場合、このオーバーヘッド時間が原因で、ビルドの開始前にビルドにレイテンシーが発生する可能性があります。ただし、オーバーヘッド時間ではなく、実際のビルド時間に対してのみ課金されます。

ビルドインスタンスタイプは、新しいアプリケーションを作成するときに設定することも、既存のアプリケーションでインスタンスタイプを更新することもできます。Amplify コンソールでこの設定を構成する手順については、「Amplify コンソールでのビルドインスタンスタイプの設定」を参照してください。SDK を使用してこの設定を更新することもできます。詳細については、「Amplify API リファレンス」の「CreateApp」および「UpdateApp API」を参照してください。

カスタマイズ可能なビルドインスタンスタイプの機能のリリース前に作成された既存のアプリケーションがアカウントにある場合、デフォルトの Standard インスタンスタイプを使用しています。既存のアプリケーションのビルドインスタンスタイプを更新すると、更新前にキューに入っているか進行中のビルドは、以前に設定したビルドインスタンスタイプを使用します。たとえば、`main` ブランチが Amplify にデプロイされた既存のアプリケーションがあり、そのビルドインスタンスタイプを Standard から Large に更新した場合、`main` ブランチから開始したすべての新しいビルドは Large ビルドインスタンスタイプを使用します。ただし、ビルドインスタンスタイプを更新したときに進行中のビルドは、引き続き Standard インスタンスで実行されます。

## Amplify コンソールでのビルドインスタンスタイプの設定

新しい Amplify アプリケーションを作成するときにビルドインスタンスタイプを設定するには、次の手順に従います。

新しいアプリケーションのビルドインスタンスタイプを設定するには

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
2. [すべてのアプリ] ページで、[アプリの新規作成] を選択します。
3. [Amplify で構築を開始] ページで、自分の Git リポジトリプロバイダーを選択し、[次へ] を選択します。

4. [リポジトリブランチを追加] ページで、次の操作を行います。

   a. 「最近更新されたリポジトリ」リストで、接続するリポジトリの名前を選択します。

   b. ブランチリストで、接続するリポジトリブランチの名前を選択します。

   c. [次へ] を選択します。

5. [アプリ設定] ページで、[詳細設定] セクションを開きます。

6. [ビルドインスタンスタイプ] で、リストから目的のインスタンスタイプを選択します。

7. Node.js ランタイムベースのアプリケーションをデプロイする場合は、大規模なインスタンスタイプを効果的に利用するようにヒープメモリサイズを設定します。これを行うには、[アプリ設定] で環境変数を設定するか、ビルド設定を更新します。詳細については、「大規模なインスタンスタイプを利用するようにアプリケーションのヒープメモリを設定する」を参照してください。

   - 環境変数を設定する

     a. [詳細設定] の [環境変数] セクションで、[新規追加] (Add new) を選択します。

     b. [Key] (キー) に「**NODE_OPTIONS**」と入力します。

     c. [Value] (値) に「--max-old-space-size=*memory_size_in_mb*」と入力します。*memory_size_in_mb* を目的のヒープメモリサイズにメガバイト単位で置き換えます。

   - ビルド設定の更新

     a. [ビルド設定] セクションで、[YML ファイルの編集] を選択します。

     b. 次のコマンドを preBuild フェーズに追加します。*memory_size_in_mb* を目的のヒープメモリサイズにメガバイト単位で置き換えます。

     ```
     export NODE_OPTIONS='--max-old-space-size=memory_size_in_mb'
     ```

     c. [保存] を選択します。

8. [次へ]をクリックします。

9. [レビュー]ページで、[保存してデプロイ] を選択します。

次の手順で、既存の Amplify アプリケーションのビルドインスタンスタイプを設定します。

既存のアプリケーションのビルドインスタンスタイプを設定するには

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。

2. ビルドインスタンスタイプを設定するアプリを選択します。
3. ナビゲーションペインで、[ホスティング] を選択し、[ビルド設定] を選択します。
4. [ビルド設定] ページの [詳細設定] セクションで、[編集] を選択します。
5. [設定の編集] ページの [ビルドインスタンスタイプ] で、リストから目的のインスタンスタイプを選択します。
6. [保存] を選択します。この変更は、次回アプリケーションをデプロイするときに有効になります。
7. (オプション) 更新されたアプリケーションをすぐにデプロイするには、次の手順を実行します。

    a. ナビゲーションペインで、[概要] を選択します。
    b. アプリケーションの概要ページで、再デプロイするブランチを選択します。
    c. [デプロイ] ページで、最新のデプロイなどのデプロイを選択します。次に、[このバージョンを再デプロイ] を選択します。新しいデプロイが開始します。
    d. デプロイが完了すると、アプリケーションのビルド設定に、ブランチが更新されたビルドインスタンスタイプを使用していることが示されます。

## 大規模なインスタンスタイプを利用するようにアプリケーションのヒープメモリを設定する

メモリを大量に消費するアプリケーションを構築する場合は、このセクションを使用して、大規模なインスタンスタイプを利用するようにアプリケーションを設定する方法を確認してください。プログラミング言語とフレームワークは、アプリケーションのメモリ要件を管理するために、ランタイム時にヒープメモリとも呼ばれる動的メモリの割り当てに依存することがよくあります。ヒープメモリはランタイム環境によってリクエストされ、ホストオペレーティングシステムによって割り当てられます。デフォルトでは、ランタイム環境はアプリケーションで使用可能なヒープサイズの上限を適用します。つまり、ホストオペレーティングシステムまたはコンテナで使用可能なメモリ量が多い場合でも、ヒープサイズを超える追加のメモリをアプリケーションに使用することはできません。

たとえば、JavaScript Node.js v8 ランタイム環境では、ホストメモリサイズなど、いくつかの要因に応じてデフォルトのヒープサイズ制限が適用されます。その結果、Standard および Large ビルドインスタンスのデフォルトの Node.js ヒープサイズは 2096 MB で、XLarge インスタンスのデフォルトのヒープサイズは 4144 MB になります。したがって、Amplify ビルドインスタンスタイプでデフォルトの Node.js ヒープサイズを使用してメモリ要件が 6000 MB のアプリケーションを構築すると、メモリ不足エラーによりビルドが失敗します。

Node.js のデフォルトのヒープサイズのメモリ制限を回避するには、次のいずれかを実行します。

- Amplify アプリケーションの NODE_OPTIONS 環境変数を値 --max-old-space-size=*memory_size_in_mb* に設定します。memory_size_in_mb では、必要なヒープメモリサイズをメガバイト単位で指定します。

  手順については、「環境変数の設定」を参照してください。

- Amplify アプリケーションのビルド仕様の preBuild フェーズに次のコマンドを追加します。

```
export NODE_OPTIONS='--max-old-space-size=memory_size_in_mb'
```

  ビルド仕様は、Amplify コンソールまたはプロジェクトリポジトリのアプリケーションの amplify.yml ファイルで更新できます。手順については、「Amplify アプリケーションのビルド設定の構成」を参照してください。

  次の Amplify ビルド仕様の例では、React フロントエンドアプリケーションを構築するために Node.js ヒープメモリサイズを 7000 MB に設定します。

```
version: 1
frontend:
  phases:
    preBuild:
      commands:
        # Set the heap size to 7000 MB
        - export NODE_OPTIONS='--max-old-space-size=7000'
        # To check the heap size memory limit in MB
        - node -e "console.log('Total available heap size (MB):',
 v8.getHeapStatistics().heap_size_limit / 1024 / 1024)"
        - npm ci --cache .npm --prefer-offline
    build:
      commands:
        - npm run build
  artifacts:
    baseDirectory: build
    files:
      - '**/*'
  cache:
    paths:
      - .npm/**/*
```

大規模なインスタンスタイプを効果的に活用するには、十分なヒープメモリサイズを設定することが重要です。メモリを大量に使用するアプリケーション用にヒープサイズを小さく設定すると、ビ

ルドが失敗する可能性があります。アプリケーションのランタイムが予期せずクラッシュする可能性があるため、アプリケーションのビルドログはメモリ不足エラーを直接示していない可能性があります。ホストメモリと同じサイズのヒープサイズを設定すると、ホストオペレーティングシステムが他のプロセスをスワップまたは終了し、ビルドプロセスが中断される可能性があります。参考として、Node.js では、メモリが約 2000 MB のマシンで最大ヒープサイズを 1536 MB に設定して、他の用途にメモリを残すことをお勧めします。

最適なヒープサイズは、アプリケーションのニーズとリソースの使用状況によって異なります。メモリ不足エラーが発生した場合は、中程度のヒープサイズから始めて、必要に応じて徐々に増やします。ガイドラインとして、Standard インスタンスタイプでは 6000 MB、Large インスタンスタイプでは 12000 MB、XLarge インスタンスタイプでは 60000 MB から開始することをお勧めします。

# ビルドを開始するための受信ウェブフックの作成

Amplify コンソールで着信ウェブフックを設定して、Git リポジトリにコードを入力せずにビルドを開始します。ヘッドレス CMS ツール (Contentful や GraphCMS など)でウェブフックを使用すると、コンテンツが変更されるたびにビルドを開始したり、Zapier などのサービスを使用して毎日ビルドを実行したりできます。

受信ウェブフックを作成するには

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
2. ウェブフックを作成するアプリを選択します。
3. ナビゲーションペインで [ホスティング]、[ビルド設定] を選択します。
4. ビルド設定ページで、「受信ウェブフック」セクションまでスクロールし、「ウェブフックを作成」を選択します。
5. [ウェブフックの作成] ダイアログボックスで、以下の操作を行います。

   a. [ウェブフック名] には、ウェブフックの名前を入力します。

   b. [ビルドするブランチ] で、受信したウェブフックのリクエストに基づいてビルドするブランチを選択します。

   c. [ウェブフックの作成] を選択します。
6. [受信するウェブフック] セクションで、次のいずれかの操作を行います。
   - ウェブフック URL をコピーし、ヘッドレス CMS ツールまたはその他のサービスに提供してビルドを開始します。
   - ターミナルウィンドウで curl コマンドを実行して、新しいビルドを開始します。

# ビルド用の E メール通知の設定

AWS Amplify アプリケーションの E メール通知を設定して、ビルドが成功または失敗したときに利害関係者またはチームメンバーに警告できます。Amplify ホスティングはアカウントに Amazon Simple Notification Service (Amazon SNS) トピックを作成し、それを使用してメール通知を設定します。通知は、Amplify アプリのすべてのブランチまたは特定のブランチに適用するように設定できます。

## E メール通知の設定

以下の手順を使用して、Amplify アプリのすべてのブランチまたは特定のブランチにメール通知を設定します。

Amplify アプリのメール通知を設定するには

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
2. メール通知を設定したいアプリを選択します。
3. ナビゲーションペインで [ホスティング]、[ビルドの通知] を選択します。[ビルドの通知] ページで、[通知の管理] を選択します。
4. [通知の管理] ページで、[新規追加] を選択します。
5. 次のいずれかを行います。
   - 1 つのブランチに通知を送信するには、[メール] に通知を送信する先のメールアドレスを入力します。[ブランチ] では、通知を送信するブランチの名前を選択します。
   - 接続しているすべてのブランチに通知を送信するには、[メール] に通知を送信する先のメールアドレスを入力します。[ブランチ] で [すべてのブランチ] を選択します。
6. [保存] を選択します。

# カスタムドメインの接続

Amplify ホスティングでデプロイしたアプリをカスタムドメインに接続できます。Amplify を使用してウェブアプリをデプロイすると、Amplify はそれを `https://branch-name.d1m7bkiki6tdw1.amplifyapp.com` などの URL を持つデフォルトの `amplifyapp.com` ドメインでホストします。アプリをカスタムドメインに接続すると、ユーザーは、アプリがカスタム URL (`https://www.example.com` など) でホストされていることを理解できます。

ドメインレジストラ (例: Amazon Route 53、GoDaddy) からカスタムドメインを購入することができます。Route 53 は Amazon のドメインネームシステム (DNS) ウェブサービスです。Route 53 の詳細については、「What is Amazon Route 53」(Amazon Route 53 とは?) を参照してください。サードパーティー認定ドメインレジストラの一覧については、ICANN のウェブサイトの「認定レジストラディレクトリ」を参照してください。

カスタムドメインを設定するときは、Amplify がプロビジョニングするデフォルトのマネージド証明書を使用するか、独自のカスタム証明書を使用できます。ドメインで使用中の証明書はいつでも変更できます。証明書の管理の詳細については、「SSL/TLS 証明書の使用」を参照してください。

カスタムドメインの設定を行う前に、次の前提条件を満たしていることを確認してください。

- 登録済みドメイン名を所有しています。
- によって発行またはインポートされた証明書がある AWS Certificate Manager。
- Amplify ホスティングにアプリをデプロイしました。

  この手順を完了するには、「Amplify ホスティングへのアプリのデプロイの概要」を参照してください。
- ドメインと DNS 用語に関する基本的な知識があります。

  ドメインと DNS の詳細については、「DNS の用語と概念を理解する」を参照してください。

> **Warning**
>
> 同じリージョン内の他の AWS アカウント (複数可) の異なる Amplify App (複数可) に既または以前に関連付けられているドメインを持つ Amplify アプリの DomainAssociation リクエストを開始する場合、これはクロスアカウントドメインの関連付けと見なされます。クロスアカウントドメインの関連付けリクエストには、手動検証が必要です。クロスアカウントドメインの関連付けを続行する場合は、AWS サポートにお問い合わせください。

トピック



# DNS の用語と概念を理解する

ドメインネームシステム (DNS) に関連する用語や概念に慣れていない場合は、以下のトピックがカスタムドメインを追加する手順を理解するのに役立ちます。

## DNS の用語

DNS に共通する用語を以下に示します。カスタムドメインを追加する手順を理解する助けになります。

CNAME

CNAME (正規レコード名) は、一連のウェブページに対してドメインをマスクし、それらが他の場所にあるかのように見せるための DNS レコードの一種です。CNAMES はサブドメインを完全修飾ドメイン名 (FQDN) にポイントします。たとえば、新しい CNAME レコードを作成して、サブドメイン www.example.com (www がサブドメイン) を、Amplify コンソールでアプリに割り当てられた FQDN ドメイン branch-name.d1m7bkiki6tdw1.cloudfront.net にマッピングできます。

ANAME

ANAME レコードは CNAME レコードと似ていますが、ルートレベルにあります。ANAME はドメインのルートを FQDN にポイントします。この FQDN は IP アドレスを指します。

ネームサーバー

ネームサーバーは、ドメイン名のさまざまなサービスの場所に関するクエリの処理に特化したインターネット上のサーバーです。Amazon Route 53 でドメインを設定した場合、ネームサーバーのリストはすでにドメインに割り当てられています。

NS レコード

NS レコードは、ドメインの詳細を検索するネームサーバーを指します。

## DNS の検証

ドメインネームシステム (DNS) は、人間が読めるドメイン名をコンピューターにとって使いやすい IP アドレスに変換する電話帳のようなものです。ブラウザに **`https://google.com`** と入力すると、DNS プロバイダーで検索操作が実行され、ウェブサイトをホストしているサーバーの IP アドレスが検索されます。

DNS プロバイダーには、ドメインとそれに対応する IP アドレスのレコードが含まれています。最も一般的に使用される DNS レコードは CNAME レコード、ANAME レコード、および NS レコードです。

Amplify は CNAME レコードを使用して、お客様がカスタムドメインを所有していることを確認します。ドメインを Route53 でホストしている場合、検証はお客様に代わって自動的に行われます。ただし、GoDaddy などのサードパーティプロバイダーでドメインをホストしている場合は、ドメインの DNS 設定を手動で更新し、Amplify が提供する新しい CNAME レコードを追加する必要があります。

## カスタムドメインアクティベーションのプロセス

> **Warning**
>
> 同じリージョン内の他の AWS アカウント (複数可) の異なる Amplify App (複数可) に既または以前に関連付けられているドメインを持つ Amplify アプリの DomainAssociation リクエストを開始する場合、これはクロスアカウントドメインの関連付けと見なされます。クロスアカウントドメインの関連付けリクエストには、手動検証が必要です。クロスアカウントドメインの関連付けを続行する場合は、AWS サポートにお問い合わせください。

Amplify コンソールで Amplify アプリをカスタムドメインに接続する場合、カスタムドメインを使用してアプリを表示する前に Amplify が完了する必要があるステップがいくつかあります。次のリストでは、ドメイン設定やアクティベーションのプロセスの各手順を説明しています。

SSL/TLS の作成

マネージド証明書を使用している場合、 は安全なカスタムドメインを設定するための SSL/TLS 証明書 AWS Amplify を発行します。

SSL/TLS の設定と検証

Amplify は、マネージド証明書を発行する前に、自身がドメインの所有者であることを確認します。Amazon Route 53 によって管理されているドメインの場合は、Amplify は DNS 検証レコードを自動的に更新します。Route53 の外部で管理されているドメインの場合は、Amplify コンソールで提供される DNS 検証レコードを、サードパーティー DNS プロバイダーによるドメインに手動で追加する必要があります。

カスタム証明書を使用している場合は、ドメインの所有権を検証する責任があります。

ドメインアクティベーション

ドメインは正常に検証されました。Route53 の外部で管理されているドメインの場合は、Amplify コンソールで提供される CNAME レコードを、サードパーティー DNS プロバイダーによるドメインに手動で追加する必要があります。

# SSL/TLS 証明書の使用

SSL/TLS 証明書は、安全な SSL/TLS プロトコルを使用してウェブブラウザがウェブサイトへの暗号化されたネットワーク接続を識別して確立できるようにするデジタルドキュメントです。カスタムドメインを設定するときは、Amplify がプロビジョニングするデフォルトのマネージド証明書を使用するか、独自のカスタム証明書を使用できます。

マネージド証明書により、Amplify は、すべてのトラフィックが HTTPS/2 を介して保護されるように、アプリに接続されたすべてのドメインに対して SSL/TLS 証明書を発行します。 AWS Certificate Manager (ACM) によって生成されたデフォルトの証明書は 13 か月間有効で、アプリが Amplify でホストされている限り自動的に更新されます。

> **Warning**
>
> ドメインプロバイダーの DNS 設定で CNAME 検証レコードが変更または削除されている場合、Amplify は証明書を更新できません。Amplify コンソールでドメインを削除して追加し直す必要があります。

カスタム証明書を使用するには、まず選択したサードパーティーの認証機関から証明書を取得する必要があります。Amplify ホスティングは、RSA (Rivest-Shamir-Adleman) と ECDSA (楕円曲線DSA) の 2 種類の証明書をサポートしています。各証明書タイプは、以下の要件に準拠する必要があります。

RSA 証明書

- Amplify ホスティングは、1024 ビット、2048 ビット、3072 ビット、4096 ビットの RSA キーをサポートしています。
- AWS Certificate Manager (ACM) は、最大 2048 ビットのキーを持つ RSA 証明書を発行します。
- 3072 ビットまたは 4096 ビットの RSA 証明書を使用するには、証明書を外部から取得して ACM にインポートします。その後、Amplify ホスティングで使用できます。

ECDSA 証明書

- Amplify ホスティングは 256 ビットキーをサポートしています。
- prime256v1 楕円曲線を使用して、Amplify ホスティング用の ECDSA 証明書を取得します。

証明書を取得したら、 にインポートします AWS Certificate Manager。ACM は、 および内部接続リソースで使用するパブリックおよびプライベート SSL/TLS 証明書を簡単にプロビジョニング、管理 AWS のサービス 、デプロイできるサービスです。米国東部 (バージニア北部) (us-east-1) リージョンの証明書をリクエストまたはインポートしていることを確認します。

カスタム証明書が、追加する予定のすべてのサブドメインをカバーしていることを確認します。ドメイン名の先頭にあるワイルドカードを使用すれば、複数のサブドメインをカバーできます。例えば、ドメインが `example.com` の場合、ワイルドカードドメイン `*.example.com` を入れることができます。これは、`product.example.com` や `api.example.com` などのサブドメインを対象とします。

ACM でカスタム証明書が利用可能になると、ドメイン設定プロセス中に選択できるようになります。 AWS Certificate Managerに証明書をインポートする手順については、「AWS Certificate Manager ユーザーガイド」の「AWS Certificate Managerに証明書をインポートする」を参照してください。

ACM でカスタム証明書を更新または再インポートすると、Amplify はカスタムドメインに関連付けられた証明書データを更新します。インポートされた証明書の場合、ACM は更新を自動的に管理しません。カスタム証明書を更新し、再度インポートする必要があります。

ドメインで使用中の証明書はいつでも変更できます。例えば、デフォルトのマネージド証明書からカスタム証明書に切り替えることも、カスタム証明書からマネージド証明書に変更することもできます。さらに、使用中のカスタム証明書を別のカスタム証明書に変更できます。証明書を更新する手順については、「ドメインの SSL/TLS 証明書を更新する」を参照してください。

## Amazon Route 53 が管理するカスタムドメインの追加

Amazon Route 53 は、高可用性でスケーラブルな DNS Web サービスです。詳細については、「Amazon Route 53 デベロッパーガイド」の「Amazon Route 53」を参照してください。Route 53 ドメインが既にある場合は、次の手順を使用してカスタムドメインを Amplify アプリに接続します。

Amazon Route 53 で管理されているカスタムドメインを追加するには

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
2. カスタムドメインに接続するアプリを選択します。
3. ナビゲーションペインで、[ホスティング]、[カスタムドメイン] を選択します。
4. [カスタムドメイン] ページで [ドメインの追加] を選択します。
5. お使いのルートドメインの名前を入力します。たとえば、ドメインの名前が https://example.com の場合は、**`example.com`** と入力します。

   入力を開始すると、Route 53 ですでに管理しているルートドメインがリストに表示されます。リストから使用するドメインを選択できます。ドメインをまだ所有しておらず、利用可能な場合は、Amazon Route 53 でドメインを購入できます。
6. ドメイン名を入力したら、[ドメインの設定] を選択します。
7. デフォルトでは、Amplify はドメインに対して2つのサブドメインエントリを自動的に作成します。たとえば、ドメイン名が example.com である場合、ルートドメインからwww サブドメインへのリダイレクトが設定された https://www.example.com と https://example.com のサブドメインが表示されます。

（オプション）サブドメインのみを追加する場合は、デフォルト設定を変更できます。デフォルト設定を変更するには、ナビゲーションペインから [書き換えとリダイレクト] を選択し、ドメインを設定します。

8. 使用する SSL/TLS 証明書を選択します。Amplify がプロビジョニングするデフォルトのマネージド証明書、またはインポートしたカスタムサードパーティー証明書を使用できます AWS Certificate Manager。
   - デフォルトの Amplify マネージド証明書を使用します。
     - [Amplify マネージド証明書] を選択します。
   - カスタムサードパーティー証明書を使用します。
     a. [カスタム SSL 証明書] を選択します。
     b. リストから使用する証明書を選択します。
9. [ドメインを追加する] を選択します。

> **Note**
>
> DNS が伝播して証明書が発行されるまでに最大 24 時間かかることがあります。発生したエラーの解決方法については、カスタムドメインのトラブルシューティング を参照してください。

# サードパーティーの DNS プロバイダーが管理するカスタムドメインの追加

Amazon Route 53 を使用してドメインを管理していない場合は、サードパーティの DNS プロバイダーが管理するカスタムドメインを Amplify でデプロイされたアプリに追加できます。

GoDaddy を使用している場合は、このプロバイダーに固有の手順について、「the section called “GoDaddy が管理するドメインの DNS レコードの更新”」を参照してください。

サードパーティーの DNS プロバイダーによって管理されるカスタムドメインを追加するには

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
2. カスタムドメインに接続するアプリを選択します。
3. ナビゲーションペインで、[ホスティング]、[カスタムドメイン] を選択します。

4. [カスタムドメイン] ページで [ドメインの追加] を選択します。
5. お使いのルートドメインの名前を入力します。たとえば、ドメインの名前が https://example.com の場合は、**`example.com`** と入力します。
6. Amplify は、Route 53 ドメインを使用していないことを検出し、Route 53 でホストゾーンを作成するオプションを提供します。
   - Route 53 にホストゾーンを作成するには
     a. [Route 53 でホストゾーンを作成する] を選択します。
     b. [ドメインの設定] を選択します。
     c. ホストゾーンのネームサーバーはコンソールに表示されます。DNS プロバイダーのウェブサイトに移動し、DNS 設定にネームサーバーを追加します。
     d. [上記のネームサーバーをドメインレジストリに追加した] を選択します。
     e. ステップ 7 に進みます。
   - 手動設定を続行するには
     a. [手動設定] を選択する
     b. [ドメインの設定] を選択します。
     c. ステップ 7 に進みます。
7. デフォルトでは、Amplify はドメインに対して2つのサブドメインエントリを自動的に作成します。たとえば、ドメイン名が example.com である場合、ルートドメインからwww サブドメインへのリダイレクトが設定された https://www.example.com と https://example.com のサブドメインが表示されます。

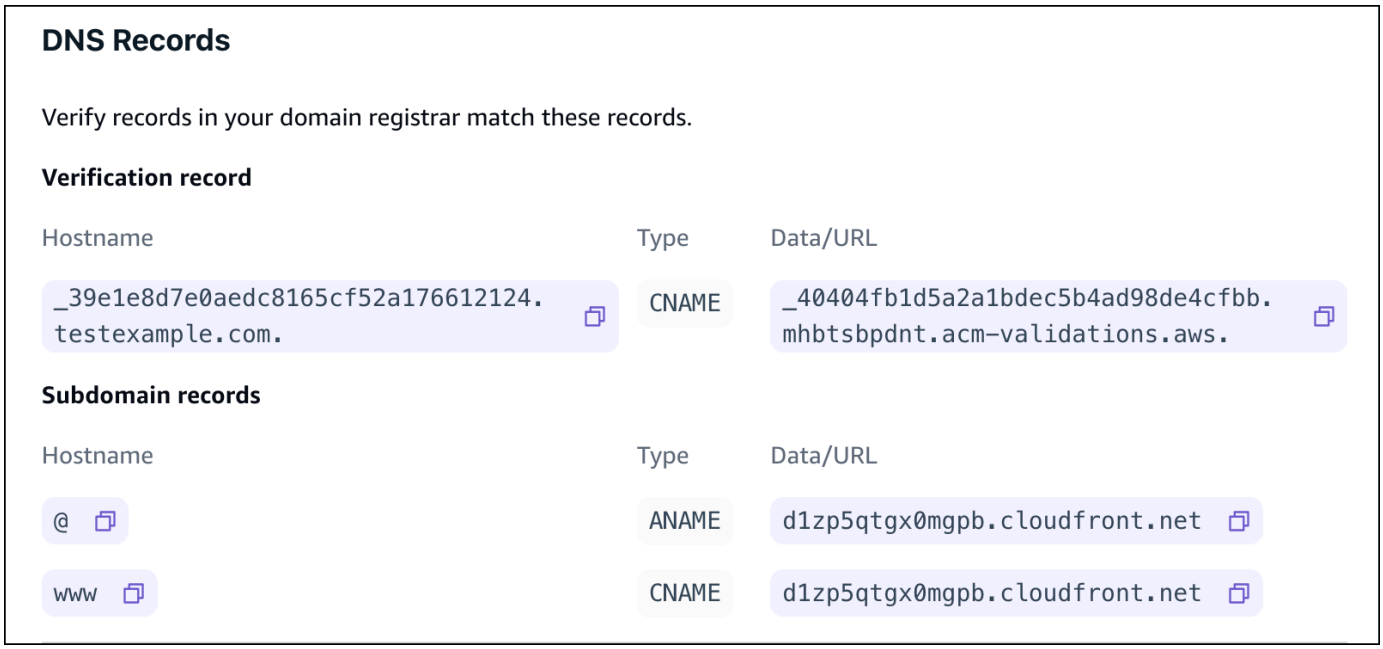
   （オプション）サブドメインのみを追加する場合は、デフォルト設定を変更できます。デフォルト設定を変更するには、ナビゲーションペインから [書き換えとリダイレクト] を選択し、ドメインを設定します。
8. 使用する SSL/TLS 証明書を選択します。Amplify がプロビジョニングするデフォルトのマネージド証明書、またはインポートしたカスタムサードパーティー証明書を使用できます AWS Certificate Manager。
   - デフォルトの Amplify マネージド証明書を使用します。
     - [Amplify マネージド証明書] を選択します。
   - カスタムサードパーティー証明書を使用します。
     a. [カスタム SSL 証明書] を選択します。

b. リストから使用する証明書を選択します。

9. [ドメインを追加する] を選択します。
10. ステップ 6 で [Route 53 でホストゾーンを作成する] を選択した場合は、ステップ 15 に進みます。

    [手動設定] を選択した場合は、ステップ 6 で、サードパーティーのドメインプロバイダーで DNS レコードを更新する必要があります。

    [アクション]メニューで、[DNS レコードの表示]を選択します。次のスクリーンショットは、コンソールに表示される DNS レコードを示しています。



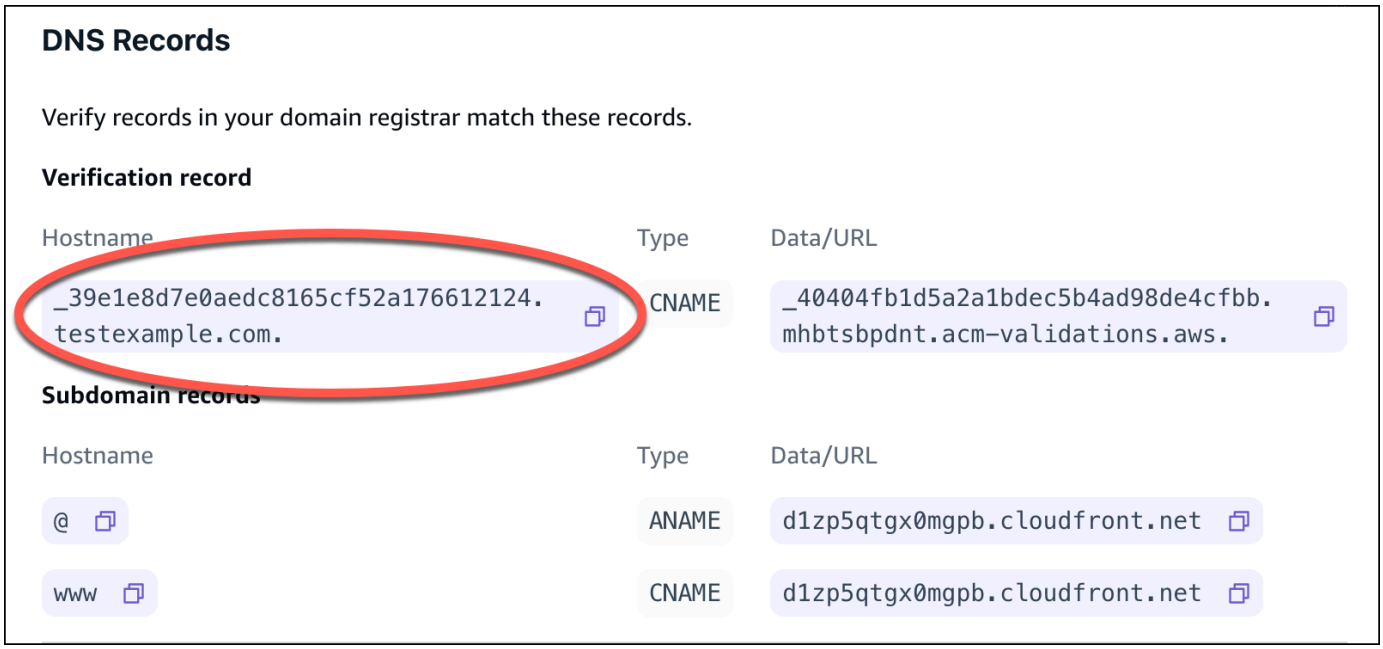

11. 次のいずれかを行います。

    - GoDaddy を使用している場合は、GoDaddy が管理するドメインの DNS レコードの更新 にアクセスしてください。
    - 別のサードパーティの DNS プロバイダーを使用している場合は、この手順の次のステップに進んでください。

12. DNS プロバイダーのウェブサイトに移動し、アカウントにログインして、ドメインの DNS 管理設定を確認します。2 つの CNAME レコードを設定できます。
13. サブドメインを検証 AWS サーバーを指すように最初の CNAME レコードを設定します。

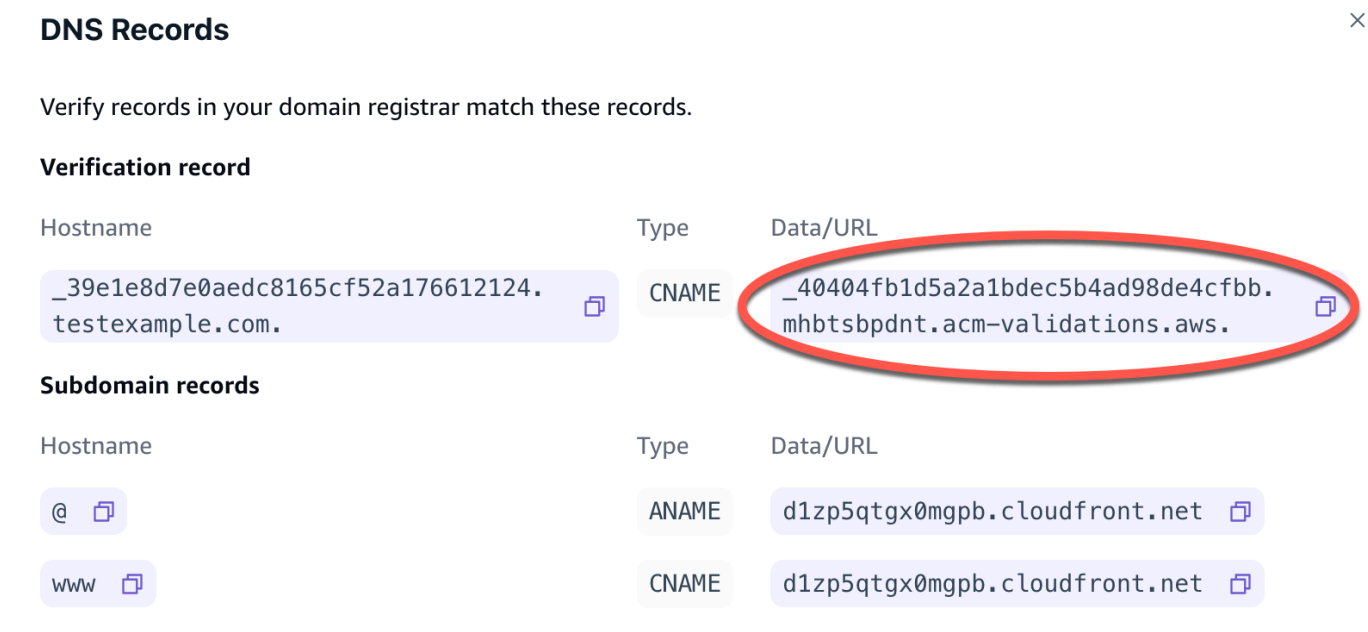
    Amplify コンソールに _c3e2d7eaf1e656b73f46cd6980fdc0e.example.com などのサブドメインの所有権を検証するための DNS レコードが表示された場合は、CNAME レコードのサブドメイン名に **_c3e2d7eaf1e656b73f46cd6980fdc0e** のみを入力します。

次のスクリーンショットは、使用する検証レコードの場所を示しています。




Amplify コンソールに _cjhwou20vhu2exampleuw20vuyb2ovb9.j9s73ucn9vy.acm-validations.aws などの ACM 検証サーバーレコードが表示された場合は、CNAME レコード値に **`_cjhwou20vhu2exampleuw20vuyb2ovb9.j9s73ucn9vy.acm-validations.aws`** を入力します。

次のスクリーンショットは、使用する ACM 検証レコードの場所を示しています。



Amplify はこの情報を使用してドメインの所有権を確認し、ドメインの SSL/TLS 証明書を生成します。Amplify でドメインの所有権が検証されると、トラフィックはすべて、HTTPS/2 を使用して提供されます。

> **Note**
>
> AWS Certificate Manager (ACM) によって生成されたデフォルトの Amplify 証明書は 13 か月間有効で、アプリが Amplify でホストされている限り、自動的に更新されます。CNAME 検証レコードが変更または削除された場合、Amplify は証明書を更新できません。Amplify コンソールでドメインを削除して追加し直す必要があります。

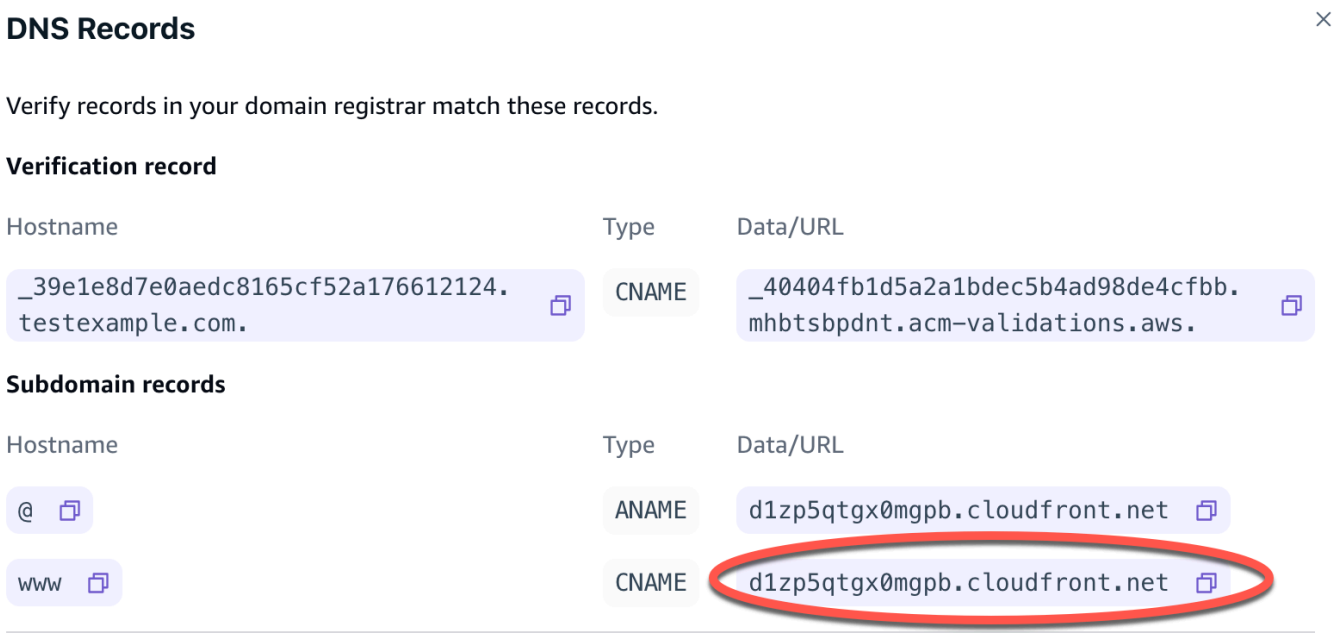
> **Important**
>
> Amplify コンソールでカスタムドメインを追加した直後に、このステップを実行することが重要です。 AWS Certificate Manager (ACM) はすぐに所有権の検証を開始します。時間が経つにつれて、チェックの頻度は少なくなります。アプリを作成してから数時間後に CNAME レコードを追加または更新すると、アプリが検証保留中の状態で停止する可能性があります。

14. 2 番目の CNAME レコードを設定して、サブドメインを Amplify ドメインにポイントします。たとえば、サブドメインが www.example.com の場合、サブドメイン名に www と入力します。

    Amplify コンソールにアプリのドメインが d111111abcdef8.cloudfront.net と表示される場合は、Amplify ドメインに **`d111111abcdef8.cloudfront.net`** と入力します。

    本稼働トラフィックがある場合は、Amplify コンソールでドメインのステータスが AVAILABLE になった後に CNAME レコードを更新することをお勧めします。

    次のスクリーンショットは、使用するドメイン名レコードの場所を示しています。




15. ANAME/ALIAS レコードを設定して、アプリのルートドメイン (https://example.com など) をポイントします。ANAME レコードでは、お客様のドメインのルートはホスト名を指します。本稼働トラフィックがある場合は、コンソールでドメインのステータスが AVAILABLE になった後に ANAME レコードを更新することをお勧めします。ANAME/ALIAS サポートのない DNS プロバイダーでは、DNS を Route 53 に移行することを強くお勧めします。詳細については、「Amazon Route 53 を DNS サービスとして設定する」を参照してください。

> **Note**
>
> サードパーティードメインのドメイン所有権と DNS の伝播の検証には最大 48 時間かかることがあります。発生したエラーの解決方法については、「カスタムドメインのトラブルシューティング」を参照してください。

# GoDaddy が管理するドメインの DNS レコードの更新

GoDaddy が DNS プロバイダーの場合は、次の手順を使用して GoDaddy UI の DNS レコードを更新し、Amplify アプリと GoDaddy ドメインの接続を完了します。

GoDaddy が管理するカスタムドメインを追加するには

1. GoDaddy で DNS レコードを更新する前に、手順 the section called “サードパーティーの DNS プロバイダーが管理するカスタムドメインの追加” のステップ 1～9 を完了します。

2. GoDaddy アカウントにログインします。
3. ドメインのリストで、追加するドメインを見つけて [DNS の管理] を選択します。
4. GoDaddy では、[DNS] ページの [DNS レコード] セクションにドメインのレコードリストが表示されます。2 つの新しい CNAME レコードを追加する必要があります。
5. 最初の CNAME レコードを作成して、サブドメインが Amplify ドメインを指すようにします。

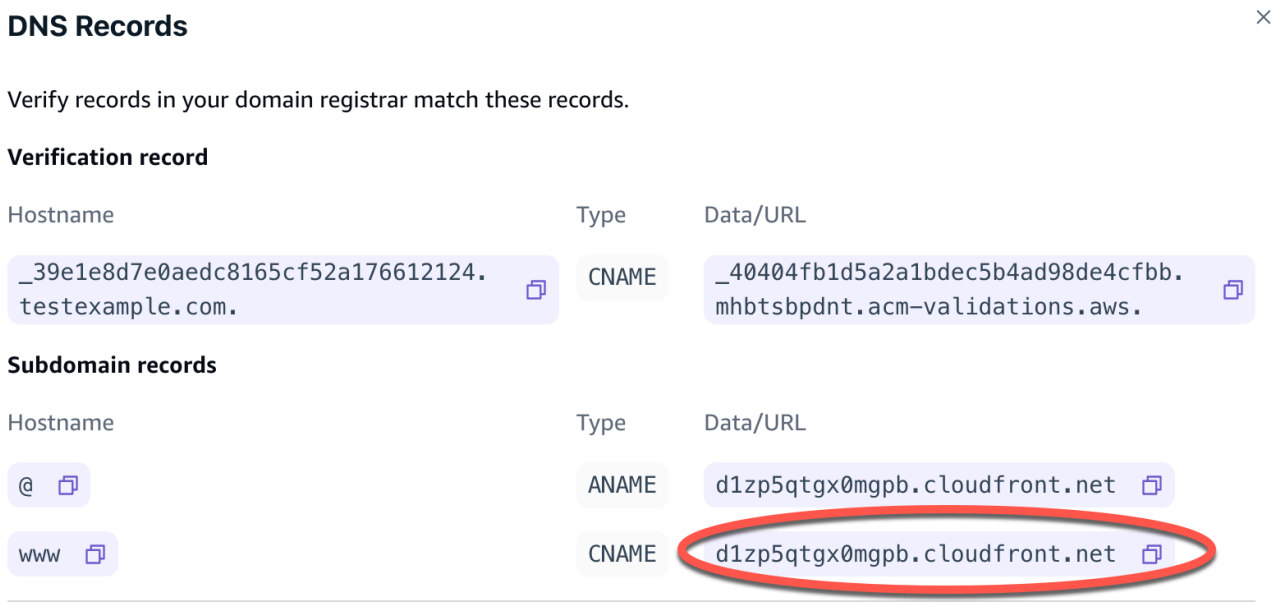
   a. [DNS レコード] セクションで、[新規レコードの追加] を選択します。

   b. [タイプ]には [CNAME] を選択します。

   c. [Name] (名前) には、サブドメインのみを入力します。たとえば、サブドメインが www.example.com の場合、[名前] に「www」と入力します。

   d. [値] には、Amplify コンソールで DNS レコードを確認し、値を入力します。Amplify コンソールにアプリのドメインが d111111abcdef8.cloudfront.net と表示されている場合は、[値] に **`d111111abcdef8.cloudfront.net`** と入力します。

      次のスクリーンショットは、使用するドメイン名レコードの場所を示しています。




   e. [保存] を選択します。

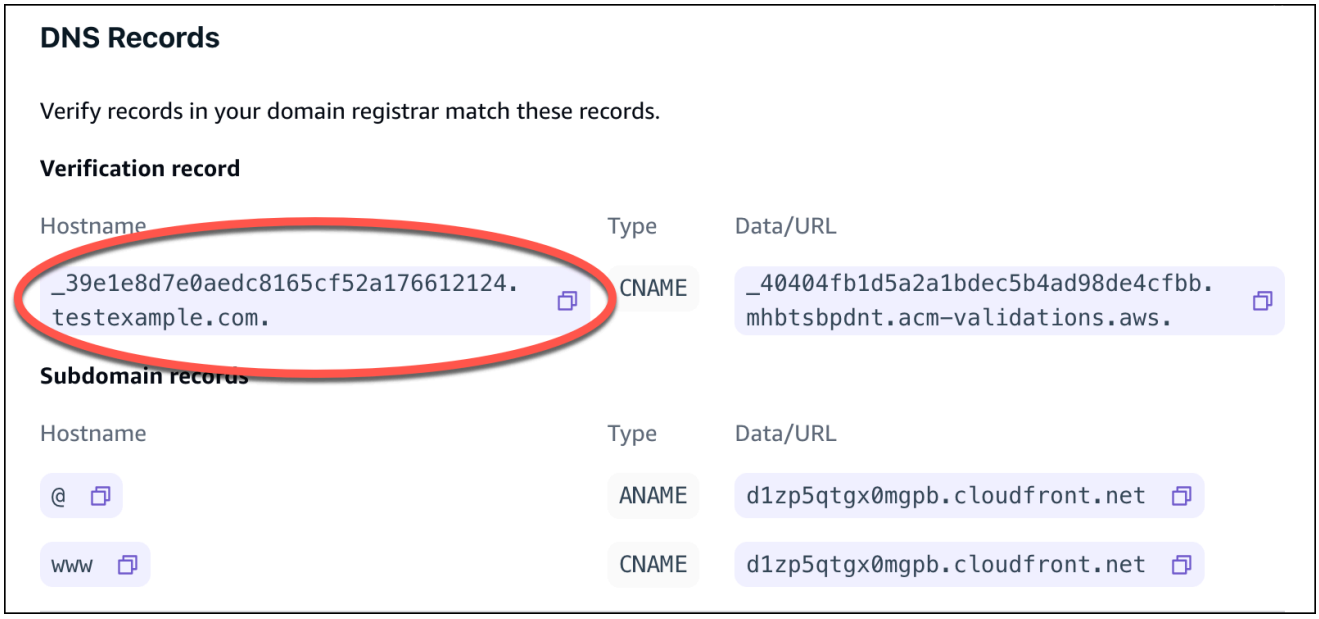
6. ( AWS Certificate Manager ACM) 検証サーバーを指す 2 番目の CNAME レコードを作成します。検証済みの 1 つの ACM はドメインの SSL/TLS 証明書を生成します。

   a. [タイプ]には [CNAME] を選択します。

   b. [Name] (名前) には、サブドメインを入力します。

たとえば、サブドメインの所有権を確認するための Amplify コンソールの DNS レコードが _c3e2d7eaf1e656b73f46cd6980fdc0e.example.com の場合、[名前] には **_c3e2d7eaf1e656b73f46cd6980fdc0e** のみを入力します。

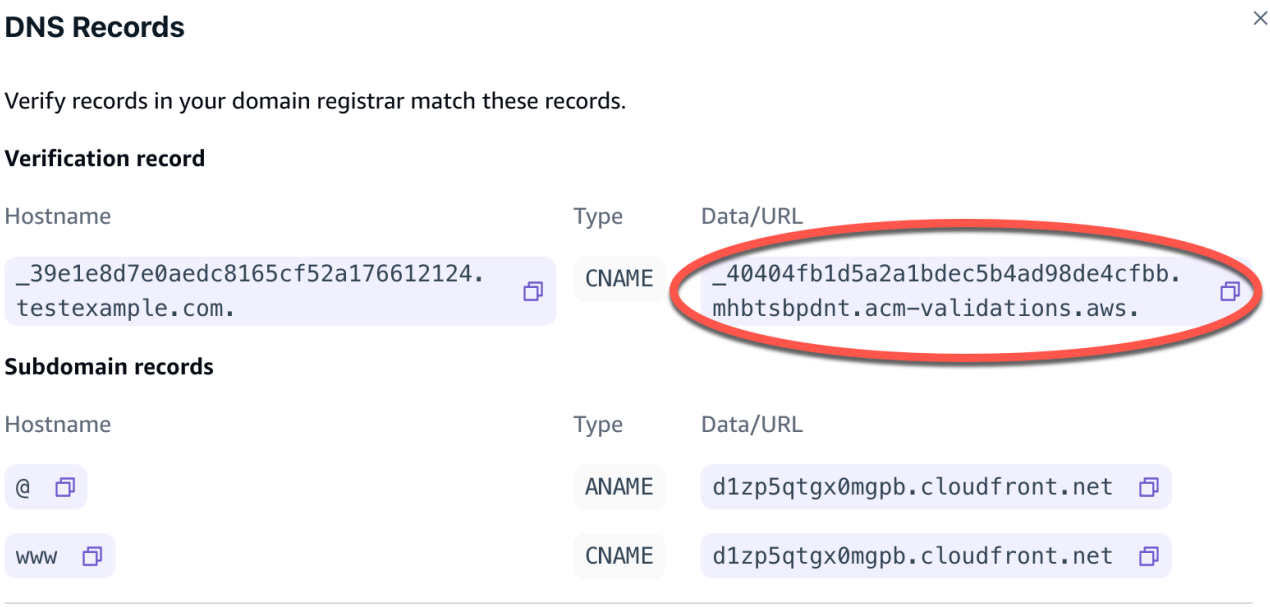
次のスクリーンショットは、使用する検証レコードの場所を示しています。




c. [値] には、ACM 検証証明書を入力します。

たとえば、検証サーバーが _cjhwou20vhu2exampleuw20vuyb2ovb9.j9s73ucn9vy.acm-validations.aws の場合、[値] には _cjhwou20vhu2exampleuw20vuyb2ovb9.j9s73ucn9vy.acm-validations.aws と入力します。

次のスクリーンショットは、使用する ACM 検証レコードの場所を示しています。




d. [保存] を選択します。

> **Note**
>
> AWS Certificate Manager (ACM) によって生成されたデフォルトの Amplify 証明書は 13 か月間有効で、アプリが Amplify でホストされている限り、自動的に更新されます。CNAME 検証レコードが変更または削除された場合、Amplify は証明書を更新できません。Amplify コンソールでドメインを削除して追加し直す必要があります。

7. このステップはサブドメインには必要ありません。GoDaddy では、ANAME/ALIAS レコードはサポートされていません。ANAME/ALIAS サポートのない DNS プロバイダーでは、DNS を Amazon Route 53 に移行することを強くお勧めします。詳細については、「Amazon Route 53 を DNS サービスとして設定する」を参照してください。

   GoDaddy をプロバイダーとして保持し、ルートドメインを更新する場合は、[転送] を追加して、ドメイン転送を設定します。

   a. [DNS] ページで、ページの上部にあるメニューを見つけ、[転送] を選択します。
   b. [ドメイン] セクションで、[転送の追加] を選択します。
   c. [http://] を選択し、[転送先 URL] には、転送先のサブドメイン名 (www.example.com など) を入力します。
   d. [転送タイプ] には [一時 (302)] を選択します。
   e. [保存] を選択します。

# ドメインの SSL/TLS 証明書の更新

ドメインで使用中の SSL/TLS 証明書は、いつでも変更できます。例えば、マネージド証明書からカスタム証明書を使用するように変更できます。これは、証明書とその有効期限通知を管理する場合に便利です。ドメインで使用中のカスタム証明書を変更することもできます。SSL 証明書を変更しても、アクティブなドメインのダウンタイムは発生しません。証明書の詳細については、「SSL/TLS 証明書の使用」を参照してください。

次の手順に従って、ドメインで使用中の証明書またはカスタム証明書のタイプを更新します。

ドメインの証明書を更新するには

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
2. 更新するアプリを選択します。
3. ナビゲーションペインで、[ホスティング]、[カスタムドメイン] を選択します。
4. [カスタムドメイン] ページで、[ドメイン設定] を選択します。
5. ドメインの詳細ページで、[カスタム SSL 証明書] セクションを見つけます。証明書を更新する手順は、変更したいタイプによって異なります。
   - カスタム証明書からデフォルトの Amplify マネージド証明書に変更するには
     - [Amplify マネージド証明書] を選択します。
   - マネージド証明書からカスタム証明書に変更するには
     a. [カスタム SSL 証明書] を選択します。
     b. リストから使用する証明書を選択します。
   - カスタム証明書を別のカスタム証明書に変更するには
     - カスタム SSL 証明書 の場合、リストから使用する新しい証明書を選択します。
6. [保存] を選択します。ドメインのステータス詳細には、Amplify がマネージド証明書の SSL 作成プロセスまたはカスタム証明書の設定プロセスを開始したことが示されます。

# サブドメインの管理

サブドメインは URL の中でドメイン名の前に表示される部分です。たとえば、www は www.amazon.com のサブドメインで、aws は aws.amazon.com のサブドメインです。本番サイトが既にある場合は、サブドメインだけを接続したほうがいいかもしれません。サブドメインは複数レ

ベルにすることもできます。たとえば、beta.alpha.example.com にはマルチレベルのサブドメイン beta.alpha があります。

## サブドメインのみを追加するには

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
2. サブドメインを追加するアプリを選択します。
3. ナビゲーションペインで、[ホスティング] を選択し、[カスタムドメイン]を選択します。
4. [カスタムドメイン] ページで [ドメインの追加] を選択します。
5. ルートドメインの名前を入力し、[ドメインの設定] を選択します。たとえば、ドメインの名前が https://example.com の場合は、example.com と入力します。
6. [ルートを除外] を選択し、サブドメインの名前を変更します。たとえば、ドメインが example.com の場合、サブドメイン alpha のみを追加するように変更できます。
7. [ドメインを追加する] を選択します。

## マルチレベルサブドメインを追加するには

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
2. マルチレベルサブドメインを追加するアプリを選択します。
3. ナビゲーションペインで、[ホスティング] を選択し、[カスタムドメイン]を選択します。
4. [カスタムドメイン] ページで [ドメインの追加] を選択します。
5. サブドメイン付きのドメインの名前を入力し、[ルートを除外] を選択し、サブドメインを変更して新しいレベルを追加します。

   たとえば、alpha.example.com というドメインを所有しており、複数レベルのサブドメイン beta.alpha.example.com を作成したい場合は、サブドメインの値として beta と入力します。
6. [ドメインを追加する] を選択します。

## サブドメインを追加または編集するには

アプリにカスタムドメインを追加したら、既存のサブドメインを編集したり、新しいサブドメインを追加したりできます。

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。

2. サブドメインを管理したいアプリを選択します。
3. ナビゲーションペインで、[ホスティング] を選択し、[カスタムドメイン]を選択します。
4. [カスタムドメイン] ページで、[ドメイン設定] を選択します。
5. [サブドメイン] セクションでは、必要に応じて既存のサブドメインを編集できます。
6. (省略可) 新しいサブドメインを追加するには、[新規追加] を選択します。
7. [保存] を選択します。

# ワイルドカードサブドメインの設定

Amplify ホスティングはワイルドカードサブドメインをサポートするようになりました。ワイルドカードサブドメインは、既存のサブドメインと存在しないサブドメインをアプリケーションの特定のブランチに向けることができる包括的なサブドメインです。ワイルドカードを使用してアプリのすべてのサブドメインを特定のブランチに関連付けると、どのサブドメインでもアプリのユーザーに同じコンテンツを配信でき、各サブドメインを個別に設定する必要がなくなります。

ワイルドカードサブドメインを作成するには、サブドメイン名としてアスタリスク (*) を指定します。たとえば、アプリの特定のブランチにワイルドカードサブドメイン `*.example.com` を指定すると、example.com で終わる URL はすべてそのブランチにルーティングされます。この場合、`dev.example.com` および `prod.example.com` のリクエストは `*.example.com` サブドメインにルーティングされます。

Amplify はカスタムドメインでのみワイルドカードサブドメインをサポートしていることに注意してください。この機能はデフォルトの `amplifyapp.com` ドメインでは使用できません。

ワイルドカードサブドメインには、次の要件が適用されます。

- サブドメイン名はアスタリスク (*) のみで指定する必要があります。
- *domain.example.com のように、サブドメイン名の一部をワイルドカードで置き換えることはできません。
- 「subdomain.*.example.com」 のように、ドメイン名の途中にあるサブドメインを置き換えることはできません。
- デフォルトでは、Amplify でプロビジョニングされるすべての証明書は、カスタムドメインのすべてのサブドメインを対象としています。

## ワイルドカードサブドメインを追加または削除するには

アプリにカスタムドメインを追加したら、アプリブランチにワイルドカードサブドメインを追加できます。

1. にサインイン AWS マネジメントコンソール し、Amplify ホスティングコンソールを開きます。
2. ワイルドカードサブドメインを管理したいアプリを選択します。
3. ナビゲーションペインで、[ホスティング] を選択し、[カスタムドメイン]を選択します。
4. [カスタムドメイン] ページで、[ドメイン設定] を選択します。
5. [サブドメイン] セクションでは、ワイルドカードサブドメインを追加または削除できます。
   - ワイルドカードサブドメインを新しく追加するには
     a. [新規追加] を選択します。
     b. サブドメインの場合、* を入力します。
     c. アプリブランチの場合、リストからブランチ名を選択します。
     d. [保存] を選択します。
   - ワイルドカードサブドメインを削除するには
     a. サブドメイン名の横にある [削除] を選択します。明示的に設定されていないサブドメインへのトラフィックは停止し、Amplify ホスティングはそれらのリクエストに 404 ステータスコードを返します。
     b. [保存] を選択します。

# Amazon Route 53 カスタムドメイン用の自動サブドメインの設定

アプリを Route 53 のカスタムドメインに接続すると、Amplify では新しく接続されたブランチのサブドメインを自動的に作成できます。たとえば、dev ブランチに接続すると、Amplify は dev.exampledomain.com を自動的に作成できます。ブランチを削除すると、関連するサブドメインはすべて自動的に削除されます。

新しく接続したブランチにサブドメインを自動作成するように設定するには

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
2. Route 53 で管理されているカスタムドメインに接続されているアプリを選択します。
3. ナビゲーションペインで、[ホスティング] を選択し、[カスタムドメイン]を選択します。

4. [カスタムドメイン] ページで、[ドメイン設定] を選択します。
5. [自動サブドメインの作成] セクションで、機能をオンにします。

> **Note**
>
> この機能は exampledomain.com などのルートドメインでのみ使用できます。ドメインが既に dev.exampledomain.com などのサブドメインになっている場合、Amplify コンソールにはこのチェックボックスは表示されません。

## サブドメインを使ったウェブプレビュー

前述の手順で、[サブドメインの自動作成] を有効にすると、自動的に作成されたサブドメインでアプリのプルリクエストのウェブプレビューにもアクセスできるようになります。プルリクエストがクローズされると、関連するブランチとサブドメインは自動的に削除されます。プルリクエストのウェブプレビューの設定に関する詳細は、プルリクエストの Web プレビュー を参照してください。

# カスタムドメインのトラブルシューティング

AWS Amplify コンソールでアプリにカスタムドメインを追加する際に問題が発生した場合は、Amplify トラブルシューティングの章の「カスタムドメインのトラブルシューティング」を参照してください。そちらで問題の解決策が見つからない場合は、 サポートにお問い合わせください。詳細については、「AWS サポート ユーザーガイド」の「サポートケースの作成」を参照してください。

# Amplify ホストサイトのファイアウォールサポート

Amplify ホストサイトのファイアウォールサポートにより、 と直接統合してウェブアプリケーションを保護できます AWS WAF。 AWS WAF を使用すると、ウェブアクセスコントロールリスト (ウェブ ACL) と呼ばれる一連のルールを設定して、定義したカスタマイズ可能なウェブセキュリティルールと条件に基づいてウェブリクエストを許可、ブロック、またはモニタリング (カウント) できます。Amplify アプリを と統合すると AWS WAF、アプリが受け入れる HTTP トラフィックをより詳細に制御し、可視化できます。詳細については AWS WAF、「 AWS WAF デベロッパーガイド」のAWS WAF 「 の仕組み」を参照してください。

ファイアウォールのサポートは、Amplify ホスティングが動作するすべての AWS リージョン で利用できます。この統合は、CloudFront と同様に AWS WAF グローバルリソースに分類されます。ウェブ ACL は複数の Amplify ホスティングアプリケーションにアタッチできますが、同じリージョンに存在する必要があります。

AWS WAF を使用して、SQL インジェクションやクロスサイトスクリプティングなどの一般的なウェブエクスプロイトから Amplify アプリを保護できます。これらは、アプリケーションの可用性とパフォーマンスに影響を与え、セキュリティを侵害したり、過剰なリソースを消費したりする可能性があります。たとえば、指定した IP アドレス範囲からのリクエスト、CIDR ブロックからのリクエスト、特定の国またはリージョンからのリクエスト、予期しない SQL コードやスクリプトを含むリクエストを許可またはブロックするルールを作成できます。

また、HTTP ヘッダー、メソッド、クエリ文字列、URI、およびリクエストボディの指定された文字列または定型表現パターン (最初の 8 KB に制限されます) に一致するルールを作成することもできます。さらに、特定のユーザーエージェント、ボット、またはコンテンツスクレーパーからのイベントをブロックするルールを作成できます。例えば、レートベースのルールを使用して、継続的に更新される後続の 5 分間で、各クライアント IP によって許可されるウェブリクエストの数を指定できます。

サポートされるルールのタイプと追加 AWS WAF 機能の詳細については、「AWS WAF デベロッパーガイド」とAWS WAF 「 API リファレンス」を参照してください。

> **Important**
>
> セキュリティは、 AWS とお客様の責任を共有します。 AWS WAF は、すべてのインターネットセキュリティ問題のソリューションではなく、セキュリティとコンプライアンスの目的を満たすように設定する必要があります。の使用時に責任共有モデルを適用する方法を理

解するには AWS WAF、AWS WAF 「サービスの使用におけるセキュリティ」を参照してください。

トピック

- で Amplify AWS WAF アプリケーションの を有効にする AWS マネジメントコンソール
- Amplify アプリケーションからウェブ ACL の関連付けを解除する
- を使用して Amplify AWS WAF アプリケーションで を有効にする AWS CDK
- Amplify と の統合方法 AWS WAF
- Amplify アプリケーションのファイアウォール料金

# で Amplify AWS WAF アプリケーションの を有効にする AWS マネジメントコンソール

Amplify アプリ AWS WAF の保護は、Amplify コンソールまたは AWS WAF コンソールで有効にできます。

- Amplify コンソール — Amplify コンソールで AWS WAF ウェブ ACL をアプリに関連付けることで、既存の Amplify アプリのファイアウォール機能を有効にできます。ワンクリック保護を使用して、ほとんどのアプリのベストプラクティスとして考えられる、事前設定されたルールを持つウェブ ACL を作成します。IP アドレスと国ごとにアクセスをカスタマイズすることもできます。このセクションの手順では、ワンクリック保護の設定について説明します。
- AWS WAF コンソール — AWS WAF コンソールまたは AWS WAF APIs。グローバル (CloudFront) リージョンで Amplify アプリに関連付けるウェブ ACL を作成する必要があります。リージョン別ウェブ ACLs は に既に存在する可能性がありますが AWS アカウント、Amplify と互換性がありません。開始手順については、「 AWS WAF デベロッパーガイド」の「 AWS WAF とそのコンポーネントのセットアップ」を参照してください。

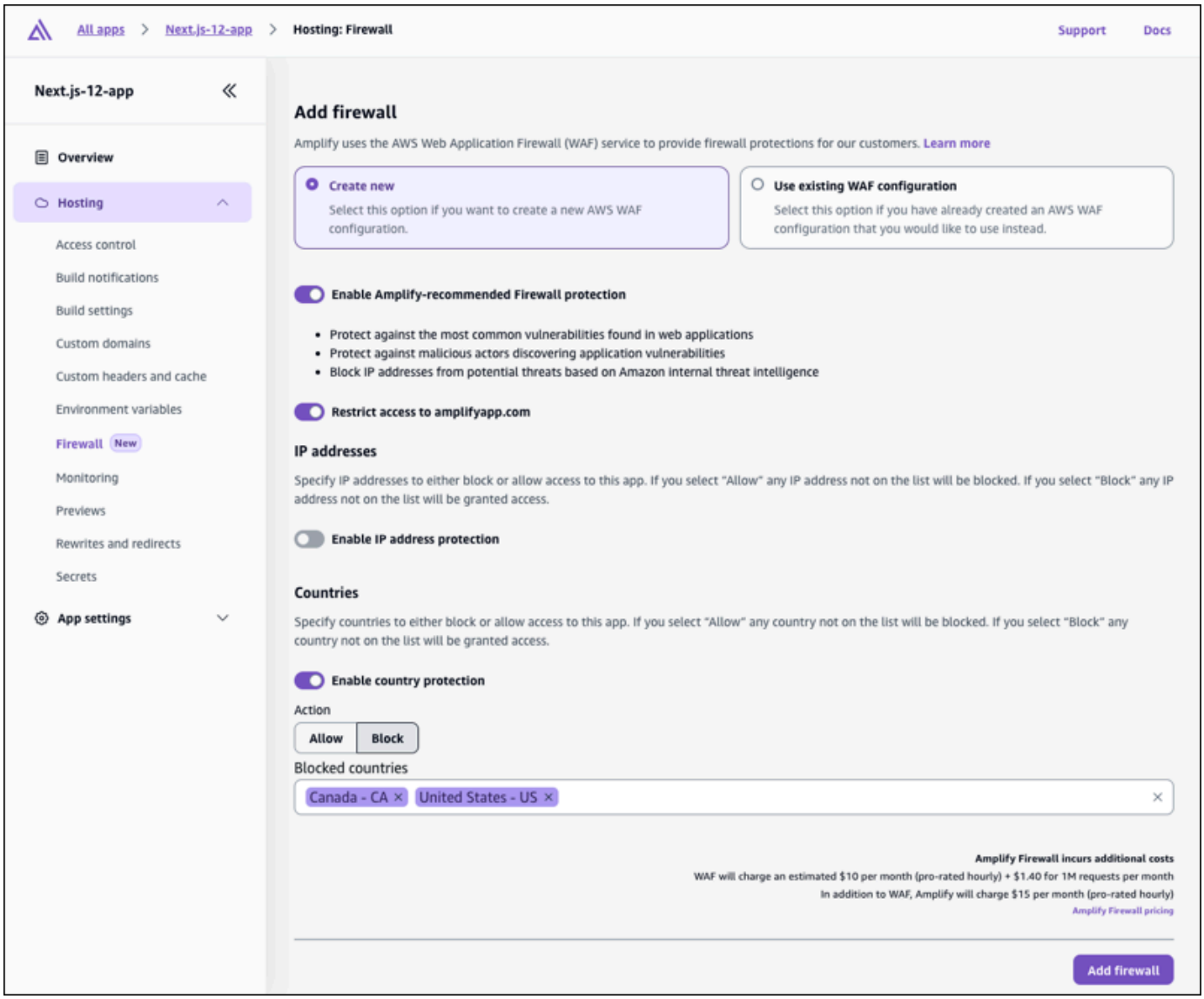
Amplify コンソールで既存のアプリ AWS WAF に対して を有効にするには、次の手順に従います。

既存の Amplify アプリ AWS WAF で を有効にする

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/amplify/ で Amplify コンソールを開きます。

2. [すべてのアプリ] ページで、ファイアウォール機能を有効にするデプロイ済みアプリの名前を選択します。
3. ナビゲーションペインで、[ホスティング] を選択し、[ファイアウォール] を選択します。

   次のスクリーンショットは、Amplify コンソールで [ファイアウォールの追加] ページに移動する方法を示しています。




4. ファイアウォールの追加ページでのアクションは、新しい設定を作成するか、既存の AWS WAF 設定を使用するかによって異なります。
   - 新しい AWS WAF 設定を作成します。
     a. [新規作成] を選択します。
     b. 必要に応じて、次のいずれかの設定を有効にします。

i. [Amplify が推奨するファイアウォール保護を有効にする] (Enable Amplify-recommended Firewall protection) を有効にします。

ii. デフォルトの Amplify ドメインでアプリへのアクセスを回避するには、[amplifyapp.com へのアクセスを制限する] を有効にします。

iii. [IP アドレス] では、[IP アドレス保護を有効にする] (Enable IP address protections) を有効にします。

   A. [アクション] では、アクセス権を持つ IP アドレスを指定し、他のすべての IP アドレスをブロックする場合は [許可] を選択します。ブロックされる IP アドレスを指定し、他のすべての IP アドレスにアクセスを許可する場合は、[ブロック] を選択します。

   B. [IP バージョン] では、[IPV4] または [IPV6] を選択します。

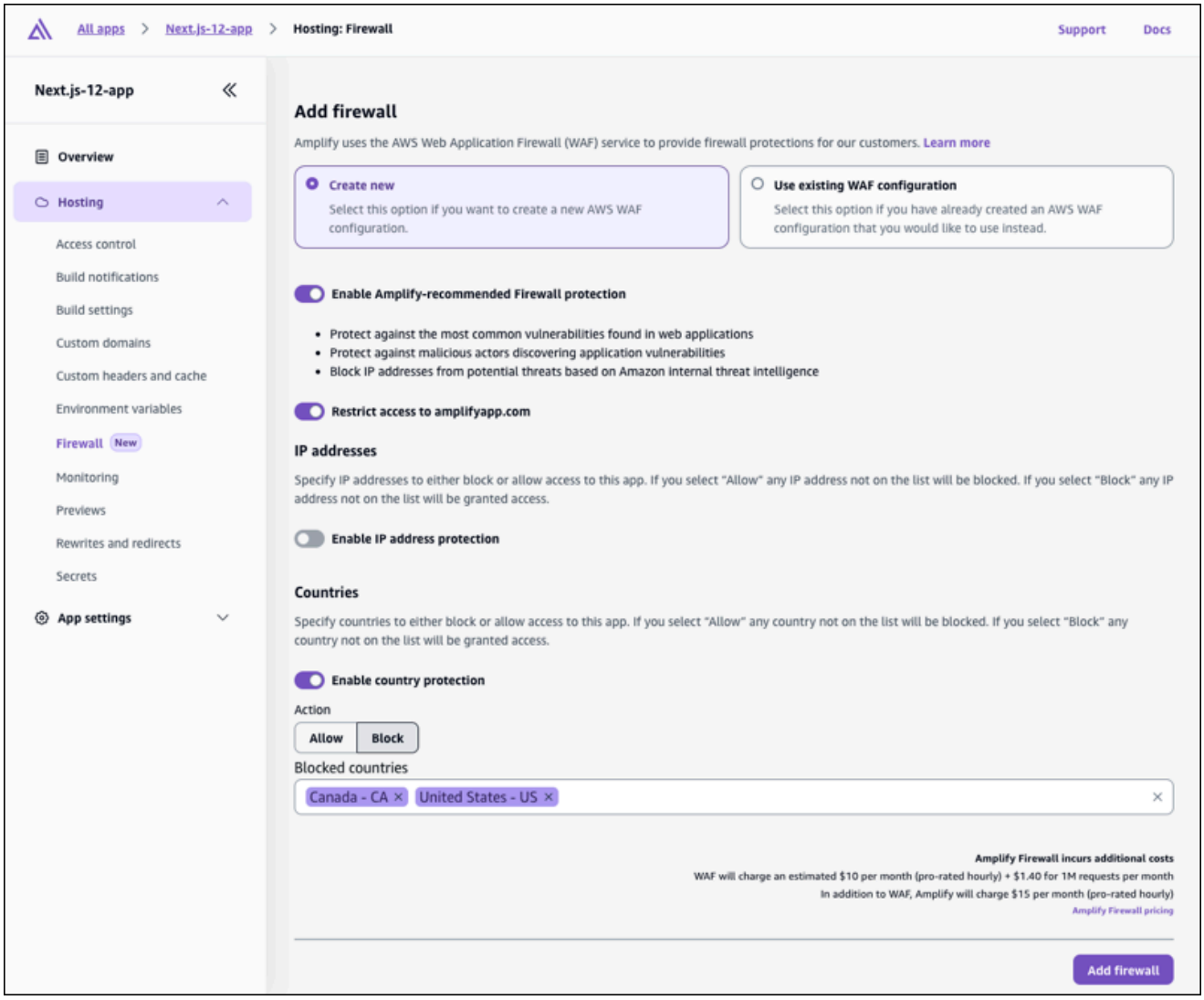
   C. [IP アドレス] テキストボックスに、許可された IP アドレスまたはブロックされた IP アドレスを CIDR 形式で 1 行に 1 つずつ入力します。

iv. [国] では、[国保護を有効にする] (Enable country protection) を有効にします。

   A. [アクション] では、アクセス権を持つ国を指定し、他のすべての国をブロックする場合は [許可] を選択します。ブロックされる国を指定し、他のすべての国にアクセスを許可する場合は、[ブロック] を選択します。

   B. [国] では、リストから許可またはブロックされる国を選択します。

次のスクリーンショットは、アプリケーションの新しい AWS WAF 設定を有効にする方法を示しています。



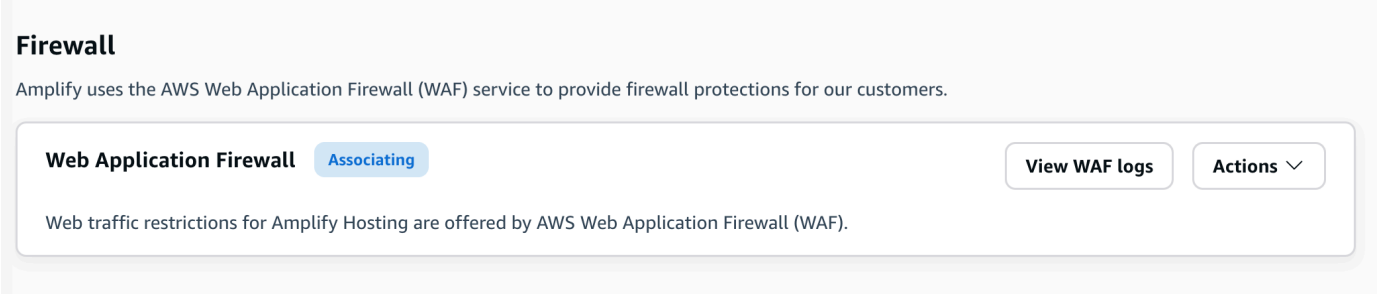

- 既存の AWS WAF 設定を使用します。

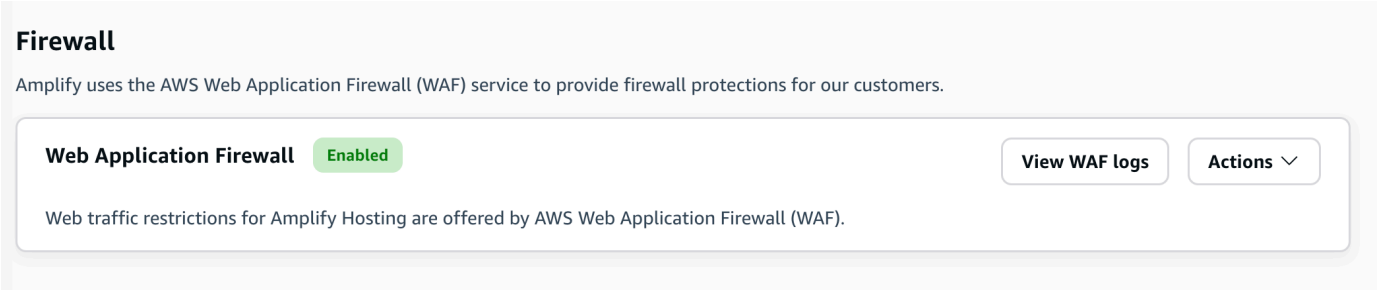
  a. 既存の AWS WAF 設定を使用する を選択します。

  b. の AWS WAF にあるウェブ ACLs のリストから、保存された設定を選択します AWS アカウント。Amplify アプリに関連付けるウェブ ACL は、グローバル (CloudFront) リージョンで作成する必要があります。リージョン別ウェブ ACLs は に既に存在する可能性がありますが AWS アカウント、Amplify と互換性がありません。

5. [ファイアウォールの追加] を選択します。
6. ファイアウォールページで、関連付けステータスが表示され、 AWS WAF 設定が伝播されていることを示します。プロセスが完了すると、ステータスは [有効] に変わります。

次のスクリーンショットは、Amplify コンソールのファイアウォールの進行状況ステータスを示しています。これは、 AWS WAF 設定が関連付けられているタイミングと有効になっていることを示します。




# Amplify アプリケーションからウェブ ACL の関連付けを解除する

Amplify アプリに関連付けられているウェブ ACL は削除できません。まず、Amplify コンソールでウェブ ACL とアプリの関連付けを解除する必要があります。 AWS WAF コンソールで削除できます。

Amplify アプリからウェブ ACL の関連付けを解除するには

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/amplify/ で Amplify コンソールを開きます。
2. [すべてのアプリ] ページで、ウェブ ACL の関連付けを解除するアプリの名前を選択します。
3. ナビゲーションペインで、[ホスティング] を選択し、[ファイアウォール] を選択します。
4. [ファイアウォール] ページで [アクション] を選択し、[ファイアウォールの関連付けを解除] (Disassociate firewall) を選択します。
5. 確認モーダルで「**`disassociate`**」と入力し、[ファイアウォールの関連付けを解除] (Disassociate firewall) を選択します。
6. ファイアウォールページで、関連付け解除ステータスが表示され、 AWS WAF 設定が伝播されていることを示します。

プロセスが完了したら、 AWS WAF コンソールでウェブ ACL を削除できます。

# を使用して Amplify AWS WAF アプリケーションで を有効にする AWS CDK

を使用して AWS Cloud Development Kit (AWS CDK) 、Amplify アプリケーション AWS WAF に対して を有効にできます。CDK の使用の詳細については、「AWS Cloud Development Kit (AWS CDK) デベロッパーガイド」の「CDK とは」を参照してください。

次の TypeScript コード例は、2 つの CDK スタックを持つ AWS CDK アプリケーションを作成する方法を示しています。1 つは Amplify 用、もう 1 つは 用です AWS WAF。 AWS WAF スタックは米国東部 (バージニア北部) (us-east-1) リージョンにデプロイする必要があることに注意してください。Amplify アプリケーションスタックは、別のリージョンにデプロイできます。グローバル (CloudFront) リージョンで Amplify アプリに関連付けるウェブ ACL を作成する必要があります。リージョン別ウェブ ACLs は に既に存在する可能性がありますが AWS アカウント、Amplify と互換性がありません。

```
import * as cdk from "aws-cdk-lib";
import { Construct } from "constructs";
import * as wafv2 from "aws-cdk-lib/aws-wafv2";
import * as amplify from "aws-cdk-lib/aws-amplify";

interface WafStackProps extends cdk.StackProps {
  appArn: string;
}

export class AmplifyStack extends cdk.Stack {
  public readonly appArn: string;
  constructor(scope: Construct, id: string, props?: cdk.StackProps) {
    super(scope, id, props);
    const amplifyApp = new amplify.CfnApp(this, "AmplifyApp", {
      name: "MyApp",
    });
    this.appArn = amplifyApp.attrArn;
  }
}

export class WAFStack extends cdk.Stack {
  constructor(scope: Construct, id: string, props: WafStackProps) {
```

```
    super(scope, id, props);
    const webAcl = new wafv2.CfnWebACL(this, "WebACL", {
      defaultAction: { allow: {} },
      scope: "CLOUDFRONT",
      rules: [
        // Add your own rules here.
      ],
      visibilityConfig: {
        cloudWatchMetricsEnabled: true,
        metricName: "my-metric-name",
        sampledRequestsEnabled: true,
      },
    });

    new wafv2.CfnWebACLAssociation(this, "WebACLAssociation", {
      resourceArn: props.appArn,
      webAclArn: webAcl.attrArn,
    });
  }
}

const app = new cdk.App();

// Create AmplifyStack in your desired Region.
const amplifyStack = new AmplifyStack(app, 'AmplifyStack', {
  env: { region: 'us-west-2' },
});

// Create WAFStack in IAD region, passing appArn from AmplifyStack.
new WAFStack(app, 'WAFStack', {
  env: { region: 'us-east-1' },
  crossRegionReferences: true,

  appArn: amplifyStack.appArn,  // Pass appArn from AmplifyStack.
});
```

# Amplify と の統合方法 AWS WAF

次のリストは、Firewall サポートを AWS WAF と統合する方法と、ウェブ ACL を作成して Amplify アプリに関連付ける際に考慮すべき制約に関する具体的な詳細を示しています。

- は、任意のタイプの Amplify アプリ AWS WAF に対して有効にできます。これには、サポートされているフレームワーク、サーバー側レンダリング (SSR) アプリ、完全静的サイトが含まれます。 AWS WAF は、Amplify Gen 1 および Gen 2 アプリでサポートされています。
- グローバル (CloudFront) リージョンで Amplify アプリに関連付けるウェブ ACL を作成する必要があります。リージョンウェブ ACLs は に既に存在する可能性がありますが AWS アカウント、Amplify と互換性がありません。
- ウェブ ACL と Amplify アプリは、同じ AWS アカウントで作成する必要があります。を使用して AWS WAF ルール AWS Firewall Manager をレプリケートし AWS アカウント、組織ルールを複数の にまたがって一元化および分散させることを簡素化できます AWS アカウント。詳細については、「 AWS WAF デベロッパーガイド」の 「AWS Firewall Manager」を参照してください。
- 同じ AWS アカウント内の複数の Amplify アプリ間で同じウェブ ACL を共有できます。アプリはすべて、同じリージョンにある必要があります。
- ウェブ ACL を Amplify アプリに関連付けると、ウェブ ACL はデフォルトでアプリ内のすべてのブランチにアタッチされます。新しいブランチを作成すると、ウェブ ACL が作成されます。
- ウェブ ACL を Amplify アプリに関連付けると、そのウェブ ACL はアプリのすべてのドメインに自動的に関連付けられます。ただし、ホストヘッダー一致ルールを使用して、単一のドメイン名に適用されるルールを設定できます。
- Amplify アプリに関連付けられているウェブ ACL は削除できません。 AWS WAF コンソールでウェブ ACL を削除する前に、アプリからウェブ ACL の関連付けを解除する必要があります。

## Amplify ウェブ ACL リソースポリシー

Amplify がウェブ ACL にアクセスできるようにするために、関連付け中にリソースポリシーがウェブ ACL にアタッチされます。Amplify はこのリソースポリシーを自動的に構築しますが、 AWS WAFV2 GetPermissionPolicy API を使用して表示できます。ウェブ ACL を Amplify アプリに関連付けるには、次の IAM アクセス許可が必要です。

- amplify:AssociateWebACL
- wafv2:AssociateWebACL
- wafv2:PutPermissionPolicy
- wafv2:GetPermissionPolicy

# Amplify アプリケーションのファイアウォール料金

Amplify アプリケーション AWS WAF に を実装するコストは、次の 2 つのコンポーネントに基づいて計算されます。

- usageAWS WAF – AWS WAF 料金モデルに従って使用量に対して AWS WAF 課金されます。AWS WAF 料金は、作成したウェブアクセスコントロールリスト (ウェブ ACLs)、ウェブ ACL ごとに追加するルールの数、および受信したウェブリクエストの数に基づきます。料金の詳細については、「AWS WAF の料金」を参照してください。
- Amplify ホスティング統合コスト – ウェブ ACL を Amplify アプリケーションにアタッチすると、アプリケーションごとに 1 か月あたり 15.00 USD の料金が発生します。これは時間あたりで日割り計算されます。

# 機能ブランチのデプロイとチームのワークフロー

Amplify ホスティングは、機能ブランチと GitFlow ワークフローで動作するように設計されています。Amplify は Git ブランチを使用して、リポジトリに新しいブランチを接続するたびに新しいデプロイを作成します。最初のブランチを接続すると、追加の機能ブランチが作成されます。

ブランチをアプリに追加するには

1. ブランチを追加するアプリを選択します。
2. [アプリ設定] を選択し、次に [ブランチ設定] を選択します。
3. [ブランチ設定] ページで、[ブランチの追加] を選択します。
4. リポジトリからブランチを選択します。
5. [ブランチの追加] を選択します。
6. アプリを再デプロイします。

ブランチを追加すると、アプリには https://main.appid.amplifyapp.com や https://dev.appid.amplifyapp.com など、Amplify のデフォルトドメインで 2 つのデプロイが利用可能になります。これはチーム間で異なる場合がありますが、通常は main ブランチはリリースコードを追跡します。また、本稼働ブランチでもあります。開発ブランチは、新機能をテストするための統合ブランチとして使用されます。これによって、ベータテスターは main ブランチデプロイの本稼働エンドユーザーに影響を及ぼすことなく、開発ブランチデプロイの未リリース機能をテストできます。

トピック

- フルスタックの Amplify Gen 2 アプリを使用したチームワークフロー
- フルスタックの Amplify Gen 1 アプリを使用したチームワークフロー
- パターンベースの機能ブランチのデプロイ
- Amplify 設定のビルド時の自動生成 (Gen 1 アプリ専用)
- 条件付きバックエンドビルド (Gen 1 アプリ専用)
- アプリ間で Amplify バックエンドを使用する (Gen 1 アプリ専用)

# フルスタックの Amplify Gen 2 アプリを使用したチームワークフロー

AWS Amplify Gen 2 では、バックエンドを定義するための TypeScript ベースのコードファースト開発者エクスペリエンスが導入されています。Amplify Gen 2 アプリケーションを使用したフルスタックワークフローの詳細については、「Amplify ドキュメント」の「フルスタックワークフロー」を参照してください。

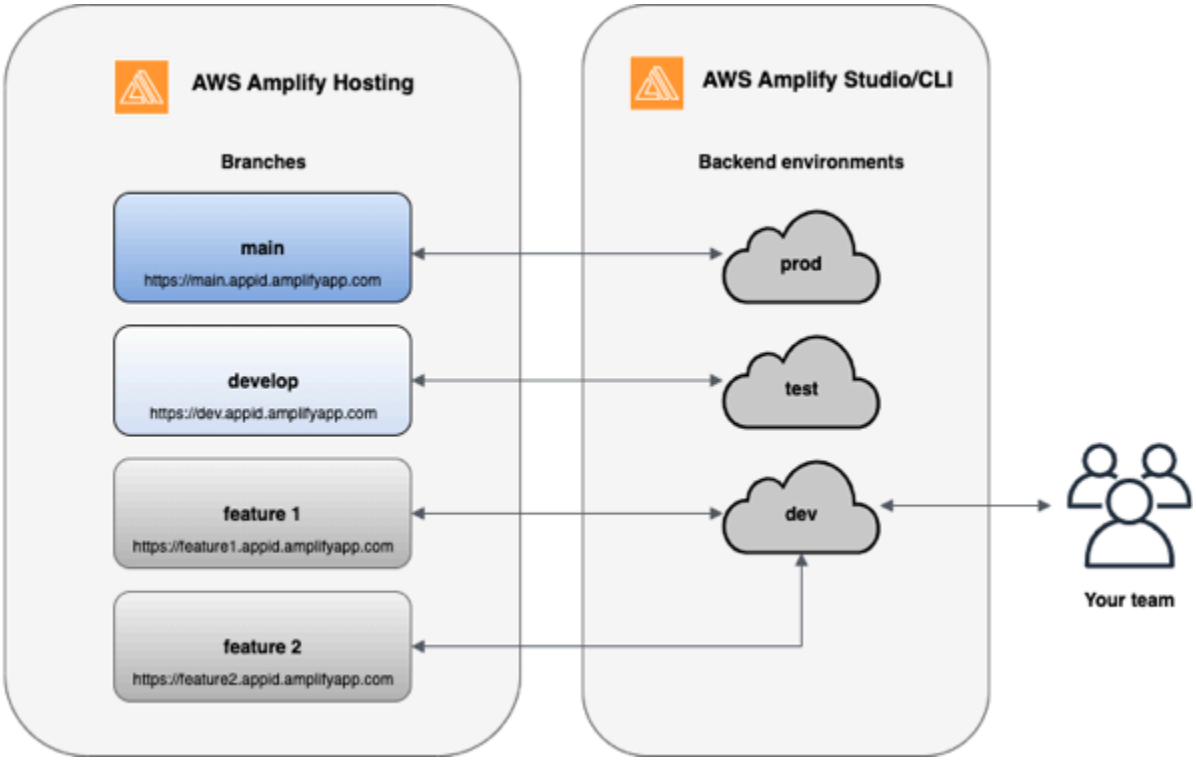
# フルスタックの Amplify Gen 1 アプリを使用したチームワークフロー

機能ブランチのデプロイは、フロントエンドと、オプションのバックエンド環境で構成されます。フロントエンドはグローバルコンテンツ配信ネットワーク (CDN) に構築およびデプロイされ、バックエンドは Amplify Studio または Amplify CLI によって AWSにデプロイされます。このデプロイシナリオを設定する方法については、「アプリケーションのバックエンドの構築」を参照してください。

Amplify ホスティングは、機能ブランチのデプロイで GraphQL API や Lambda 関数などのバックエンドリソースを継続的にデプロイします。次のブランチモデルを使用して、バックエンドとフロントエンドを Amplify ホスティングでデプロイできます。

## 機能ブランチのワークフロー

- Amplify Studio または Amplify CLI で、prod、test、dev バックエンド環境を作成します。
- prod バックエンドを main ブランチにマッピングします。
- test バックエンドを develop ブランチにマッピングします。
- チームメンバーは dev バックエンド環境を使用して個々の機能ブランチをテストできます。




1. Amplify CLI をインストールして新しい Amplify プロジェクトを初期化します。

```
npm install -g @aws-amplify/cli
```

2. プロジェクト用の prod バックエンド環境を初期化します。プロジェクトがない場合は、create-react-app や Gatsby などのブートストラップツールを使用してプロジェクトを作成します。

```
create-react-app next-unicorn
cd next-unicorn
amplify init
 ? Do you want to use an existing environment? (Y/n): n
 ? Enter a name for the environment: prod
...
amplify push
```

3. test と dev バックエンド環境を追加します。

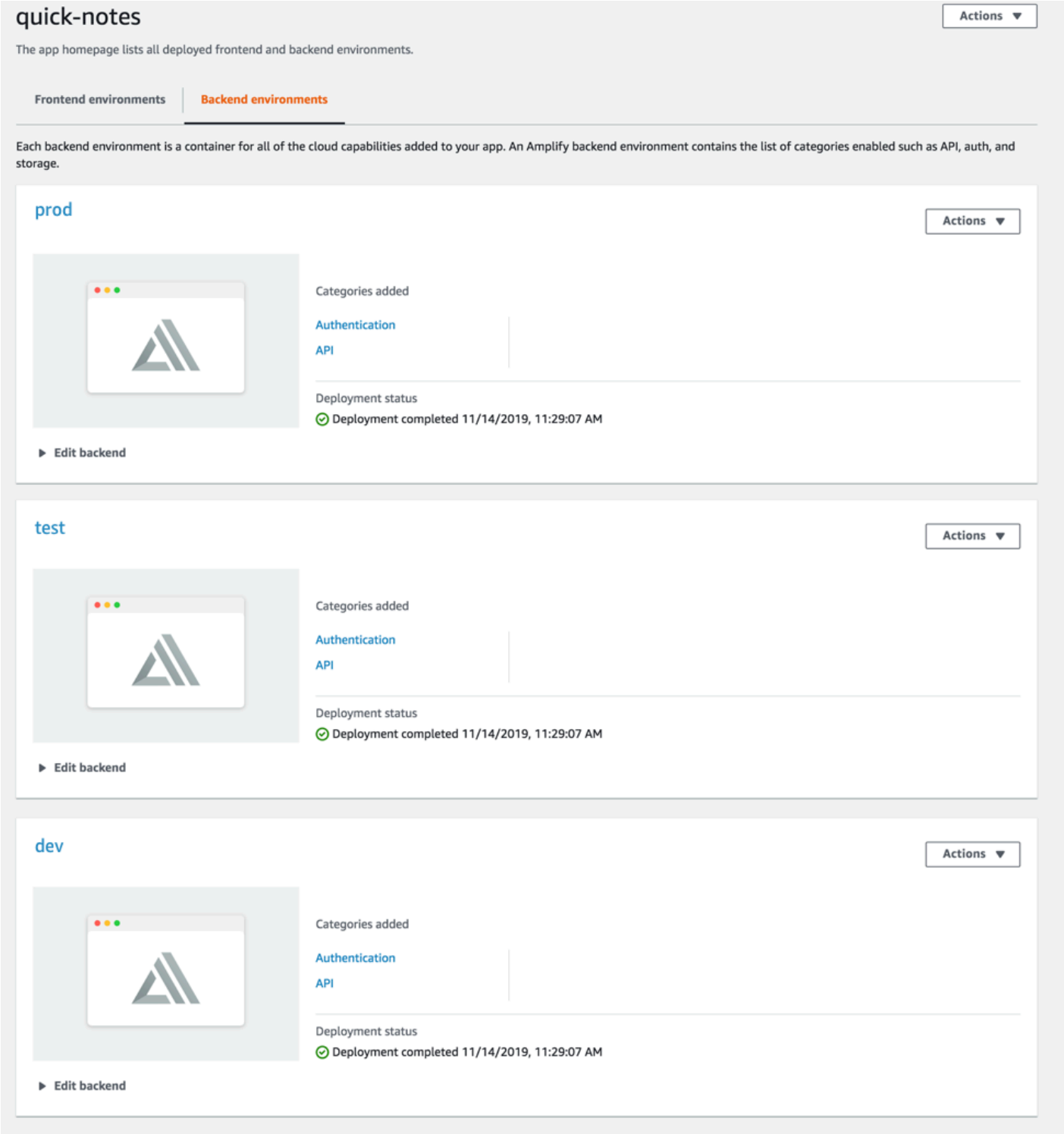
```
amplify env add
 ? Do you want to use an existing environment? (Y/n): n
 ? Enter a name for the environment: test
...
amplify push
```

```
amplify env add
 ? Do you want to use an existing environment? (Y/n): n
 ? Enter a name for the environment: dev
...
amplify push
```

4. 選択した Git リポジトリにコードをプッシュします (この例では、main にプッシュしたと仮定します)。

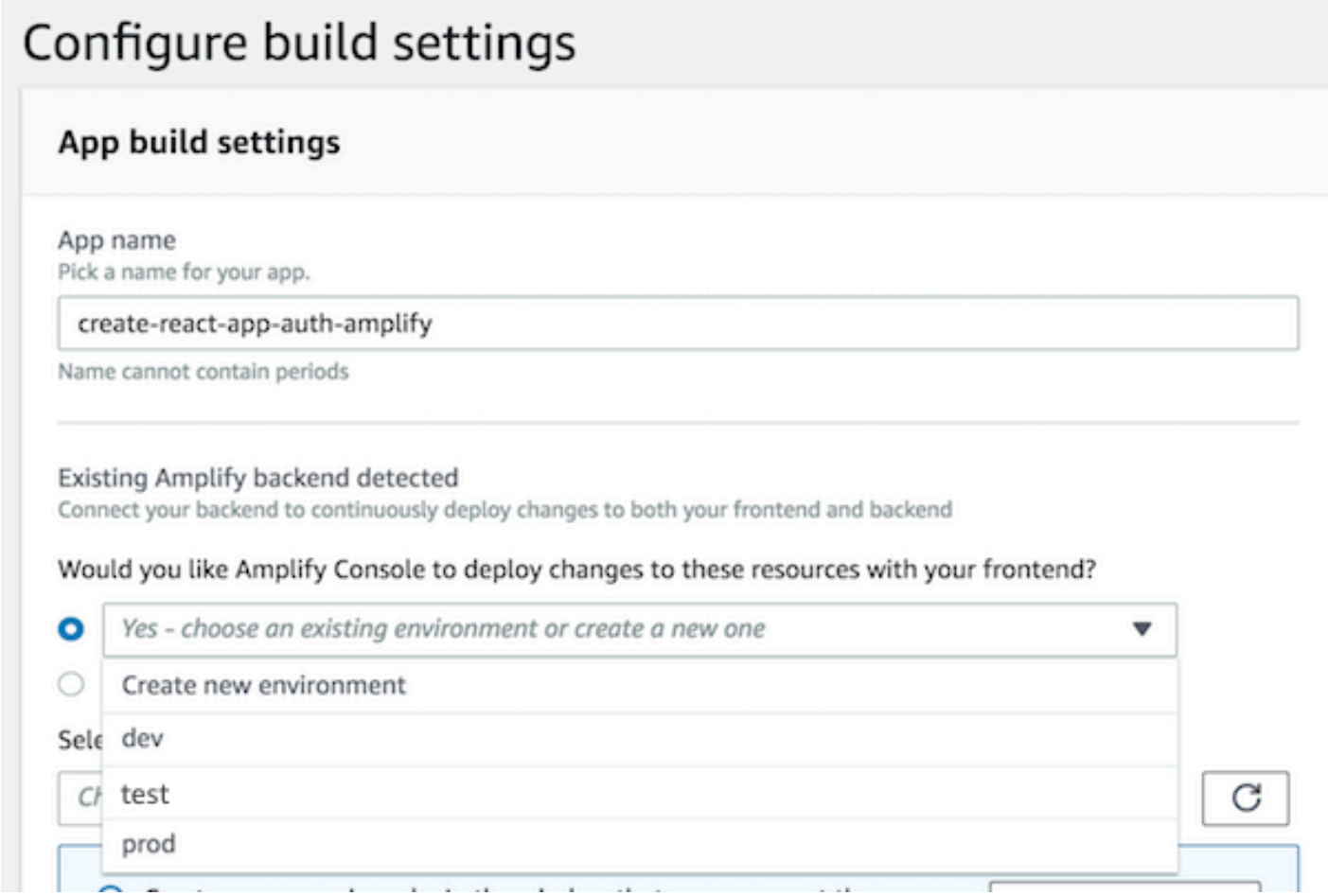
```
git commit -am 'Added dev, test, and prod environments'
git push origin main
```

5. の Amplify AWS マネジメントコンソール にアクセスして、現在のバックエンド環境を確認します。パンくずリストから1つ上のレベルに移動すると、[バックエンド環境] タブに作成されたすべてのバックエンド環境のリストが表示されます。

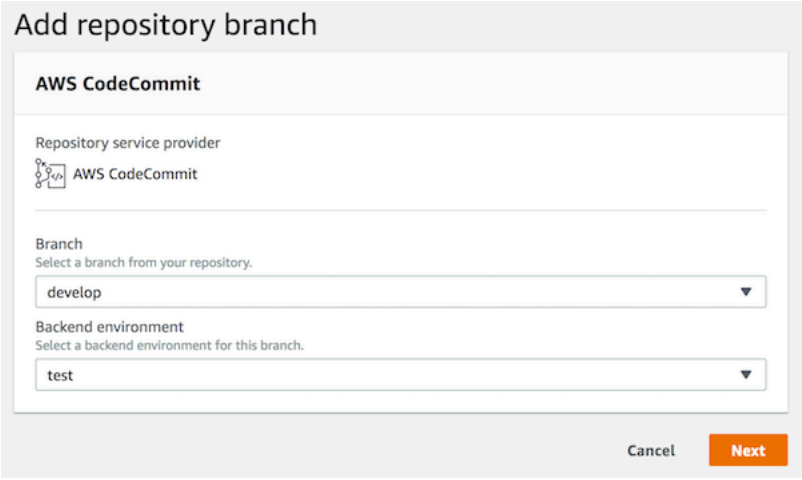



6. [フロントエンド環境] タブに切り替え、リポジトリプロバイダーと main ブランチを接続します。
7. ビルド設定ページで、既存のバックエンド環境を選択し、main ブランチとの継続的デプロイを設定します。リストから prod を選択し、サービスロールを Amplify に付与します。[保存してデプロ

イ] を選択します。ビルドが完了したら、https://main.appid.amplifyapp.com で利用可能な main ブランチのデプロイを取得します。




8. Amplify で develop ブランチを接続します (この時点で develop ブランチと main ブランチは同じであることを前提としています)。テストバックエンド環境を選択します。



9. これで Amplify のセットアップが完了しました。機能ブランチで新機能を使用することができます。ローカルワークステーションの dev バックエンド環境を使用して、バックエンド機能を追加します。

```
git checkout -b newinternet
amplify env checkout dev
amplify add api
...
amplify push
```

10機能を使用するための準備が整ったら、コードをコミットして、内部でレビューするためのプルリクエストを作成します。

```
git commit -am 'Decentralized internet v0.1'
git push origin newinternet
```

11変更内容をプレビューするには、Amplify コンソールに移動して機能ブランチを接続します。注: (Amplify CLI ではなく) システムに AWS CLI がインストールされている場合は、ターミナルからブランチを直接接続できます。アプリ ID を検索するには、[App settings] > [General] > AppARN: arn:aws:amplify:<region>:<region>:apps/<appid> の順に進みます。

```
aws amplify create-branch --app-id <appid> --branch-name <branchname>
aws amplify start-job --app-id <appid> --branch-name <branchname> --job-type RELEASE
```

12https://newinternet.appid.amplifyapp.com から機能にアクセスして、チームメイトと共有できるようになります。問題なければ、PR を develop ブランチにマージします。

```
git checkout develop
git merge newinternet
git push
```

13これにより、https://dev.appid.amplifyapp.com のブランチデプロイで、Amplify のバックエンドとフロントエンドを更新するビルドが開始されます。新機能を確認できるように、このリンクを社内の関係者と共有することができます。

14Git の Amplify から機能ブランチを削除し、クラウドからバックエンド環境を削除します (「amplify env checkout prod」および「amplify env add」を実行することで、いつでも新しい環境にスピンアップできます)。

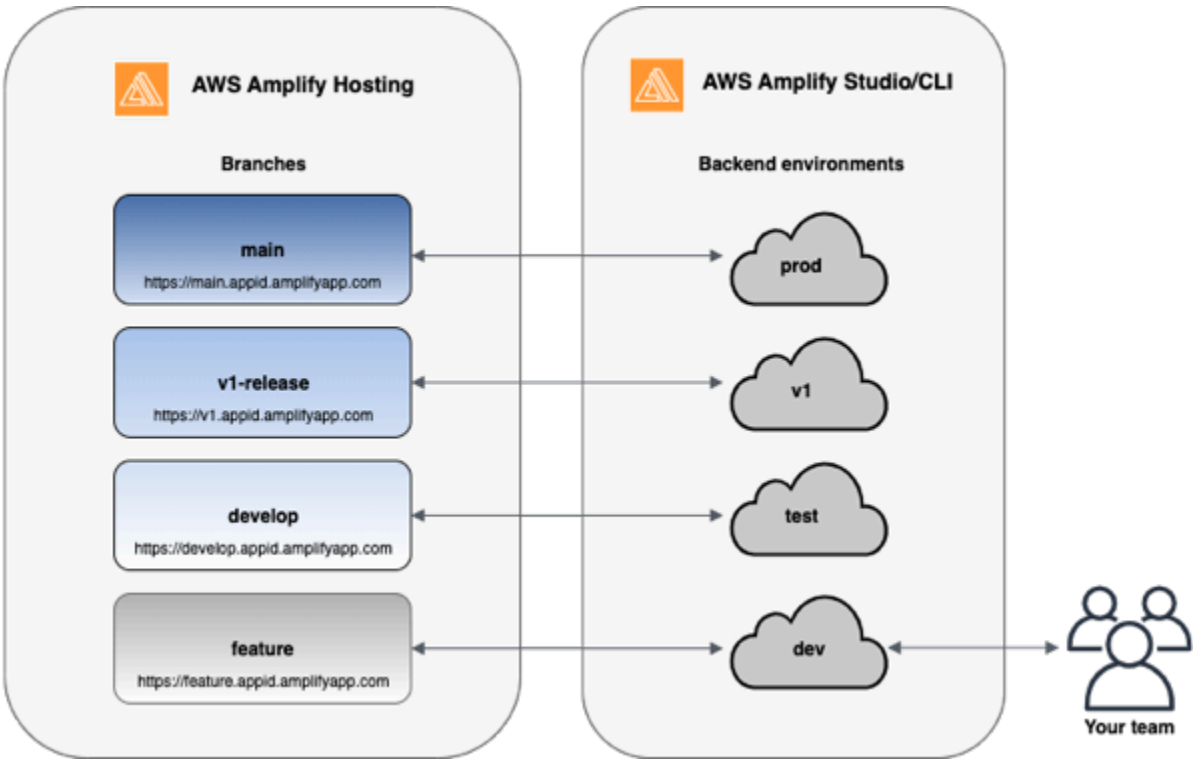
```
git push origin --delete newinternet
aws amplify delete-branch --app-id <appid> --branch-name <branchname>
```

```
amplify env remove dev
```

## GitFlow ワークフロー

GitFlow は2 つのブランチを使ってプロジェクトの履歴を記録します。main ブランチでは、リリースコードのみ追跡し、develop ブランチは、新機能の統合ブランチとして使用されます。GitFlow は、完了した作業から新しい開発を切り離すことによって、並列開発を簡素化します。新機能の開発 (機能や緊急ではないバグの修正など) は機能ブランチで行われます。開発者がコードのリリース準備が整ったことを確認すると、機能ブランチは統合開発ブランチにマージされます。main ブランチへの唯一のコミットは release ブランチと hotfix ブランチからのマージです (緊急のバグを修正するため)。

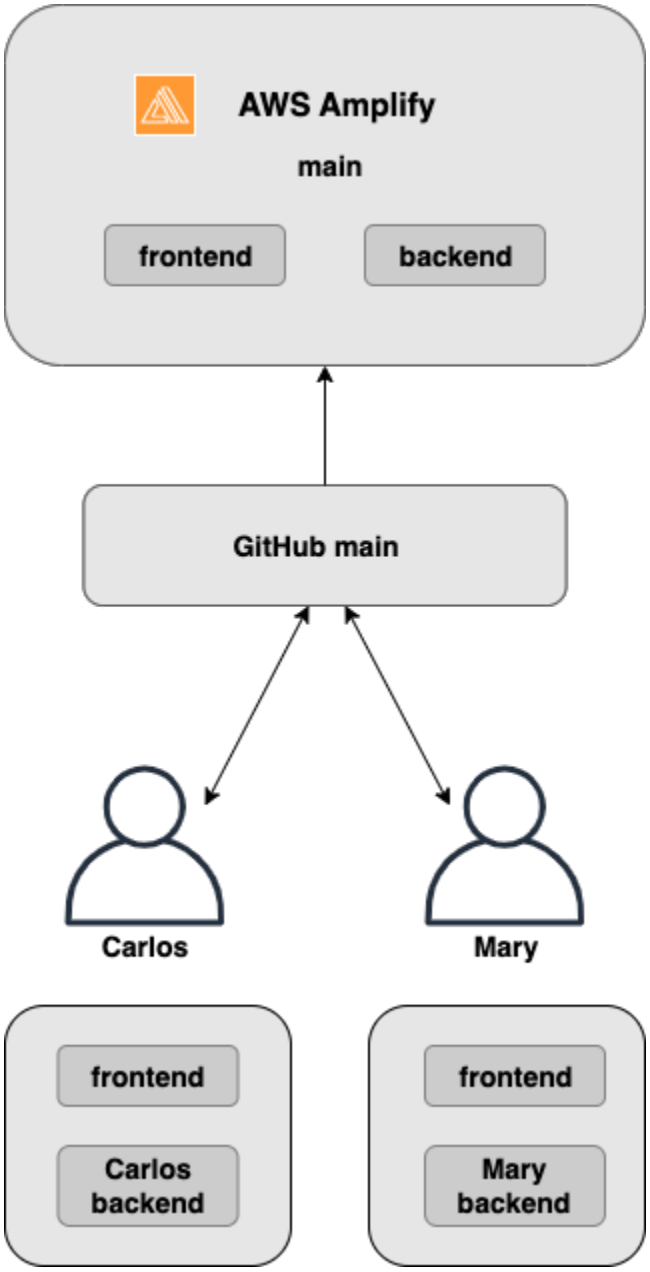
以下の図は GitFlow で推奨された設定を示します。上記の機能ブランチのワークフローのセクションで説明したプロセスと同じプロセスに従って行うことができます。




## 開発者ごとのサンドボックス

- チーム内の各開発者は、自分のローカルコンピュータとは別に、サンドボックス環境をクラウド内に作成すること。これにより、開発者は他のチームメンバーの変更を上書きすることなく互いに独立して作業することができます。

- Amplify の各ブランチには独自のバックエンドがあります。これにより、Amplify は、チームの開発者が自分のローカルコンピュータから本稼働環境に手動でバックエンドやフロントエンドをプッシュするのではなく、Git リポジトリを変更のデプロイ元となる唯一の真のソースとして使用します。




1. Amplify CLI をインストールして新しい Amplify プロジェクトを初期化します。

```
npm install -g @aws-amplify/cli
```

2. プロジェクト用の mary バックエンド環境を初期化します。プロジェクトがない場合は、create-react-app や Gatsby などのブートストラップツールを使用してプロジェクトを作成します。

```
cd next-unicorn
amplify init
 ? Do you want to use an existing environment? (Y/n): n
 ? Enter a name for the environment: mary
...
amplify push
```

3. 選択した Git リポジトリにコードをプッシュします (この例では、main にプッシュしたと仮定します)。

```
git commit -am 'Added mary sandbox'
git push origin main
```

4. リポジトリ > main を Amplify に接続します。
5. Amplify コンソールは、Amplify CLI によって作成されたバックエンド環境を検出します。ドロップダウンから [新しい環境を作成]を選択し、サービスロールを Amplify に付与します。[保存してデプロイ] を選択します。ビルドが完了したら、ブランチにリンクされている新しいバックエンド環境を持つ、https://main.appid.amplifyapp.com で利用可能な main ブランチのデプロイを取得します。
6. Amplify で develop ブランチを接続し (この時点で develop ブランチと main ブランチは同じであることを前提としています)、[作成] を選択します。

# パターンベースの機能ブランチのデプロイ

パターンベースのブランチデプロイでは、特定のパターンに一致するブランチを自動的に Amplify にデプロイできます。リリースに機能ブランチまたは GitFlow ワークフローを使用している製品チームは、**`release**`** のようなパターンを定義して、「release」で始まる Git ブランチを共有可能な URL に自動的にデプロイできるようになりました。

1. [アプリ設定] を選択し、次に [ブランチ設定] を選択します。
2. [ブランチ設定] ページで、[編集] を選択します。
3. [ブランチの自動検出] を選択して、パターンセットに一致する Amplify にブランチを自動的に接続します。

4. [ブランチ自動検出 - パターン] ボックスに、ブランチを自動的にデプロイするためのパターンを入力します。
   - ***** – リポジトリのすべてのブランチがデプロイされます。
   - **release*** –「release」という単語で始まるすべてのブランチをデプロイします。
   - **release*/** –「release /」パターンに一致するすべてのブランチをデプロイします。
   - 複数のパターンをカンマ区切りのリストで指定します。例えば、**release*, feature***。
5. [ブランチ自動検出アクセスコントロール] を選択して、自動的に作成されるすべてのブランチに対して自動パスワード保護を設定します。
6. Amplify バックエンドを使用して構築された Gen 1 アプリケーションの場合は、接続されたブランチごとに新しい環境を作成するか、すべてのブランチを既存のバックエンドにポイントするかを選択できます。
7. [保存] を選択します。

## カスタムドメインに接続されたアプリのパターンベースの機能ブランチデプロイ

Amazon Route 53 カスタムドメインに接続されたアプリに対して、パターンベースの機能ブランチデプロイを使用することができます。

- パターンベースの機能ブランチデプロイをセットアップする手順については、「Amazon Route 53 カスタムドメイン用の自動サブドメインの設定」を参照してください
- Amplify のアプリを Route 53 で管理されているカスタムドメインに接続する手順については、Amazon Route 53 が管理するカスタムドメインの追加 を参照してください
- Route 53 の詳細については、「What is Amazon Route 53」(Amazon Route 53 とは) を参照してください。

# Amplify 設定のビルド時の自動生成 (Gen 1 アプリ専用)

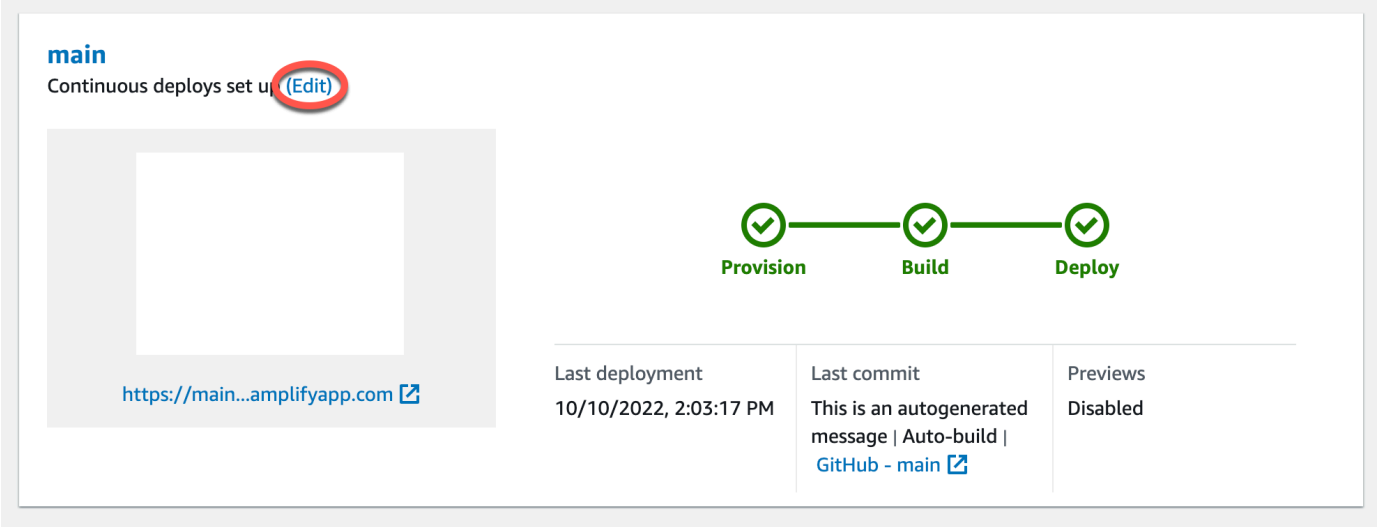
> **Note**
>
> このセクションの情報は、Gen 1 アプリケーション専用です。Gen 2 アプリの機能ブランチからインフラストラクチャとアプリケーションコードの変更を自動的にデプロイする場

合は、「Amplify ドキュメント」の「フルスタックブランチのデプロイ」を参照してください。

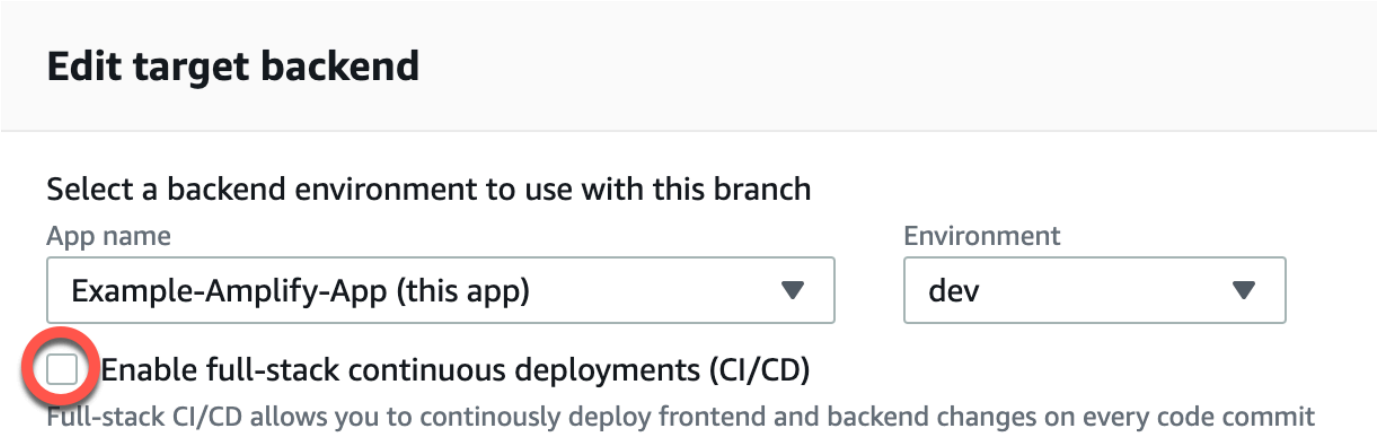
Amplify は、Gen 1 アプリ用 Amplify 設定 `aws-exports.js` ファイルのビルド時の自動生成をサポートしています。フルスタック CI/CD デプロイをオフにすることで、アプリによる `aws-exports.js` ファイルの自動生成が可能になり、ビルド時にバックエンドが更新されないようになります。

ビルド時に **`aws-exports.js`** を自動生成するには

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
2. 編集するアプリを選択します。
3. [ホスティング環境] タブを選択します。
4. 編集するブランチを見つけて [編集] を選択します。




5. 「ターゲットバックエンドの編集」ページで、「フルスタック継続的デプロイメント (CI/CD) を有効にする」のチェックを外して、このバックエンドのフルスタック CI/CD を無効にします。




6. 既存のサービスロールを選択して、アプリのバックエンドを変更するために必要な権限を Amplify に付与します。サービスロールを作成する必要がある場合は、[新しいロールを作成]を選択します。サービスロールの作成の詳細については、「バックエンドリソースをデプロイするアクセス許可を持つサービスロールの追加」を参照してください。
7. [保存] を選択します。Amplify は、次回アプリケーションを構築するときに、これらの変更を適用します。

# 条件付きバックエンドビルド (Gen 1 アプリ専用)

> **Note**
>
> このセクションの情報は、Gen 1 アプリケーション専用です。Amplify Gen 2 では、TypeScript ベースのコードファースト開発者エクスペリエンスが導入されています。したがって、この機能は Gen 2 バックエンドには必要ありません。

Amplify は、Gen 1 アプリ内のすべてのブランチでの条件付きバックエンドビルドをサポートします。条件付きバックエンドビルドを設定するには、環境変数 `AMPLIFY_DIFF_BACKEND` を `true` に設定します。条件付きバックエンドビルドを有効にすると、フロントエンドのみに変更が加えられるビルドをスピードアップするのに役立ちます。

差分ベースのバックエンドビルドを有効にすると、各ビルドの開始時に、Amplify はリポジトリ内の `amplify` フォルダーで差分を実行しようとします。Amplify が差分を見つけられない場合、バックエンドのビルドステップをスキップし、バックエンドリソースを更新しません。プロジェクトのリポジトリに `amplify` フォルダがない場合、Amplify は環境変数 `AMPLIFY_DIFF_BACKEND` の値を無視

します。環境変数 AMPLIFY_DIFF_BACKEND を設定する手順については、Gen 1 アプリケーションの diff ベースのバックエンドビルドの設定 を参照してください。

現在、バックエンドフェーズのビルド設定でカスタムコマンドが指定されている場合、条件付きバックエンドビルドは機能しません。これらのカスタムコマンドを実行したい場合は、アプリの amplify.yml ファイルにあるビルド設定のフロントエンドフェーズに移動する必要があります。amplify.yml ファイル更新の詳細については、「ビルド仕様に関するリファレンス」を参照してください。

# アプリ間で Amplify バックエンドを使用する (Gen 1 アプリ専用)

> **Note**
>
> このセクションの情報は、Gen 1 アプリケーション専用です。Gen 2 アプリのバックエンドリソースを共有する場合は、「Amplify ドキュメント」の「ブランチ間でリソースを共有する」を参照してください

Amplify では、特定のリージョンのすべての Gen 1 アプリで既存のバックエンド環境を簡単に再利用できます。これは、新しいアプリを作成したり、新しいブランチを既存のアプリに接続したり、既存のフロントエンドを更新して別のバックエンド環境を指すようにする際に活用できます。

## 新しいアプリを作成するときはバックエンドを再利用してください

新しい Amplify のアプリを作成するときにバックエンドを再利用するには

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
2. この例で使用する新規バックエンドを作成するには、以下を実行します。
   a. ナビゲーションペインで、[すべてのアプリ]を選択します。
   b. [新規アプリ]、[アプリを構築] の順に選択します。
   c. アプリの名前 (**Example-Amplify-App** など) を入力します。
   d. [デプロイを確認] を選択します。
3. フロントエンドを新しいバックエンドに接続するには、[ホスティング環境] タブを選択します。
4. Git プロバイダーを接続してから [ブランチを接続]を選択します。
5. 「リポジトリブランチを追加」ページの「最近更新されたリポジトリ」で、リポジトリ名を選択します。[ブランチ] では、リポジトリから接続するブランチを選択します。

6. [ビルドの設定]ページで、以下の操作を行います。

   a. [アプリ名] では、バックエンド環境の追加に使用するアプリを選択します。現在のアプリ、または現在のリージョンの他のアプリを選択できます。

   b. [環境] では、追加するバックエンド環境の名前を選択します。既存の環境を使用するか、新しい環境を作成できます。

   c. デフォルトでは、フルスタック CI/CD はオフになっています。フルスタック CI/CD をオフにすると、アプリはプルオンリーモードで実行されます。ビルド時に、Amplify はバックエンド環境を変更せずに `aws-exports.js` ファイルのみを自動的に生成します。

   d. 既存のサービスロールを選択して、アプリのバックエンドを変更するために必要な権限を Amplify に付与します。サービスロールを作成する必要がある場合は、[新しいロールを作成]を選択します。サービスロールの作成の詳細については、「バックエンドリソースをデプロイするアクセス許可を持つサービスロールの追加」を参照してください。

   e. [次へ] を選択します。

7. [保存してデプロイ] を選択します。

## ブランチを既存のアプリに接続するときはバックエンドを再利用してください。

ブランチを既存の Amplify アプリに接続するときにバックエンドを再利用するには

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
2. 新しいブランチを接続するアプリを選択します。
3. ナビゲーションペインで、[アプリの設定]、[全般]の順に選択します。
4. 「ブランチ」セクションで、[ブランチを接続] を選択します。
5. 「リポジトリブランチを追加」ページの [ブランチ] で、リポジトリから接続するブランチを選択します。
6. [アプリ名] では、バックエンド環境の追加に使用するアプリを選択します。現在のアプリ、または現在のリージョンの他のアプリを選択できます。
7. [環境] では、追加するバックエンド環境の名前を選択します。既存の環境を使用するか、新しい環境を作成できます。
8. アプリのバックエンドを変更するために必要な権限を Amplify に付与するサービスロールを設定する必要がある場合、コンソールからこのタスクを実行するように求められます。サービスロー

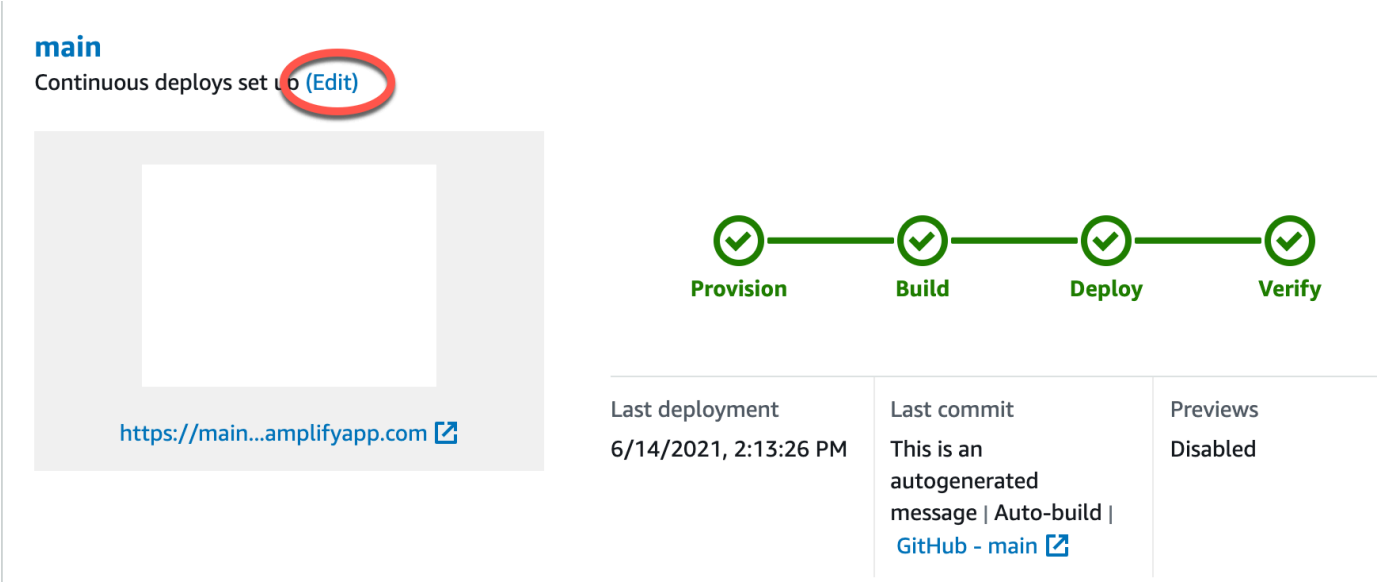
ルの作成の詳細については、「バックエンドリソースをデプロイするアクセス許可を持つサービスロールの追加」を参照してください。

9. デフォルトでは、フルスタック CI/CD はオフになっています。フルスタック CI/CD をオフにすると、アプリはプルオンリーモードで実行されます。ビルド時に、Amplify はバックエンド環境を変更せずに `aws-exports.js` ファイルのみを自動的に生成します。
10. [次へ] を選択します。
11. [保存してデプロイ] を選択します。

## 既存のフロントエンドを編集して、別のバックエンドを指すようにします

フロントエンド Amplify のアプリを編集して別のバックエンドを指すようにするには

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
2. バックエンドを編集するアプリを選択します。
3. [ホスティング環境] タブを選択します。
4. 編集するブランチを見つけて [編集] を選択します。




5. 「このブランチで使用するバックエンド環境を選択」ページの [アプリ名] で、バックエンド環境を編集したいフロントエンドアプリを選択します。現在のアプリ、または現在のリージョンの他のアプリを選択できます。
6. [バックエンド環境] では、追加するバックエンド環境の名前を選択します。
7. デフォルトでは、フルスタック CI/CD は有効になっています。このオプションのチェックを外すと、このバックエンドのフルスタック CI/CD がオフになります。フルスタック CI/CD をオフ

にすると、アプリはプルオンリーモードで実行されます。ビルド時に、Amplify はバックエンド環境を変更せずに `aws-exports.js` ファイルのみを自動的に生成します。

8. [保存] を選択します。Amplify は、次回アプリケーションを構築するときに、これらの変更を適用します。

# アプリケーションのバックエンドの構築

AWS Amplify を使用すると、デプロイ先のデータ、認証、ストレージ、フロントエンドホスティングを使用してフルスタックアプリケーションを構築できます AWS。

AWS Amplify Gen 2 では、バックエンドを定義するための TypeScript ベースのコードファースト開発者エクスペリエンスが導入されています。Amplify Gen 2 を使用して、バックエンドを構築し、アプリに接続する方法については、「Amplify ドキュメント」の「バックエンドの構築と接続」を参照してください。

CLI と Amplify Studio を使用して Gen 1 アプリのバックエンドを構築するためのドキュメントが必要な場合は、「Gen 1 Amplify ドキュメント」の「バックエンドの構築と接続」を参照してください。

トピック



## Gen 2 アプリのバックエンドを作成する

TypeScript ベースのバックエンドで Amplify Gen 2 フルスタックアプリケーションを作成する手順を説明するチュートリアルについては、「Amplify ドキュメント」の「概要」を参照してください。

## Gen 1 アプリのバックエンドを作成する

このチュートリアルでは、Amplify を使用してフルスタック CI/CD ワークフローを設定します。フロントエンドアプリを Amplify ホスティングにデプロイします。次に、Amplify Studio を使用してバックエンドを作成します。最後に、クラウドバックエンドをフロントエンドアプリに接続します。

### 前提条件

このチュートリアルを始める前に、次の前提条件を完了してください。

にサインアップする AWS アカウント

> まだ AWS のお客様でない場合は、オンラインの手順に従って を作成 AWS アカウントする必要があります。サインアップすると、Amplify やアプリケーションで使用できるその他の AWS サービスにアクセスできます。

Git リポジトリを作成する

Amplify は、GitHub、Bitbucket、GitLab、および をサポートしています AWS CodeCommit。アプリケーションを Git リポジトリにプッシュします。

Amplify コマンドラインインターフェイス (CLI)のインストール

手順については、「Amplify Framework ドキュメント」の「Amplify CLI のインストール」を参照してください。

## ステップ 1: フロントエンドをデプロイする

この例で、使用したい既存のフロントエンドアプリが Git リポジトリにある場合は、フロントエンドアプリをデプロイする手順に進むことができます。

この例に使用するのに新しいフロントエンドアプリケーションを作成する必要がある場合は、「Create React App ドキュメント」の「Create React App」の手順に従います。

フロントエンドアプリをデプロイするには

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
2. 「すべてのアプリ」ページで、[新規アプリ]を選択し、右上隅の[ウェブアプリをホスト]を選択します。
3. GitHub、Bitbucket、GitLab、または AWS CodeCommit リポジトリプロバイダーを選択し、続行を選択します。
4. Amplify は git リポジトリへのアクセスを許可します。GitHub リポジトリの場合、Amplify は GitHub Apps 機能を使用して Amplify へのアクセスを承認できるようになりました。

   GitHub App のインストールと認証の詳細については、GitHub リポジトリへの Amplify アクセスの設定 をご参照ください。
5. 「リポジトリブランチを追加」ページで、以下の操作を行います。

   a. 「最近更新されたリポジトリ」リストで、接続するリポジトリの名前を選択します。

   b. ブランチリストで、接続するリポジトリブランチの名前を選択します。

   c. [次へ] を選択します。
6. [ビルド設定の構成]ページで、[次へ]を選択します。
7. [確認]ページ で、[保存してデプロイ]を選択します。デプロイが完了したら、`amplifyapp.com` デフォルトドメインにアプリを表示できます。

> **Note**
>
> Amplify のアプリケーションのセキュリティを強化するために、amplifyapp.com ドメインはパブリックサフィックスリスト (PSL) に登録されています。セキュリティを強化するために、Amplify アプリケーションのデフォルトドメイン名に機密性の高いCookieを設定する必要がある場合は、__Host-プレフィックスの付いたCookieを使用することをお勧めします。このプラクティスは、クロスサイトリクエストフォージェリ (CSRF) 攻撃からドメインを防ぐ際に役立ちます。詳細については、Mozilla 開発者ネットワークの「Set-Cookie」ページを参照してください。

## ステップ 2: バックエンドを作成する

Amplify ホスティングにフロントエンドアプリをデプロイしたので、バックエンドを作成できます。次のステップに従って、シンプルなデータベースと GraphQL API エンドポイントを含むバックエンドを作成します。

バックエンドを作成するには

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
2. [すべてのアプリ] ページで、ステップ 1 で作成したアプリを選択します。
3. アプリのホームページで [バックエンド環境] タブを選択し、[はじめに] を選択します。これにより、デフォルトのステージング環境の設定プロセスが開始されます。
4. セットアップが完了したら、[Studio を起動する] を選択して Amplify Studio のステージングバックエンド環境にアクセスします。

Amplify Studio は、バックエンドを作成および管理し、フロントエンド UI 開発を加速するためのビジュアルインターフェイスです。Amplify の詳細については、Amplify Studio ドキュメントを参照してください。

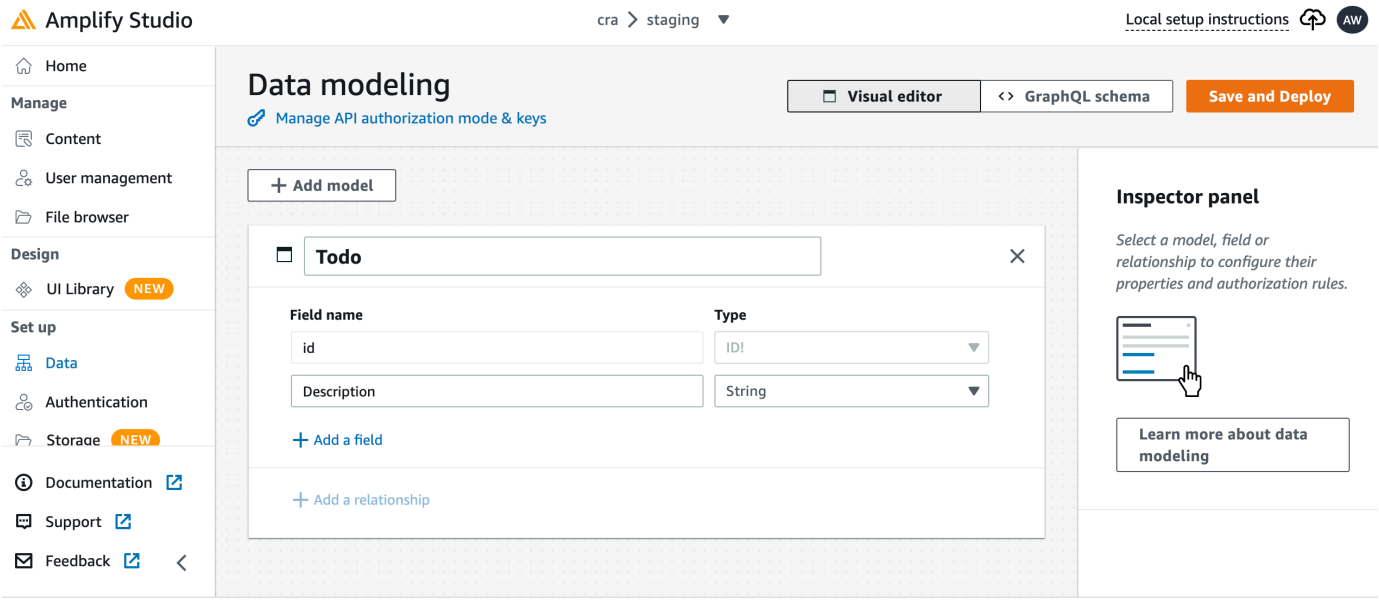
次のステップに従って、Amplify Studio ビジュアルバックエンドビルダーインターフェイスを使用して簡単なデータベースを作成します。

データモデルの作成

1. アプリのステージング環境のホームページで、[データモデルを作成] を選択します。データモデルデザイナーが開きます。

2. [データモデリング] ページで [モデルの追加] を選択します。
3. タイトルに、**Todo** と入力します。
4. [フィールドを追加]を選択します。
5. [フィールド名]に **Description** と入力します。

   次のスクリーンショットは、データモデルがデザイナーでどのように表示されるかを示す例です。




6. [保存してデプロイ] を選択します。
7. Amplify ホスティングコンソールに戻ると、ステージング環境のデプロイが進行中です。

デプロイ中、Amplify Studio はバックエンドに必要なすべての AWS リソースを作成します。これには、データにアクセスするための AWS AppSync GraphQL API や、Todo 項目をホストするための Amazon DynamoDB テーブルが含まれます。Amplify は CloudFormation を使用してバックエンドをデプロイします。これにより、バックエンド定義を infrastructure-as-code として保存できます。

## ステップ 3: バックエンドをフロントエンドに接続する

フロントエンドをデプロイし、データモデルを含むクラウドバックエンドを作成したので、次はそれらを接続する必要があります。以下の手順に従って、Amplify CLI を使用してバックエンド定義をローカルアプリプロジェクトに取り込みます。

クラウドバックエンドをローカルフロントエンドに接続するには

1. ターミナルウィンドウを開き、ローカルプロジェクトのルートディレクトリに移動します。

2. ターミナルウィンドウで次のコマンドを実行し、赤いテキストをプロジェクト固有のアプリ ID とバックエンド環境名に置き換えます。

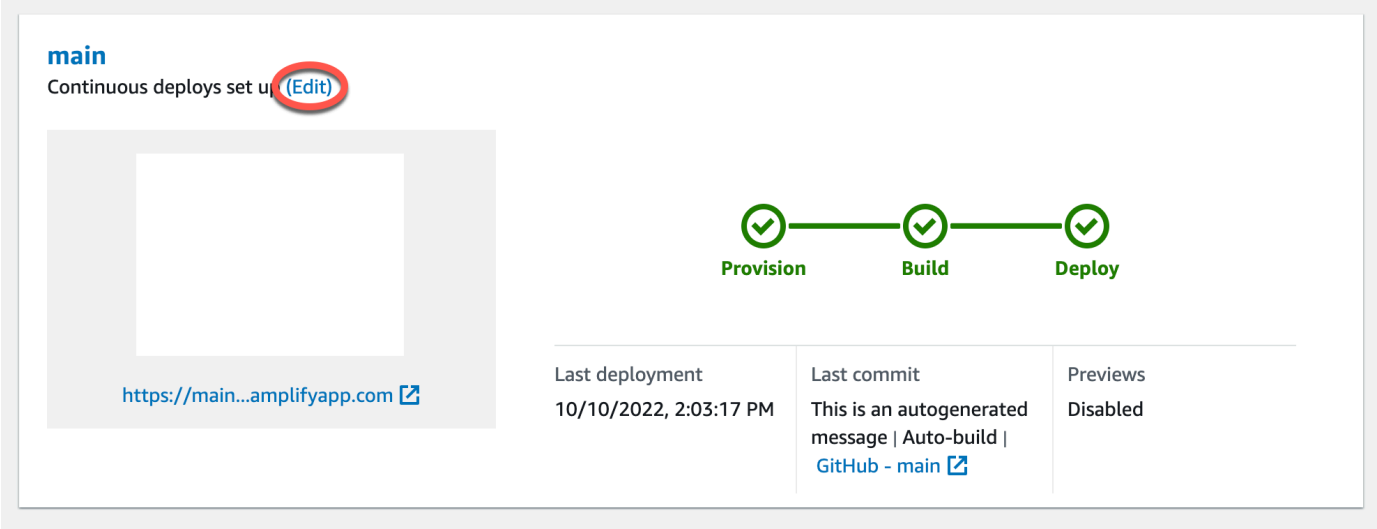
```
amplify pull --appId abcd1234 --envName staging
```

3. ターミナルウィンドウの指示に従って、プロジェクトの設定を完了します。

これで、継続的デプロイメントワークフローにバックエンドを追加するようにビルドプロセスを設定できるようになりました。以下の手順に従って、Amplify ホスティングコンソールのフロントエンドブランチとバックエンドを接続します。

フロントエンドアプリブランチとクラウドバックエンドを接続するには

1. アプリのホームページで [ホスティング環境] タブを選択します。
2. メインブランチを見つけて [編集] を選択します。




3. 「ターゲットバックエンドの編集」ウィンドウの「環境」で、接続するバックエンドの名前を選択します。この例では、ステップ 2 で作成したステージングバックエンドを選択します。

   デフォルトでは、フルスタック CI/CD は有効になっています。このオプションのチェックを外すと、このバックエンドのフルスタック CI/CD がオフになります。フルスタック CI/CD をオフにすると、アプリはプルオンリーモードで実行されます。ビルド時に、Amplify はバックエンド環境を変更せずに `aws-exports.js` ファイルのみを自動的に生成します。

4. 次に、アプリのバックエンドを変更するために必要な権限を Amplify に付与するサービスロールを設定する必要があります。既存のサービスロールを使用するか、新しいロールを作成できま

す。手順については、「バックエンドリソースをデプロイするアクセス許可を持つサービスロールの追加」を参照してください。

5. サービスロールを追加したら、「ターゲットバックエンドの編集」ウィンドウに戻り、[保存] を選択します。
6. ステージングバックエンドをフロントエンドアプリのメインブランチに接続し終えるには、プロジェクトの新規ビルドを実行します。

   次のいずれかを行います。

   - Git リポジトリから、コードをプッシュして Amplify コンソールでビルドを開始します。
   - Amplify コンソールで、アプリのビルド詳細ページに移動し、[このバージョンを再デプロイ]を選択します。

# 次の手順

## 機能ブランチのデプロイのセットアップ

推奨ワークフローに従って、複数のバックエンド環境でフィーチャーブランチのデプロイを設定します。

## Amplify Studio でフロントエンド UI を作成する

Studio を使用して、すぐに使用できる一連の UI コンポーネントでフロントエンド UI を構築してから、アプリのバックエンドにその UI を接続します。詳細とチュートリアルについては、「Amplify Framework ドキュメント」の Amplify Studio 用ユーザーガイドを参照してください。

# 高度なデプロイ機能

この章では、Amplify ホスティングワークフローを強化する高度なデプロイ機能について説明します。これらの機能により、チームがデプロイをより効果的に管理し、コード品質を確保し、開発ライフサイクルを通じてセキュリティを維持するのに役立つ追加の制御と機能が提供されます。

パスワード認証を使用して機能ブランチを保護し、リリースされていない機能へのアクセスを制限する方法について説明します。プルリクエストのウェブプレビューを有効にして、コードを本番ブランチにマージする前に、一意のプレビュー URL の変更を確認します。Cypress フレームワークを使用してエンドツーエンドのテストを設定し、コードを本番環境にプッシュする前にリグレッションをキャッチします。Amplify にデプロイするためのボタン機能は使用できなくなりましたが、Amplify ホスティングを使用してリポジトリから直接アプリケーションを簡単にデプロイできます。

トピック

- Amplify アプリケーションのブランチへのアクセスの制限
- プルリクエストの Web プレビュー
- Amplify アプリケーションのエンドツーエンドの Cypress テストの設定
- GitHub プロジェクトを共有するための [Deploy to Amplify] ボタンの使用

## Amplify アプリケーションのブランチへのアクセスの制限

未発表の機能に取り組んでいる場合は、機能ブランチをパスワードで保護し、特定のユーザーへのアクセス制限を施すことができます。ブランチにアクセス制御を設定すると、ユーザーがブランチの URL にアクセスしようとすると、ユーザー名とパスワードの入力を求められます。

個々のブランチに適用するパスワードや接続されているすべてのブランチにグローバルに適用するパスワードを設定することもできます。ブランチレベルとグローバルレベルの両方でアクセスコントロールが有効になっている場合、ブランチレベルのパスワードはグローバル (アプリケーション) レベルのパスワードよりも優先されます。

Amplify は、パスワードで保護されたリソースにアクセスしようとしている失敗したリクエストをスロットリングします。この動作は、辞書攻撃や、アクセスコントロールの背後にあるデータを読み取ろうとする試みなどからアプリケーションを保護します。

Amplify アプリのブランチへのアクセスを制限するパスワードを設定するには、次の手順に従います。

機能ブランチにパスワードを設定するには

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
2. 機能ブランチのパスワードを設定したいアプリを選択します。
3. ナビゲーションペインで、[ホスティング] を選択して、[アクセスコントロール] を選択します。
4. [アクセス制御設定]セクションで、[アクセスを管理] を選択します。
5. [アクセスコントロールを管理する] ページで、次の操作のいずれかを行います。
   - 接続しているすべてのブランチに適用されるユーザー名とパスワードを設定するには
     - [すべてのブランチのアクセスを管理する] を有効にします。たとえば、main ブランチ、dev ブランチ、feature ブランチをが接続されている場合は、すべてのブランチに同じユーザー名およびパスワードを適用できます。
   - 個々のブランチに適用されるユーザー名とパスワードを設定するには
     a. [すべてのブランチ のアクセス管理] をオフにします。
     b. 管理するブランチを検索します。[アクセス設定] では、[必須の制限付きパスワード] を選択します。
     c. [ユーザー名] には、ユーザー名を入力します。
     d. [パスワード] には、パスワードを入力します。
   - [保存] を選択します。
6. サーバーサイドレンダリング (SSR) アプリのアクセス制御を管理している場合は、Git リポジトリから新しいビルドを実行してアプリを再デプロイします。この手順は、Amplify がアクセス制御設定を適用できるようにするために必要です。

# プルリクエストの Web プレビュー

Web プレビューは、開発チームや品質保証 (QA) チームに、コードをプロダクションブランチやインテグレーションブランチにマージする前に、プルリクエスト (PR) からの変更をプレビューする方法を提供します。プルリクエストを使うと、リポジトリ内のブランチにプッシュした変更を他の人に伝えることができます。プルリクエストが開かれたら、変更の可能性についてコラボレーターと話し合ったりレビューしたり、変更がベースブランチにマージされる前にフォローアップコミットを追加したりできます。

Web プレビューは、リポジトリに対して行われたすべてのプルリクエストを、メインサイトが使用している URL とはまったく異なる固有のプレビュー URL にデプロイします。Amplify CLI または

Amplify Studio を使用してバックエンド環境をプロビジョニングしたアプリでは、すべてのプルリクエスト (プライベート Git リポジトリのみ) は一時的なバックエンドを作成し、そのバックエンドは PR がクローズされると削除されます。

アプリのウェブプレビューがオンになっている場合、各 PR はアプリごとに 50 ブランチという Amplify クォータにカウントされます。このクォータを超えないように、必ず PR をクローズします。クォータの詳細については、「Amplify ホスティング Service Quotas」を参照してください。

> **Note**
>
> 現在、 `AWS_PULL_REQUEST_ID` 環境変数は、 をリポジトリプロバイダー AWS CodeCommit として使用する場合は使用できません。

Web プレビューのセキュリティ

セキュリティ上の理由から、プライベートリポジトリを持つすべてのアプリで Web プレビューを有効にできますが、パブリックリポジトリを持つすべてのアプリでは有効にできません。Git リポジトリが公開されている場合、IAM サービスロールを必要としないアプリにのみプレビューを設定できます。例えば、バックエンドを備えたアプリや`WEB_COMPUTE`ホスティングプラットフォームにデプロイされるアプリには IAM サービスロールが必要です。そのため、これらの種類のアプリのリポジトリが公開されている場合、ウェブプレビューを有効にすることはできません。Amplify はこの制限を適用して、アプリのIAMロール権限を使用して実行されるような任意のコードを第三者が送信することを防ぎます。

SSR コンピューティングロールを使用してパブリックリポジトリ内のアプリケーションで Web プレビューが有効になっている場合は、ロールにアクセスできるブランチを慎重に管理する必要があります。アプリケーションレベルのロールを使用しないことをお勧めします。代わりに、ブランチレベルでコンピューティングロールをアタッチする必要があります。これにより、特定のリソースへのアクセスを必要とするブランチにのみアクセス許可を付与できます。詳細については、「AWS リソースへのアクセスを許可する SSR コンピューティングロールの追加」を参照してください。

## プルリクエストの Web プレビューを有効にする

GitHub リポジトリに保存されているアプリの場合、Web プレビューでは Amplify GitHub App を使用してリポジトリにアクセスします。アクセス用に OAuth を使用して GitHub リポジトリからデプロイした既存の Amplify アプリのウェブプレビューを有効にする場合は、まず Amplify GitHub App を使用するようにアプリを移行する必要があります。移行手順については、既存の OAuth アプリを Amplify GitHub アプリに移行するを参照してください。

プルリクエストの Web プレビューを有効にするには

1. [ホスティング] の次に [プレビュー] を選択します。

   > **Note**
   >
   > [プレビュー]は、アプリが継続的デプロイ用に設定され、Git リポジトリに接続されている場合にのみ[アプリ設定]メニューに表示されます。この種類のデプロイの手順については、「既存のコードを使い始める」を参照してください。

2. GitHub リポジトリの場合のみ、次の手順を実行して Amplify GitHub App をアカウントにインストールして認可します。

   a. 「GitHub App をインストールしてプレビューを有効にする」ウィンドウで、「GitHub アプリをインストール」を選択します。

   b. Amplify GitHub アプリを設定したい GitHub アカウントを選択します。

   c. GitHub.com で、アカウントのリポジトリ権限を設定するページが開きます。

   d. 次のいずれかを行います。

      - インストールをすべてのリポジトリに適用するには、「全てのリポジトリ」を選択します。
      - 選択した特定のリポジトリのみにインストールを制限するには、[リポジトリのみ選択] を選択します。Web プレビューを有効にするアプリのリポジトリを、選択するリポジトリに必ず含めてください。

   e. [保存] を選択します。

3. リポジトリのプレビューを有効にしたら、Amplify コンソールに戻って特定のブランチのプレビューを有効にします。[プレビュー] ページで、リストからブランチを選択し、[設定を編集] を選択します。

4. [プレビュー設定を管理] ページ、[プルリクエストのプレビュー] をオンにしてください。[Confirm] (確認) を選択します。

5. フルスタックのアプリケーションでは、以下のいずれかを行ってください。

   - [プルリクエストごとに新しいバックエンド環境を作成] を選択します。このオプションを使うと、本番環境に影響を及ぼすことなく変更をテストできます。
   - [このブランチのすべてのプルリクエストを既存の環境に限定する] を選択します。

6. [確認] を選択します。

次にブランチのプルリクエストを送信すると、Amplify はPRをビルドしてプレビューURLにデプロイします。プルリクエストがクローズされると、プレビュー URL は削除され、プルリクエストにリンクされている一時的なバックエンド環境はすべて削除されます。GitHub リポジトリの場合のみ、GitHub アカウントのプルリクエストから URL のプレビューに直接アクセスできます。

## サブドメインによる Web プレビューアクセス

プルリクエストのウェブプレビューには、Amazon Route 53 が管理するカスタムドメインに接続されている Amplify アプリのサブドメインでアクセスできます。プルリクエストがクローズされると、そのプルリクエストに関連するブランチとサブドメインは自動的に削除されます。これは、アプリにパターンベースの機能ブランチのデプロイを設定した後のウェブプレビューのデフォルト動作です。自動サブドメインをセットアップする手順については、「Amazon Route 53 カスタムドメイン用の自動サブドメインの設定」を参照してください。

# Amplify アプリケーションのエンドツーエンドの Cypress テストの設定

Amplify のアプリのテストフェーズでエンドツーエンド (E2E) テストを実行して、コードを本番環境にプッシュする前にリグレッションを捕捉できます。テストフェーズはビルド仕様 YAML で設定できます。現在、ビルド中に実行できるのは Cypress のテストフレームワークだけです。

Cypress は JavaScript ベースのテストフレームワークで、ブラウザ上で E2E テストを実行することができます。E2E テストの設定方法を示すチュートリアルについては、ブログ記事「Amplify によるフルスタック CI/CD デプロイのためのエンドツーエンド Cypress テストの実行」を参照してください。

## 既存の Amplify アプリケーションへの Cypress テストの追加

Amplify コンソールでアプリのビルド設定を更新することで、既存のアプリに Cypress テストを追加できます。YAML ビルド仕様には、Amplify でビルドの実行に使用される一連のビルドコマンドと関連設定が含まれます。この `test` ステップを使用して、ビルド時にテストコマンドを実行します。E2E テストの場合、Amplify ホスティングは Cypress とのより緊密な統合を提供しているため、テスト用の UI レポートを生成できます。

以下のリストでは、テスト設定とその使用方法について説明しています。

テスト前

Cypress テストの実行に必要な依存関係をインストールします。Amplify ホスティングは、mochawesome を使用してテスト結果を確認するためのレポートを生成し、wait-on を使用して、ビルド中にローカルホストサーバーをセットアップします。

test

コマンド cypress を実行し、mochawesome を使用してテストを実行します。

テスト後

mochawesome のレポートは JSON 出力から生成されます。Yarn を使用している場合は、このコマンドをサイレントモードで実行して mochawesome のレポートを生成する必要があることに注意してください。Yarn では以下のコマンドを使用できます。

```
yarn run --silent mochawesome-merge cypress/report/mochawesome-report/
mochawesome*.json > cypress/report/mochawesome.json
```

artifacts>baseDirectory

テストを実行するディレクトリ。

artifacts>configFilePath

生成されたテストレポートデータ。

artifacts>files

生成されたアーティファクト (スクリーンショットとビデオ) をダウンロードできます。

以下のビルド仕様 amplify.yml ファイルからの抜粋例は、Cypress によるテストをアプリに追加する方法を示しています。

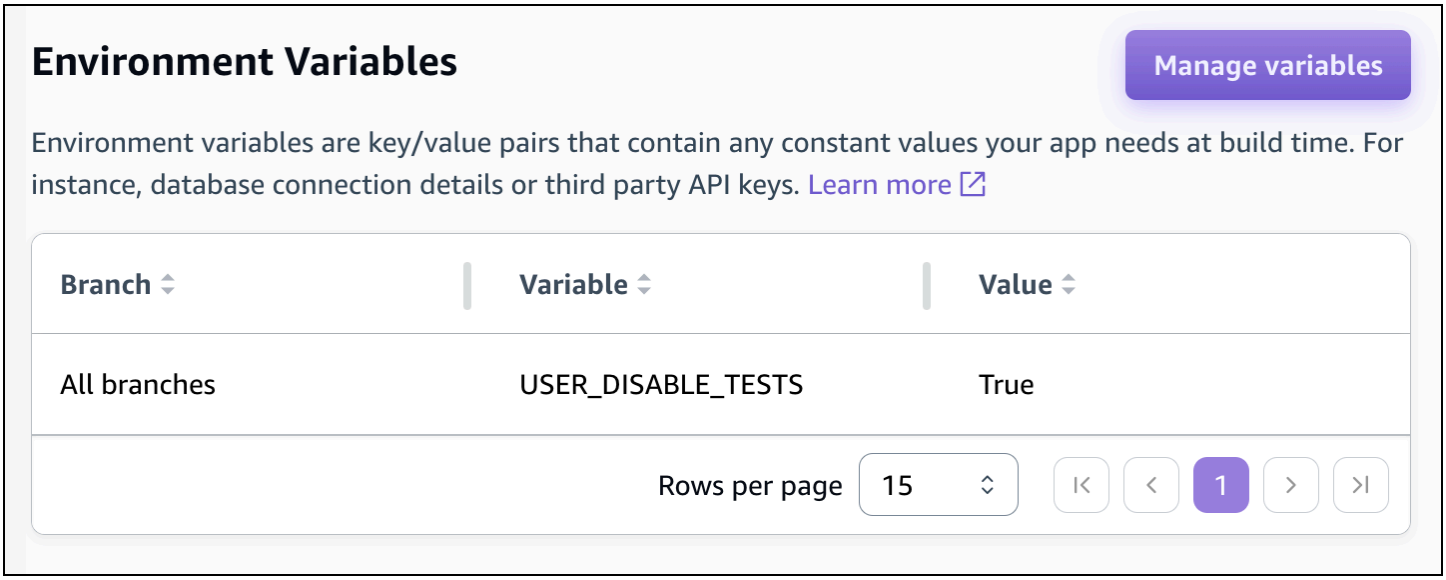
```
test:
  phases:
    preTest:
      commands:
        - npm ci
        - npm install -g pm2
        - npm install -g wait-on
        - npm install mocha mochawesome mochawesome-merge mochawesome-report-generator
        - pm2 start npm -- start
        - wait-on http://localhost:3000
```

```
    test:
      commands:
        - 'npx cypress run --reporter mochawesome --reporter-options
 "reportDir=cypress/report/mochawesome-
report,overwrite=false,html=false,json=true,timestamp=mmddyyyy_HHMMss"'
    postTest:
      commands:
        - npx mochawesome-merge cypress/report/mochawesome-report/mochawesome*.json >
 cypress/report/mochawesome.json
        - pm2 kill
  artifacts:
    baseDirectory: cypress
    configFilePath: '**/mochawesome.json'
    files:
      - '**/*.png'
      - '**/*.mp4'
```

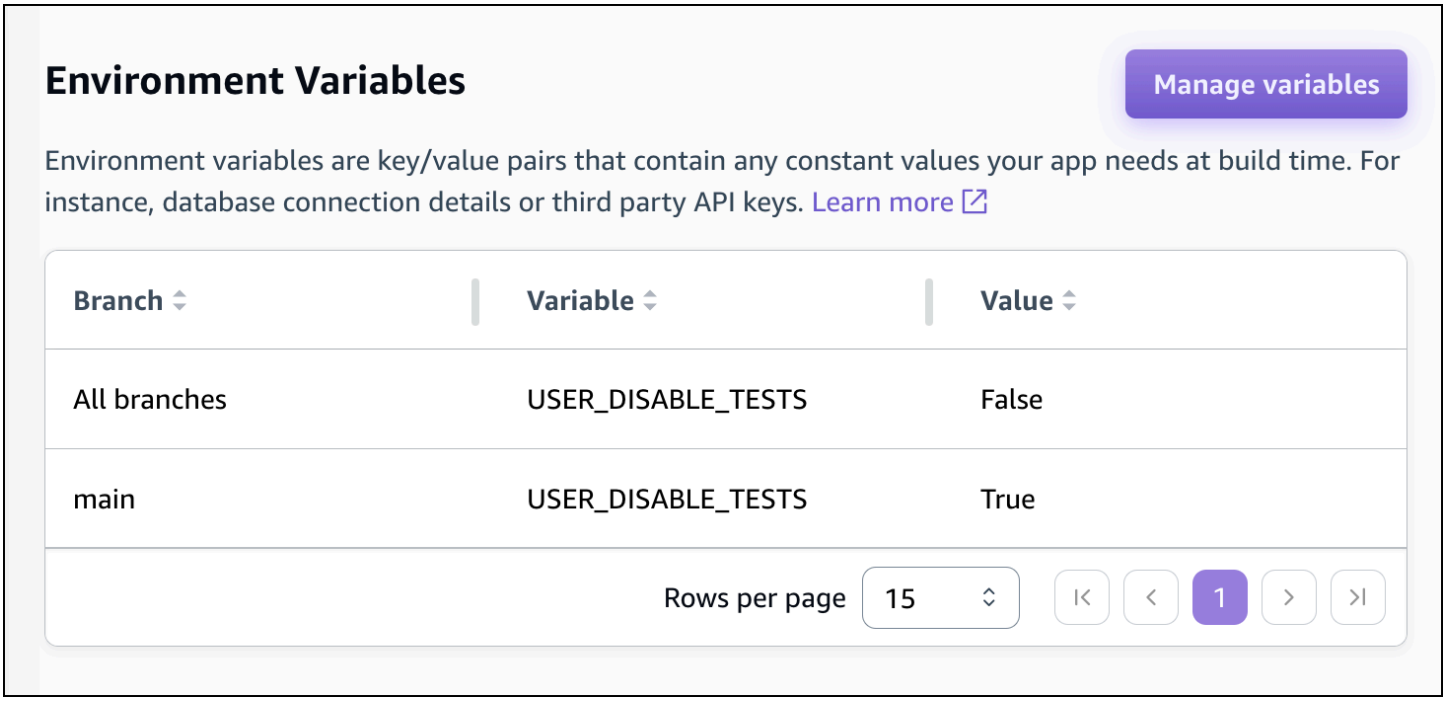
## Amplify アプリケーションまたはブランチのテストをオフにする

テスト構成が `amplify.yml` のビルド設定に追加されると、`test` ステップはすべてのビルド、すべてのブランチで実行されます。テストの実行をグローバルに無効にしたり、特定のブランチのみのテストを実行したりする場合は、ビルド設定を変更せずに環境変数 USER_DISABLE_TESTS を使用できます。

すべてのブランチのテストをグローバルに無効にするには、すべてのブランチに対して `true` の値を持つ環境変数 USER_DISABLE_TESTS を追加します。次のスクリーンショットは、すべてのブランチでテストが無効になっている Amplify のコンソールの「環境変数」セクションを示しています。



特定のブランチのテストを無効にするには、全てのブランチに対して `false` の値を持つ環境変数 USER_DISABLE_TESTS を追加し、無効にする各ブランチに `true` の値でオーバーライドを追加します。次のスクリーンショットでは、「メイン」ブランチではテストが無効になっており、他のすべてのブランチではテストが有効になっています。




この変数を使用してテストを無効にすると、ビルド中にテストステップが完全にスキップされます。テストを再度有効にするには、この値を `false` に設定するか、環境変数を削除してください。

# GitHub プロジェクトを共有するための [Deploy to Amplify] ボタンの使用

> **Important**
>
> [Amplify ホスティングにデプロイ] ボタンを使用したワンクリックデプロイは使用できなくなりました。リポジトリからデプロイするには、Amplify ホスティングで新しいアプリケーションを作成してください。手順については、「Amplify ホスティングへのアプリのデプロイの概要」を参照してください。

[Amplify コンソールへのデプロイ]ボタンを使用すると、GitHub プロジェクトをパブリックに共有するか、チーム内で共有することができます。以下は、ボタンの画像です。


# リポジトリまたはブログへの [Amplify ホスティングにデプロイ] ボタンの追加

ボタンを GitHub の README.md ファイル、ブログ記事、または HTML を表示するその他のマークアップページに追加します。ボタンには次のような 2 つのコンポーネントがあります。

1. URL `https://oneclick.amplifyapp.com/button.svg` にある SVG 画像
2. GitHub リポジトリへのリンクを含む Amplify コンソールの URL。リポジトリの URL (`https://github.com/username/repository` など) をコピーすることも、特定のフォルダー (`https://github.com/username/repository/tree/branchname/folder` など) へのディープリンクを指定することもできます。Amplify ホスティングは、リポジトリのデフォルトのブランチをデプロイします。アプリが接続されたら、ブランチを追加接続することができます。

次の例を使用して、GitHub README.md などのマークダウンファイルにボタンを追加します。`https://github.com/username/repository` をリポジトリの URL に置き換えます。

```
[![amplifybutton](https://oneclick.amplifyapp.com/button.svg)](https://
console.aws.amazon.com/amplify/home#/deploy?repo=https://github.com/username/
repository)
```

次の例を使用して、任意の HTML ドキュメントにボタンを追加します。`https://github.com/username/repository` をリポジトリの URL に置き換えます。

```
<a href="https://console.aws.amazon.com/amplify/home#/deploy?repo=https://github.com/
username/repository">
    <img src="https://oneclick.amplifyapp.com/button.svg" alt="Deploy to Amplify
 Hosting">
</a>
```

# Amplify アプリケーションのリダイレクトと書き換えの設定

リダイレクトを使用すると、ウェブサーバーで 1 つの URL から別の URL にナビゲーションを再ルートすることができます。リダイレクトは一般的に、URL の外観をカスタマイズする、リンクが壊れないようにする、アドレスを変更せずにアプリまたはサイトのホスティング場所を移動する、ウェブアプリで必要なフォームへの URL を変更するといった理由で使用されます。

## Amplify がサポートするリダイレクトについて

Amplify は、コンソールで次のリダイレクトタイプをサポートしています。

恒久的なリダイレクト (301)

301 リダイレクトは、ウェブアドレスの送信先への恒久的な変更を目的としています。新しい送信先アドレスには、元のアドレスの検索エンジンのランキング履歴が適用されます。リダイレクトはクライアント側で行われるため、ブラウザのナビゲーションバーには、リダイレクト後の送信先アドレスが表示されます。

301 リダイレクトを使用する一般的な理由を以下に示します。

- ページのアドレスが変更されたときにリンク切れを回避する。
- ユーザーがアドレスに予測可能なタイプミスをしたときにリンクが壊れるのを防ぐ。

一時的なダイレクト (302)

302 リダイレクトは、ウェブアドレスの送信先への一時的な変更を目的としています。新しい送信先アドレスには、元のアドレスの検索エンジンのランキング履歴は適用されません。リダイレクトはクライアント側で行われるため、ブラウザのナビゲーションバーには、リダイレクト後の送信先アドレスが表示されます。

302 リダイレクトを使用する一般的な理由を以下に示します。

- 元のアドレスの修正中に迂回先を提供する。
- ユーザーインターフェイスの A/B 比較用のテストページを提供する。

> **Note**
>
> アプリが予期しない 302 レスポンスを返す場合、エラーの原因はアプリのリダイレクトとカスタムヘッダーの設定を変更したことが原因と考えられます。この問題を解決するに

は、カスタムヘッダーが有効であることを確認し、アプリのデフォルトの 404 リライトルールを再度有効にします。

書き換え (200)

200 リダイレクト (書き換え) は、まるで元のアドレスから配信されたかのように、送信アドレスからのコンテンツを表示することを目的としています。検索エンジンのランキング履歴は、引き続き元のアドレスに適用されます。リダイレクトはサーバー側で行われるため、ブラウザのナビゲーションバーには、リダイレクト後の元のアドレスが表示されます。200 リダイレクトを使用する一般的な理由を以下に示します。

- サイトのアドレスを変更せずに、サイト全体を新しいホスティング場所にリダイレクトする。
- 単一ページのウェブアプリケーション (SPA) へのすべてのトラフィックを index.html ページにリダイレクトし、クライアント側のルーター機能で処理する。

Not Found (404)

404 リダイレクトは、リクエストが存在しないアドレスを指している場合に発生します。リクエストされたページではなく、404 の送信先ページが表示されます。404 リダイレクトが発生する一般的な理由を以下に示します。

- ユーザーが誤った URL を入力したときにリンク切れメッセージが表示されないようにする。
- ウェブアプリケーションの存在しないページへのリクエストをクライアント側のルーター機能で処理する index.html ページに向けること。

## リダイレクトの順序について

リダイレクトはリストの上から順に適用されます。順序に、意図した効果があることを確認してください。例えば、次のリダイレクト順序では、/docs/ 以下の特定のパスに対するすべてのリクエストが /documents/の下の同じパスにリダイレクトされます。ただし/docs/specific-filename.htmlは/documents/different-filename.htmlにリダイレクトされます。

```
/docs/specific-filename.html /documents/different-filename.html 301
/docs/<*> /documents/<*>
```

次のリダイレクト順序では、specific-filename.html から different-filename.html へのリダイレクトは無視されます。

```
/docs/<*> /documents/<*>
/docs/specific-filename.html /documents/different-filename.html 301
```

# Amplify がクエリパラメータを転送する方法について

クエリパラメータを使用して、URL の一致をより詳細に制御できます。Amplify は、以下の例外を除いて、301 および302 リダイレクトのすべてのクエリパラメータを宛先パスに転送します。

- 元のアドレスに特定の値に設定されたクエリ文字列が含まれている場合、Amplify はクエリパラメータを転送しません。この場合、リダイレクトは指定されたクエリ値を持つ宛先 URL へのリクエストにのみ適用されます。
- マッチングルールの宛先アドレスにクエリパラメータがある場合、クエリパラメータは転送されません。たとえば、リダイレクトの宛先アドレスが`https://example-target.com?q=someParam`の場合、クエリパラメータは渡されません。

# Amplify コンソールでのリダイレクトの作成と編集

Amplify コンソールで、アプリのリダイレクトを作成および編集できます。開始する前に、リダイレクトの構成要素に関する以下の情報が必要です。

元のアドレス

ユーザーがリクエストしたアドレス。

送信先アドレス

ユーザーに表示されるコンテンツを実際に提供するアドレス。

リダイレクトの種類

タイプには、恒久的なリダイレクト (301)、一時的なリダイレクト (302)、書き換え (200)、または not found (404) などがあります。

2 文字の国コード (オプション)

アプリのユーザーエクスペリエンスを地域別にセグメント化するために含めることができる値。

Amplify コンソールでリダイレクトを作成するには

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
2. リダイレクトを作成するアプリを選択します。
3. ナビゲーションペインで [ホスティング] を選択し、[書き換えとリダイレクト] を選択します。
4. [書き換えとリダイレクト] ページで、[リダイレクトの管理] を選択します。
5. JSON エディターの [書き換えとリダイレクト] で、リダイレクトを手動で追加または更新します。

   a. `source` には、ユーザーがリクエストした元のアドレスを指定します。
   b. `status` には、リダイレクトのタイプを指定します。
   c. `target` には、コンテンツをユーザーにレンダリングする宛先アドレスを指定します。
   d. (オプション) `condition` には、2 文字の国コード条件を入力します。
6. [保存] を選択します。

## サンプルリファレンスをリダイレクトして書き換える

このセクションでは、一般的なリダイレクトのさまざまな例について説明します。

> **Important**
>
> ドメイン固有のリダイレクトは、ソースフィールドのパスコンポーネントをサポートしていません。
> サポートされている:
>
> - `"source": "https://example.com"` (パスは自動的に追加されます)
>
> サポートされていない:
>
> - `"source": "https://example.com/specific-path"`
> - `domain+path` の組み合わせのルールは現在サポートされていません。
>
> 代替パターン
> ドメイン固有のパスリダイレクトには、次を使用します。
>
> 1. ドメインのみのルールを分離する (パスは自動的に追加されます)

2. 条件付きロジックを使用したパスのみのルール
3. 複数のルールの組み合わせ

これらの例を使用すれば、Amplify コンソール JSON エディタで独自のリダイレクトと書き換えを作成するための JSON 構文を理解できるようになります。

> **Note**
>
> 元のアドレスのドメイン照合では、大文字と小文字は区別されません。

トピック



## シンプルなリダイレクトと書き換え

次の例を使用して、特定のページを新しいアドレスに恒久的にリダイレクトすることができます。

| 元のアドレス | 送信先アドレス | リダイレクトの種類 | 国コード |
| --- | --- | --- | --- |
| `/original.html` | `/destination.html` | `permanent redirect (301)` | |

JSON 形式

```
[
```

```
  {
    "source": "/original.html",
    "status": "301",
    "target": "/destination.html",
    "condition": null
  }
]
```

次の例を使用して、フォルダ内の任意のパスを別のフォルダ内の同じパスにリダイレクトできます。

| 元のアドレス | 送信先アドレス | リダイレクトの種類 | 国コード |
| --- | --- | --- | --- |
| /docs/<*> | /documents/<*> | permanent redirect (301) | |

JSON 形式

```
[
  {
    "source": "/docs/<*>",
    "status": "301",
    "target": "/documents/<*>",
    "condition": null
  }
]
```

次の例を使用して、書き換えとしてすべてのトラフィックを index.html にリダイレクトすることができます。このシナリオでは、書き換えによって元のアドレスに到達したようにユーザーに表示されます。

| 元のアドレス | 送信先アドレス | リダイレクトの種類 | 国コード |
| --- | --- | --- | --- |
| /<*> | /index.html | rewrite (200) | |

JSON 形式

```
[
  {
```

```
    "source": "/<*>",
    "status": "200",
    "target": "/index.html",
    "condition": null
  }
]
```

次の例を使用して、書き換えによって、ユーザーに表示されるサブドメインを変更することができます。

| 元のアドレス | 送信先アドレス | リダイレクトの種類 | 国コード |
| --- | --- | --- | --- |
| https://mydomain.com | https://www.mydomain.com | rewrite (200) | |

JSON 形式

```
[
  {
    "source": "https://mydomain.com",
    "status": "200", "target": "https://www.mydomain.com",
    "condition": null
  }
]
```

次の例を使用して、パスプレフィックスを付けた別のドメインにリダイレクトできます。

| 元のアドレス | 送信先アドレス | リダイレクトの種類 | 国コード |
| --- | --- | --- | --- |
| https://mydomain.com | https://www.mydomain.com/documents | temporary redirect (302) | |

JSON 形式

```
[
```

```
  {
    "source": "https://mydomain.com",
    "status": "302",
    "target": "https://www.mydomain.com/documents/",
    "condition": null
  }
]
```

次の例を使用して、見つからないフォルダのパスをカスタムの 404 ページにリダイレクトすることができます。

| 元のアドレス | 送信先アドレス | リダイレクトの種類 | 国コード |
| --- | --- | --- | --- |
| /<*> | /404.html | not found (404) | |

JSON 形式

```
[
  {
    "source": "/<*>",
    "status": "404",
    "target": "/404.html",
    "condition": null
  }
]
```

> **Important**
>
> ドメインベースのソースルール ("https://domain.com/path" など) のパスコンポーネントはサポートされていないため、ルールは エラーなしで無視されます。

## 単一ページのウェブアプリケーション (SPA) のリダイレクト

大抵の SPA フレームワークは、HTML5 history.pushState() をサポートしており、サーバーリクエストを行わなくても、ブラウザの場所を変更できます。この方法は、ユーザーがルート (または / index.html) から作業を開始する問題ありませんが、他のページに直接ナビゲートする場合は失敗します。

次の例では、正規表現を使用して、正規表現で指定されている特定のファイル拡張子を除き、すべてのファイルをindex.html に200回書き換えるように設定します。

| 元のアドレス | 送信先アドレス | リダイレクトの種類 | 国コード |
| --- | --- | --- | --- |
| </^[^.]+$\|\.(?!(css\|gif\|ico\|jpg\|js\|png\|txt\|svg\|woff\|woff2\|ttf\|map\|json\|webp)$)([^.]+$)/> | /index.html | 200 | |

JSON 形式

```
[
  {
    "source": "</^[^.]+$|\.(?!(css|gif|ico|jpg|js|png|txt|svg|woff|woff2|ttf|map|json|
webp)$)([^.]+$)/>",
    "status": "200",
    "target": "/index.html",
    "condition": null
  }
]
```

## リバースプロキシの書き換え

次の例では、ドメインが変更されていないように見えるように、別の場所からプロキシコンテンツへの書き換えを使用しています。HTTPS は、リバースプロキシでサポートされている唯一のプロトコルです。

| 元のアドレス | 送信先アドレス | リダイレクトの種類 | 国コード |
| --- | --- | --- | --- |
| /images/<*> | https://images.otherdomain.com/<*> | rewrite (200) | |

JSON 形式

```
[
  {
    "source": "/images/<*>",
    "status": "200",
    "target": "https://images.otherdomain.com/<*>",
    "condition": null
  }
]
```

## 末尾のスラッシュとクリーン URL

about.htmlの代わりにaboutのようなきれいな URL 構造を作成するために、Hugo などの静的サイトジェネレーターは index.html (/about/index.html) を含むページのディレクトリを生成します。Amplify では、必要に応じて末尾のスラッシュを追加することによって、クリーン URL を自動的に作成します。以下の表は、さまざまなシナリオをまとめたものです。

| ブラウザでのユーザー入力 | アドレスバーの URL | 提供されたドキュメント |
| --- | --- | --- |
| /about | /about | /about.html |
| /about (when about.html returns 404) | /about/ | /about/index.html |
| /about/ | /about/ | /about/index.html |

## プレースホルダー

次の例を使用して、フォルダ構造内のパスを別のフォルダ内の一致する構造にリダイレクトすることができます。

| 元のアドレス | 送信先アドレス | リダイレクトの種類 | 国コード |
| --- | --- | --- | --- |
| /docs/<year>/<month>/<date>/<itemid> | /documents/<year>/<month>/<date>/<itemid> | permanent redirect (301) | |

JSON 形式

```
[
  {
    "source":  "/docs/<year>/<month>/<date>/<itemid>",
    "status": "301",
    "target": "/documents/<year>/<month>/<date>/<itemid>",
               "condition": null
   }
]
```

## クエリ文字列とパスパラメータ

> **Warning**
>
> シークレット、認証情報、または機密データをパスまたはクエリパラメータとして URL に含めないでください。これらの値は、Amplify アプリケーションのアクセスログでプレーンテキストで表示できます。

次の例を使用して、元のアドレスのクエリ文字列要素の値と一致する名前のパスをフォルダにリダイレクトできます。

| 元のアドレス | 送信先アドレス | リダイレクトの種類 | 国コード |
| --- | --- | --- | --- |
| /docs?id=<my-blog-id-value | /documents/<my-blog-post-id-value> | permanent redirect (301) | |

JSON 形式

```
[
  {
    "source": "/docs?id=<my-blog-id-value>",
    "status": "301",
    "target": "/documents/<my-blog-id-value>",
    "condition": null
  }
```

```
]
```

> **Note**
>
> Amplify は、すべてのクエリ文字列パラメータを 301 および 302 リダイレクトの宛先パスに転送します。ただし、この例のように、元のアドレスに特定の値に設定されたクエリ文字列が含まれている場合、Amplify はクエリパラメータを転送しません。この場合、リダイレクトは指定されたクエリ値idを持つ宛先アドレスへのリクエストにのみ適用されます。

次の例を使用して、特定レベルのフォルダ構造で見つからないパスをすべて、指定フォルダ内の index.html にリダイレクトすることができます。

| 元のアドレス | 送信先アドレス | リダイレクトの種類 | 国コード |
| --- | --- | --- | --- |
| /documents/<folder>/<child-folder>/<grand-child-folder> | /documents/index.html | not found (404) | |

JSON 形式

```
[
  {
    "source": "/documents/<x>/<y>/<z>",
    "status": "404",
    "target": "/documents/index.html",
    "condition": null
  }
]
```

## リージョンベースのリダイレクト

次の例を使用して、リージョンに基づきリクエストをリダイレクトできます。

| 元のアドレス | 送信先アドレス | リダイレクトの種類 | 国コード |
| --- | --- | --- | --- |
| /documents | /documents/us/ | temporary redirect (302) | <US> |

JSON 形式

```
[
  {
    "source": "/documents",
    "status": "302",
    "target": "/documents/us/",
    "condition": "<US>"
  }
]
```

## リダイレクトと書き換えでのワイルドカード式の使用

ワイルドカード式の <*> は、リダイレクトまたは書き換えの元のアドレスで使用できます。この式は元のアドレスの末尾に配置し、一意である必要があります。Amplify は、複数のワイルドカード式を含む元のアドレスを無視するか、異なる配置で使用します。

以下は、ワイルドカード式を使用した有効なリダイレクトの例です。

| 元のアドレス | 送信先アドレス | リダイレクトの種類 | 国コード |
| --- | --- | --- | --- |
| /docs/<*> | /documents/<*> | permanent redirect (301) | |

次の 2 つ例は、ワイルドカード式による 無効な リダイレクトです。

| 元のアドレス | 送信先アドレス | リダイレクトの種類 | 国コード |
| --- | --- | --- | --- |
| /docs/<*>/content | /documents/<*>/content | permanent redirect (301) | |

| 元のアドレス | 送信先アドレス | リダイレクトの種類 | 国コード |
| --- | --- | --- | --- |
| /docs/<*>/content/<*> | /documents/<*>/content/<*> | permanent redirect (301) | |

# Amplify アプリケーションでの環境変数の使用

環境変数は、アプリケーションの設定に追加して Amplify ホスティングで使用できるようにするキーと値のペアです。ベストプラクティスとして、環境変数を使用してアプリケーションの設定データを公開することができます。追加した環境変数はすべて、不正アクセスを防ぐために暗号化されています。

Amplify は、作成する環境変数に次の制約を適用します。

- Amplify では、AWS プレフィックスを使用して環境変数名を作成することはできません。このプレフィックスは Amplify の内部でのみ使用されるよう予約されています。
- 環境変数の値は 5500 文字を超えることはできません。

> **Important**
>
> シークレットの保存に環境変数を使用しないでください。Gen 2 アプリの場合は、Amplify コンソールの [シークレット管理] 機能を使用します。詳細については、「Amplify ドキュメント」の「シークレットと環境変数」を参照してください。Gen 1 アプリの場合、Parameter Store AWS Systems Manager を使用して作成された環境シークレットにシークレットを保存します。詳細については、「環境シークレットの管理」を参照してください。

## Amplify の環境変数のリファレンス

以下の環境変数は、Amplifyコンソールからデフォルトでアクセス可能です。

| 変数名 | 説明 | 値の例 |
| --- | --- | --- |
| _BUILD_TIMEOUT | ビルドのタイムアウト時間 (分単位)。<br><br>最小値は 5 です。<br><br>最大値は 120 です。 | 30 |

| 変数名 | 説明 | 値の例 |
| --- | --- | --- |
| _LIVE_UPDATES | ツールは最新バージョンにアップグレードされます。 | `[{"name":"Amplify CLI","pkg":"@aws-amplify/cli","type":"npm","version":"latest"}]` |
| USER_DISABLE_TESTS | ビルド中はテストステップはスキップされます。アプリ内のすべてのブランチまたは特定のブランチのテストを無効にできます。<br><br>この環境変数は、ビルドフェーズ中にテストを実行するアプリに使用されます。この変数の設定の詳細については、「Amplify アプリケーションまたはブランチのテストをオフにする」を参照してください。 | `true` |
| AWS_APP_ID | 現在のビルドのアプリ ID | `abcd1234` |
| AWS_BRANCH | 現在のビルドのブランチ名 | `main`, `develop`, `beta`, `v2.0` |
| AWS_BRANCH_ARN | 現在のビルドのブランチ Amazon リソースネーム (ARN)。 | `aws:arn:amplify:us-west-2:123456789012:appname/branch/...` |
| AWS_CLONE_URL | git リポジトリの内容を取得するために使用されるクローン URL | `git@github.com:<user-name>/<repo-name>.git` |

| 変数名 | 説明 | 値の例 |
| --- | --- | --- |
| AWS_COMMIT_ID | 現在のビルドのコミット ID<br><br>再ビルドの「HEAD」 | abcd1234 |
| AWS_JOB_ID | 現在のビルドのジョブ ID。<br><br>これには、「0」のパディングが含まれるため、長さは常に同じになります。 | 0000000001 |
| AWS_PULL_REQUEST_ID | プルリクエスト Web プレビュービルドのプルリクエスト ID。<br><br>この環境変数は、 をリポジトリプロバイダー AWS CodeCommit として使用する場合は使用できません。 | 1 |
| AWS_PULL_REQUEST_SOURCE_BRANCH | Amplify コンソールのアプリケーションブランチに送信されるプルリクエストプレビューの機能ブランチの名前。 | featureA |
| AWS_PULL_REQUEST_DESTINATION_BRANCH | 機能ブランチのプルリクエストが送信される Amplify コンソール内のアプリケーションブランチの名前。 | main |
| AMPLIFY_AMAZON_CLIENT_ID | Amazon クライアント ID | 123456 |
| AMPLIFY_AMAZON_CLIENT_SECRET | Amazon クライアントシークレット | example123456 |

| 変数名 | 説明 | 値の例 |
| --- | --- | --- |
| AMPLIFY_FACEBOOK_CLIENT_ID | Facebook クライアント ID | `123456` |
| AMPLIFY_FACEBOOK_CLIENT_SECRET | Facebook クライアントシークレット | `example123456` |
| AMPLIFY_GOOGLE_CLIENT_ID | Google クライアント ID | `123456` |
| AMPLIFY_GOOGLE_CLIENT_SECRET | Google クライアントシークレット | `example123456` |
| AMPLIFY_DIFF_DEPLOY | 差分ベースのフロントエンドデプロイを有効または無効にします。詳細については、「差分ベースのフロントエンドビルドとデプロイの設定」を参照してください。 | `true` |
| AMPLIFY_DIFF_DEPLOY_ROOT | 差分ベースのフロントエンドデプロイ比較に使用するパスで、リポジトリのルートを基準にしています。 | `dist` |
| AMPLIFY_DIFF_BACKEND | 差分ベースのバックエンドビルドを有効または無効にします。これは Gen 1 アプリにのみ適用されます。詳細については、Gen 1 アプリケーションの diff ベースのバックエンドビルドの設定を参照してください。 | `true` |

| 変数名 | 説明 | 値の例 |
| --- | --- | --- |
| AMPLIFY_BACKEND_PULL_ONLY | Amplify は、この環境変数を管理します。これは Gen 1 アプリにのみ適用されます。詳細については、既存のフロントエンドを編集して、別のバックエンドを指すようにしますを参照してください。 | `true` |
| AMPLIFY_BACKEND_APP_ID | Amplify は、この環境変数を管理します。これは Gen 1 アプリにのみ適用されます。詳細については、既存のフロントエンドを編集して、別のバックエンドを指すようにしますを参照してください。 | `abcd1234` |
| AMPLIFY_SKIP_BACKEND_BUILD | ビルド仕様にバックエンドセクションがなく、バックエンドビルドを無効にする場合は、この環境変数を`true`に設定してください。これは Gen 1 アプリにのみ適用されます。 | `true` |
| AMPLIFY_ENABLE_DEBUG_OUTPUT | この変数を `true` に設定して、スタックトレースをログに出力します。これは、バックエンドビルドエラーのデバッグに役立ちます。 | `true` |
| AMPLIFY_MONOREPO_APP_ROOT | monorepo アプリのアプリルートを指定するために使用するパスで、リポジトリのルートを基準にしています。 | `apps/react-app` |

| 変数名 | 説明 | 値の例 |
| --- | --- | --- |
| AMPLIFY_USERPOOL_ID | 認証用にインポートされた Amazon Cognito ユーザープールの ID | `us-west-2_example` |
| AMPLIFY_WEBCLIENT_ID | ウェブアプリケーションが使用するアプリケーションクライアントの ID。<br><br>アプリケーションクライアントは、AMPLIFY_USERPOOL_ID の環境変数で指定された Amazon Cognito ユーザープールにアクセスできるように設定する必要があります。 | `123456` |
| AMPLIFY_NATIVECLIENT_ID | ネイティブアプリケーションが使用するアプリケーションクライアントの ID。<br><br>アプリケーションクライアントは、AMPLIFY_USERPOOL_ID の環境変数で指定された Amazon Cognito ユーザープールにアクセスできるように設定する必要があります。 | `123456` |
| AMPLIFY_IDENTITYPOOL_ID | Amazon Cognitoアイデンティティプールの ID。 | `example-identitypool-id` |
| AMPLIFY_PERMISSIONS_BOUNDARY_ARN | Amplify で作成されたすべての IAM ロールのアクセス許可の境界として使用する IAM ポリシーの ARN です。 | `arn:aws:iam::123456789012:policy/example-policy` |

| 変数名 | 説明 | 値の例 |
| --- | --- | --- |
| AMPLIFY_DESTRUCTIVE_UPDATES | この環境変数を true に設定すると、データ損失を引き起こす可能性のあるスキーマ操作で GraphQL API を更新できます。 | `true` |

> **Note**
>
> `AMPLIFY_AMAZON_CLIENT_ID` および `AMPLIFY_AMAZON_CLIENT_SECRET`環境変数は OAuth トークンであり、 AWS アクセスキーとシークレットキーではありません。

## フロントエンドフレームワーク環境変数

独自の環境変数をサポートするフロントエンドフレームワークを使用してアプリを開発している場合、これらは Amplify コンソールで設定する環境変数と同じではないことを理解することが重要です。例えば、React (プレフィックス REACT_APP) や Gatsby (プレフィックス GATSBY) では、ランタイム環境変数を作成して、それらのフレームワークがフロントエンドのプロダクションビルドに自動的にバンドルすることができます。これらの環境変数を使用して値を保存した場合の効果を理解するには、使用しているフロントエンドフレームワークのドキュメントを参照してください。

API キーなどの機密性の高い値を、フロントエンドフレームワークのプレフィックスが付いた環境変数内に保存することはベストプラクティスではないため、あまりお勧めしません。

## 環境変数の設定

次の手順に従って、Amplify コンソールのアプリケーションの環境変数を設定します。

> **Note**
>
> 環境変数は、アプリが継続的なデプロイ用に設定され、gitリポジトリに接続されている場合にのみ、Amplify コンソールのアプリ設定メニューに表示されます。この種類のデプロイの手順については、「既存のコードを使い始める」を参照してください。

環境変数の設定方法

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
2. Amplify コンソールで、[ホスティング]、[環境変数] の順に選択します。
3. 「環境変数」ページで、[変数の管理] を選択します。
4. [変数] には、お客様のキーを入力します。[値]に値を入力します。デフォルトでは、環境変数は、Amplify によってすべてのブランチに適用されるため、新しいブランチへの接続時に変数を再入力する必要はありません。
5. (オプション) 環境変数をブランチ専用にカスタマイズするには、以下のようにブランチの上書きを追加します。

   a. [アクション]、[変数の上書きを追加する] の順に選択します。

   b. これで、ブランチに固有の一連の環境変数ができました。
6. [保存] を選択します。

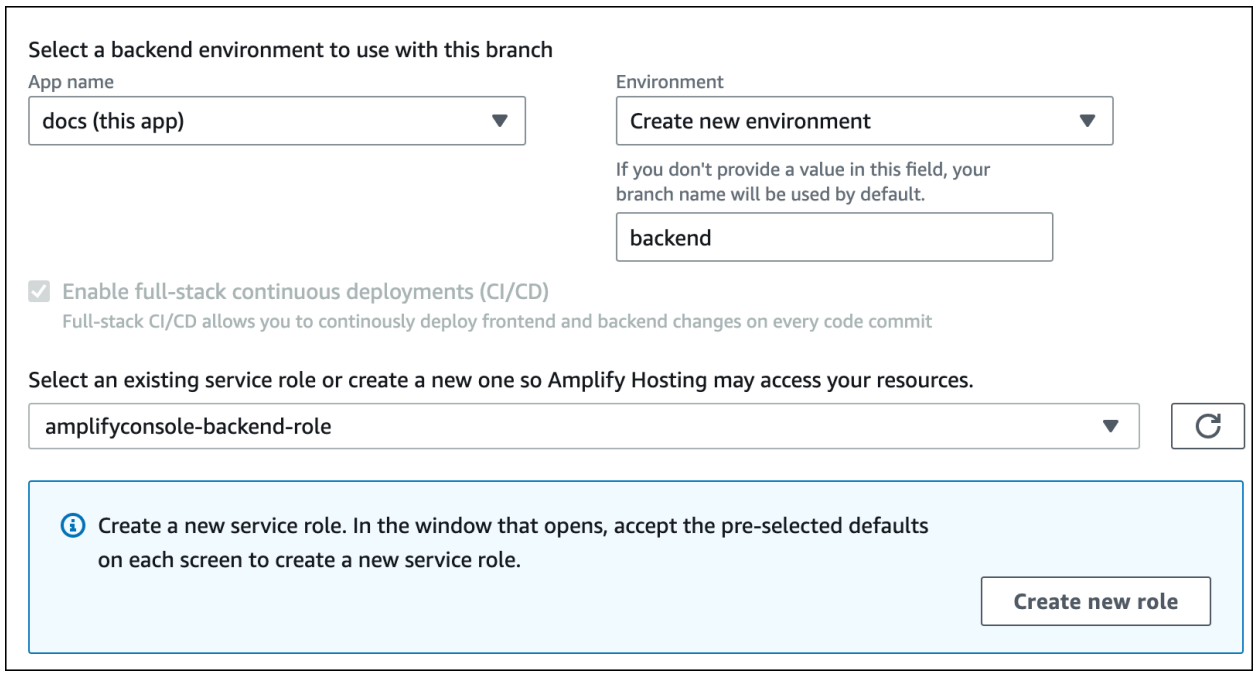
## ソーシャルサインインの認証パラメータを使用して新しいバックエンド環境を作成します。

ブランチをアプリに接続するには

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
2. ブランチをアプリケーションに接続する手順は、ブランチを新しいアプリに接続するのか、既存のアプリに接続するのかによって異なります。

   - ブランチを新しいアプリに接続する

     a. ビルド設定ページで、「このブランチで使用するバックエンド環境を選択する」セクションを探します。[環境] で [新しい環境を作成] を選択し、バックエンド環境の名前を入力します。次のスクリーンショットは、ビルド設定ページの「このブランチで使用するバックエンド環境を選択」セクションで、**backend**バックエンド環境名を入力したところを示しています。




b. ビルド設定ページの「詳細設定」セクションを展開し、ソーシャルログインキー用の環境変数を追加します。例えば、**AMPLIFY_FACEBOOK_CLIENT_SECRET**は有効な環境変数です。デフォルトで使用できる Amplify システム環境変数のリストについては、Amplify の環境変数のリファレンスの表を参照してください。

- ブランチを既存のアプリに接続する

a. 新しいブランチを既存のアプリに接続する場合は、ブランチを接続する前にソーシャルサインインの環境変数を設定します。ナビゲーションペインで、[アプリ設定]、[環境変数] を選択します。

b. [環境変数] セクションで、[編集] を選択します。

c. 「変数の管理」セクションで、「変数を追加」を選択します。

d. [変数 (キー)] には、クライアント ID を入力します。[値] にはクライアントシークレットを入力します。

e. [保存] を選択します。

# 環境シークレットの管理

Amplify Gen 2 のリリースにより、環境シークレットのワークフローが効率化され、Amplify コンソールのシークレットと環境変数の管理が一元化されます。Amplify Gen 2 アプリのシークレットを設定してアクセスする手順については、「Amplify ドキュメント」の「シークレットと環境変数」を参照してください。

Gen 1 アプリの環境シークレットは環境変数と似ていますが、暗号化可能な AWS Systems Manager パラメータストアのキーと値のペアです。Amplify のAppleのプライベートキーでサインインするなど、一部の値は暗号化する必要があります。

## AWS Systems Manager を使用して Amplify Gen 1 アプリケーションの環境シークレットを設定する

AWS Systems Manager コンソールを使用して Gen 1 Amplify アプリの環境シークレットを設定するには、次の手順に従います。

環境シークレットを設定するには

1. にサインイン AWS マネジメントコンソール し、 AWS Systems Manager コンソールを開きます。
2. ナビゲーションペインで [アプリケーション管理] を選択し、[パラメータストア] を選択します。
3. AWS Systems Manager Parameter Storeページで、[パラメータの作成] を選択します。
4. [パラメータの作成] ページの [パラメータの詳細] セクションで、以下を実行します。
   a. [名前] には、**`/amplify/{your_app_id}/{your_backend_environment_name}/{your_parameter_name}`**形式でパラメータを入力します。
   b. [タイプ] には、[安全な文字列] を選択します。
   c. KMS キーソースには、[現在のアカウント] を選択して、アカウントのデフォルトキーを使用します。
   d. [値] には、暗号化するシークレット値を入力します。
5. [パラメータの作成]を選択します。

> **Note**
>
> Amplify は、特定の環境ビルドの`/amplify/{your_app_id}/{your_backend_environment_name}`にあるキーにのみアクセスできます。Amplify AWS KMS key が値を復号できるようにするには、デフォルトを指定する必要があります。

# Gen 1 アプリケーションの環境シークレットへのアクセス

Gen 1 アプリケーションの環境シークレットは、JSON 文字列として `process.env.secrets` に保存されます。

## Amplify 環境のシークレットのリファレンス

Systems Manager パラメータをフォーマット`/amplify/{your_app_id}/{your_backend_environment_name}/AMPLIFY_SIWA_CLIENT_ID`で指定します。

Amplify コンソール内では、デフォルトでアクセス可能な以下の環境シークレットを使用することができます。

| 変数名 | 説明 | 値の例 |
| --- | --- | --- |
| AMPLIFY_SIWA_CLIENT_ID | 「Apple クライアント ID でサインイン」 | `com.yourapp.auth` |
| AMPLIFY_SIWA_TEAM_ID | 「Apple チーム ID でサインイン」 | `ABCD123` |
| AMPLIFY_SIWA_KEY_ID | 「Apple キー ID でサインイン」 | `ABCD123` |
| AMPLIFY_SIWA_PRIVATE_KEY | 「Apple プライベートキーでサインイン」 | -----プライベートキーを開始-----<br>****.....<br>-----プライベートキーを終了----- |

# Amplify アプリのカスタムヘッダーの設定

カスタム HTTP ヘッダーを使用すると、HTTP レスポンスごとにヘッダーを指定することができます。レスポンスヘッダーは、デバッグ、セキュリティ、および情報提供に使用できます。ヘッダーは Amplify コンソールで指定するか、またはアプリの `customHttp.yml` ファイルをダウンロードして編集し、プロジェクトのルートディレクトリに保存することで指定できます。詳細な手順については、カスタムヘッダーの設定 を参照してください。

以前は、Amplify コンソールのビルド仕様 (buildspec) を編集するか、`amplify.yml` ファイルをダウンロードして更新し、プロジェクトのルートディレクトリに保存することで、アプリにカスタム HTTP ヘッダーを指定していました。この方法で指定されたカスタムヘッダーは、buildspec と `amplify.yml` ファイルから移行することを強くお勧めします。手順については、「カスタムヘッダーのビルド仕様と amplify.yml からの移行」を参照してください。

トピック

- カスタムヘッダーの YAML リファレンス
- カスタムヘッダーの設定
- カスタムヘッダーのビルド仕様と amplify.yml からの移行
- モノレポカスタムヘッダーの要件

## カスタムヘッダーの YAML リファレンス

次の YAML 形式を使用してカスタムヘッダーを指定します。

```
customHeaders:
  - pattern: '*.json'
    headers:
    - key: 'custom-header-name-1'
      value: 'custom-header-value-1'
    - key: 'custom-header-name-2'
      value: 'custom-header-value-2'
  - pattern: '/path/*'
    headers:
    - key: 'custom-header-name-1'
      value: 'custom-header-value-2'
```

モノレポには、次の YAML 形式を使用します。

```
applications:
  - appRoot: app1
    customHeaders:
    - pattern: '**/*'
      headers:
      - key: 'custom-header-name-1'
        value: 'custom-header-value-1'
  - appRoot: app2
    customHeaders:
    - pattern: '/path/*.json'
      headers:
      - key: 'custom-header-name-2'
        value: 'custom-header-value-2'
```

アプリにカスタムヘッダーを追加するときは、以下に独自の値を指定します。

pattern

カスタムヘッダーが、パターンと一致するすべての URL ファイルパスに適用されます。

ヘッダー

ファイルパターンと一致するヘッダーを定義します。

key

カスタムヘッダーの名前。

値

カスタムヘッダーの値。

HTTP ヘッダーの詳細については、Mozilla の HTTP ヘッダーのリストを参照してください。

## カスタムヘッダーの設定

Amplify アプリのカスタム HTTP ヘッダーを指定するには、2 つの方法があります。ヘッダーは Amplify コンソールで指定することも、アプリの `customHttp.yml` ファイルをダウンロードして編集し、プロジェクトのルートディレクトリに保存することで指定することもできます。

アプリのカスタムヘッダーを設定し、コンソールに保存するには

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
2. カスタムヘッダーを設定するアプリを選びます。
3. ナビゲーションペインで、[ホスティング] の次に [カスタムヘッダー] を選択します。
4. [カスタムヘッダー] ページで、[編集] を選択します。
5. [カスタムヘッダーの編集] ウィンドウで、[カスタムヘッダーの YAML 形式] を使用してカスタムヘッダーの情報を入力します。

   a. `pattern` には、一致するパターンを入力します。

   b. `key` に、カスタムヘッダーの名前を入力します。

   c. `value` に、カスタムヘッダーの値を入力します。
6. [保存] を選択します。
7. アプリを再デプロイして新しいカスタムヘッダーを適用します。
   - CI/CD アプリケーションの場合は、デプロイするブランチに移動し、「このバージョンを再デプロイ」を選択します。Git リポジトリから新しいビルドを実行することもできます。
   - 手動デプロイアプリの場合は、Amplify のコンソールでアプリを再度デプロイします。

アプリケーションのカスタムヘッダーを設定し、リポジトリのルートに保存するには

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
2. カスタムヘッダーを設定するアプリを選びます。
3. ナビゲーションペインで、[ホスティング] を選択し、[カスタムヘッダー] を選択します。
4. [カスタムヘッダー] ページで、[YML のダウンロード] を選択します。
5. ダウンロードした `customHttp.yml` ファイルを任意のコードエディターで開き、カスタムヘッダー YAML 形式を使用してカスタムヘッダーの情報を入力します。

   a. `pattern` には、一致するパターンを入力します。

   b. `key` に、カスタムヘッダーの名前を入力します。

   c. `value` に、カスタムヘッダーの値を入力します。
6. 編集した `customHttp.yml` ファイルをプロジェクトのルートディレクトリに保存します。モノレポを使用している場合は、リポジトリのルートに `customHttp.yml` ファイルを保存します。
7. アプリを再デプロイして新しいカスタムヘッダーを適用します。

- CI/CD アプリの場合、新しい `customHttp.yml` ファイルを含む Git リポジトリから新しいビルドを実行します。
- 手動デプロイアプリの場合、Amplify コンソールでアプリを再度デプロイし、アップロードしたアーティファクトに新しい `customHttp.yml` ファイルを含めます。

> **Note**
>
> `customHttp.yml` ファイルに設定され、アプリのルートディレクトリにデプロイされたカスタムヘッダーは、Amplify コンソールの [カスタムヘッダー] セクションで定義されているカスタムヘッダーよりも優先されます。

## セキュリティカスタムヘッダーの例

カスタムセキュリティヘッダーによって、HTTPS を適用し、XSS 攻撃を回避して、ブラウザをクリックジャックから守ることができます。次の YAML 構文を使用して、カスタムセキュリティヘッダーをアプリに適用します。

```
customHeaders:
  - pattern: '**'
    headers:
      - key: 'Strict-Transport-Security'
        value: 'max-age=31536000; includeSubDomains'
      - key: 'X-Frame-Options'
        value: 'SAMEORIGIN'
      - key: 'X-XSS-Protection'
        value: '1; mode=block'
      - key: 'X-Content-Type-Options'
        value: 'nosniff'
      - key: 'Content-Security-Policy'
        value: "default-src 'self'"
```

## キャッシュコントロールカスタムヘッダーの設定

Amplify でホストされるアプリは、オリジンから送信される `Cache-Control` ヘッダーを尊重します。ただし、定義したカスタムヘッダーで上書きする場合は除きます。Amplify は、`200 OK` ステータスコードで成功したレスポンスにのみキャッシュコントロールカスタムヘッダーを適用します。こ

れにより、エラー応答がキャッシュされ、同じリクエストを行う他のユーザーに送信されるのを防ぎます。

s-maxage ディレクティブを手動で調整して、アプリのパフォーマンスとデプロイの可用性をより細かく制御できます。たとえば、コンテンツがエッジにキャッシュされる時間を長くするには、デフォルトの 600 秒 (10 分) よりも長い値に s-maxage を更新して Time To Live (TTL) を手動で延長できます。

s-maxage のカスタム値を指定するには、次の YAML 形式を使用します。この例では 3600 秒 (1 時間) の間、関連するコンテンツをエッジにキャッシュします。

```
customHeaders:
  - pattern: '/img/*'
    headers:
      - key: 'Cache-Control'
        value: 's-maxage=3600'
```

ヘッダーによるアプリケーションパフォーマンス制御の詳細については、「アプリのパフォーマンスを向上させるような Cache-Control ヘッダーの使用」を参照してください。

## カスタムヘッダーのビルド仕様と amplify.yml からの移行

以前は、Amplify コンソールのビルド仕様を編集するか、amplify.yml ファイルをダウンロードして更新し、プロジェクトのルートディレクトリに保存することで、アプリにカスタム HTTP ヘッダーを指定していました。カスタムヘッダーをビルド使用と amplify.yml ファイルから移行することを強くお勧めします。

Amplify コンソールの [カスタムヘッダー] セクションでカスタムヘッダーを指定するか、または customHttp.yml ファイルをダウンロードして編集して指定します。

Amplify のコンソールに保存されているカスタムヘッダーを移行するには

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
2. カスタムヘッダーの移行を実行するアプリを選択します。
3. ナビゲーションペインで [ホスティング]、[ビルド設定] を選択します。「アプリビルド仕様」セクションでは、アプリのビルドスペックを確認できます。
4. [ダウンロード] を選択して、現在のビルドスペックのコピーを保存します。設定を復元する必要がある場合、後でこのコピーを参照できます。

5. ダウンロードが完了したら、[編集]を選択します。
6. ファイル内のカスタムヘッダー情報は、後ほどステップ 9 で使用するので、メモしておいてください。「編集」ウィンドウで、ファイルからカスタムヘッダーをすべて削除し、[保存] を選択します。
7. ナビゲーションペインで、[ホスティング]、[カスタムヘッダー] を選択します。
8. [カスタムヘッダー] ページで、[編集] を選択します。
9. [カスタムヘッダーの編集] ウィンドウに、ステップ 6 で削除したカスタムヘッダーの情報を入力します。
10. [保存] を選択します。
11. 新しいカスタムヘッダーを適用したいブランチをすべて再デプロイします。

カスタムヘッダーを amplify.yml から customHTTP.yml に移行するには

1. アプリのルートディレクトリに現在デプロイされている `amplify.yml` ファイルに移動します。
2. 適切なエディタで、`amplify.yml`ファイルを開きます。
3. ファイル内のカスタムヘッダー情報は、後ほどステップ 8 で使用するので、メモしておいてください。ファイル内のカスタムヘッダーを削除します。ファイルを保存して閉じます。
4. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
5. カスタムヘッダーを設定するアプリを選びます。
6. ナビゲーションペインで、[ホスティング]、[カスタムヘッダー] を選択します。
7. [カスタムヘッダー] ページで、[ダウンロード] を選択します。
8. ダウンロードした `customHttp.yml` ファイルを任意のコードエディターで開き、ステップ 3 の `amplify.yml` から削除したカスタムヘッダーの情報を入力します。
9. 編集した `customHttp.yml` ファイルをプロジェクトのルートディレクトリに保存します。モノレポを使用している場合は、リポジトリのルートにファイルを保存します。
10. アプリを再デプロイして新しいカスタムヘッダーを適用します。
    - CI/CD アプリの場合、新しい `customHttp.yml` ファイルを含む Git リポジトリから新しいビルドを実行します。
    - 手動デプロイアプリの場合、Amplify のコンソールにアプリを再度デプロイし、アップロードしたアーティファクトを含む新しい `customHttp.yml` ファイルを含めます。

> **Note**
>
> `customHttp.yml` ファイルで設定され、アプリのルートディレクトリにデプロイされたカスタムヘッダーは、Amplify コンソールの [カスタムヘッダー] セクションで定義されたカスタムヘッダーよりも優先されます。

# モノレポカスタムヘッダーの要件

モノレポでアプリにカスタムヘッダーを指定する場合、以下の設定要件に注意してください。

- モノレポには特定の YAML 形式があります。正しい構文については、カスタムヘッダーの YAML リファレンス を参照してください。
- Amplify のコンソールの [カスタムヘッダー] セクションを使用して、モノレポ内のアプリケーションのカスタムヘッダーを指定できます。新しいカスタムヘッダーを適用するには、アプリケーションを再デプロイする必要があります。
- コンソールを使用する代わりに、`customHttp.yml` ファイルのモノレポでアプリのカスタムヘッダーを指定することもできます。新しいカスタムヘッダーを適用するには、`customHttp.yml` ファイルをリポジトリのルートに保存し、アプリケーションを再デプロイする必要があります。`customHttp.yml` ファイル内で指定されたカスタムヘッダーは、Amplify コンソールの [カスタムヘッダー] セクションを使用して指定されたカスタムヘッダーよりも優先されます。

# アプリケーションのキャッシュ設定の管理

Amplify は Amazon CloudFront を使用して、ホストされたアプリケーションのキャッシュ設定を管理します。キャッシュ構成がそれぞれのアプリに適用され、最高のパフォーマンスを実現するために最適化されます。

2024 年 8 月 13 日、Amplify はアプリケーションのキャッシュ効率の改善を発表しました。詳細については、「CDN キャッシュの改善による AWS Amplify ホスティングによるアプリケーションパフォーマンスの向上」を参照してください。

次の表は、キャッシュ改善をリリースする前後に動作する特定のキャッシュに対する Amplify サポートをまとめたものです。

| キャッシュの動作 | 以前のサポート | キャッシュの改善 |
| --- | --- | --- |
| アプリのカスタムヘッダーは、Amplify コンソールまたは `customHeaders.yaml` ファイルに追加できます。上書きできるヘッダーの 1 つは `Cache-Control` です。詳細については、「Amplify アプリのカスタムヘッダーの設定」を参照してください。 | はい | はい |
| Amplify は、`customHeaders.yaml` ファイルで定義した `Cache-Control` ヘッダーを尊重し、Amplify のデフォルトのキャッシュ設定よりも優先します。 | はい | はい |
| Amplify は、動的ルート (Next.js SSR ルートなど) のアプリケーションのフレームワーク内で設定された `Cache-Control` ヘッダー | はい | はい |

| キャッシュの動作 | 以前のサポート | キャッシュの改善 |
| --- | --- | --- |
| を尊重します。Cache-Control ヘッダーがアプリの customHeaders.yaml ファイルに設定されている場合、next.config.js ファイルの設定よりも優先されます。 | | |
| CI/CD アプリが新しくデプロイされるたびにキャッシュがクリアされます。 | はい | はい |
| アプリのパフォーマンスモードをオンにできます。 | はい | なし<br><br>パフォーマンスモード設定は、Amplify コンソールでは使用できなくなりました。ただし、s-maxage ディレクティブを設定する Cache-Control ヘッダーを作成できます。手順については、「アプリのパフォーマンスを向上させるような Cache-Control ヘッダーの使用」を参照してください。 |

次の表に、特定のキャッシュ設定のデフォルト値の変更を示します。

| キャッシュ設定 | 以前のデフォルト値 | キャッシュの改善によるデフォルト値 |
| --- | --- | --- |
| 静的アセットのキャッシュ期間 | 2 秒 | 1 年 |
| リバースプロキシレスポンスのキャッシュ期間 | 2 秒 | ゼロ秒 (キャッシュなし) |

| キャッシュ設定 | 以前のデフォルト値 | キャッシュの改善によるデフォルト値 |
| --- | --- | --- |
| 最大有効期限 (TTL) | 10 分 | 1 年 |

Amplify がアプリケーションに適用するキャッシュ設定を決定する方法と、キャッシュキー設定の管理手順の詳細については、次のトピックを参照してください。

トピック

- Amplify がアプリにキャッシュ設定を適用する方法
- キャッシュキー Cookie の管理
- アプリのパフォーマンスを向上させるような Cache-Control ヘッダーの使用

# Amplify がアプリにキャッシュ設定を適用する方法

アプリケーションのキャッシュを管理するために、Amplify はアプリのプラットフォームタイプと書き換えルールを調べて、提供コンテンツのタイプを決定します。Compute アプリの場合、Amplify はデプロイマニフェストのルーティングルールも調べます。

> **Note**
>
> アプリケーションのプラットフォームタイプは、デプロイ中に Amplify ホスティングによって設定されます。SSG (静的) アプリは、プラットフォームタイプ WEB に設定されます。SSR アプリ (Next.js 12 以降) は、プラットフォーム WEB_COMPUTE に設定されます。

Amplify は、次の 4 種類のコンテンツを識別し、指定されたマネージドキャッシュポリシーを適用します。

静的

WEB プラットフォームを持つアプリから提供されるコンテンツ、または WEB_COMPUTE アプリ内の静的ルート。

このコンテンツは Amplify-StaticContent キャッシュポリシーを使用します。

画像の最適化

`WEB_COMPUTE` アプリ内の `ImageOptimization` ルートによって提供されるイメージ。

このコンテンツは Amplify-ImageOptimization キャッシュポリシーを使用します。

コンピューティング

`WEB_COMPUTE` アプリ内の `Compute` ルートによって提供されるコンテンツ。これには、すべてのサーバーサイドレンダリング (SSR) コンテンツがあります。

このコンテンツでは、Amplify App に設定されている `cacheConfig.type` の値に応じて、Amplify-Default または Amplify-DefaultNoCookies のキャッシュポリシーを使用します。

リバースプロキシ

リバースプロキシの書き換えカスタムルールに一致するパスによって提供されるコンテンツ。このカスタムルール作成の詳細については、「リダイレクトの使用」の章の「リバースプロキシの書き換え」を参照してください。

このコンテンツでは、Amplify App に設定されている `cacheConfig.type` の値に応じて、Amplify-Default または Amplify-DefaultNoCookies のキャッシュポリシーを使用します。

## Amplify のマネージドキャッシュポリシーについて

Amplify は、以下のマネージドキャッシュポリシーを使用して、ホストされたアプリケーションのデフォルトのキャッシュ設定を最適化します。

- Amplify-Default
- Amplify-DefaultNoCookies
- Amplify-ImageOptimization
- Amplify-StaticContent

### Amplify-Default マネージドキャッシュポリシー設定

このポリシーを CloudFront コンソールで見る

このポリシーは、AWS Amplify ウェブアプリケーションであるオリジンで使用するように設計されています。

このポリシーの設定は以下のとおりです。

- 最小 TTL: 0 秒
- 最大 TTL: 31536000 秒 (1 年)
- デフォルト TTL: 0 秒
- キャッシュキーに含まれるヘッダー:
  - `Authorization`
  - `Accept`
  - `CloudFront-Viewer-Country`
  - `Host`
- キャッシュキーに含まれる Cookie: すべての Cookie が含まれます。
- キャッシュキーに含まれるクエリ文字列: すべてのクエリ文字列が含まれます。
- 圧縮オブジェクトのキャッシュ設定: Gzip と Brotli が対応しています。

## Amplify-DefaultNoCookies の管理キャッシュポリシー設定

このポリシーを CloudFront コンソールで見る

このポリシーは、AWS Amplify ウェブアプリケーションであるオリジンで使用するように設計されています。

このポリシーの設定は以下のとおりです。

- 最小 TTL: 0 秒
- 最大 TTL: 31536000 秒 (1 年)
- デフォルト TTL: 0 秒
- キャッシュキーに含まれるヘッダー:
  - `Authorization`
  - `Accept`
  - `CloudFront-Viewer-Country`
  - `Host`
- キャッシュキーに含まれる Cookie: Cookies は使用されていません。
- キャッシュキーに含まれるクエリ文字列: すべてのクエリ文字列が含まれます。
- 圧縮オブジェクトのキャッシュ設定: Gzip と Brotli が対応しています。

## Amplify-ImageOptimization 管理キャッシュポリシー設定

このポリシーを CloudFront コンソールで見る

このポリシーは、AWS Amplify ウェブアプリケーションであるオリジンで使用するように設計されています。

このポリシーの設定は以下のとおりです。

- 最小 TTL: 0 秒
- 最大 TTL: 31536000 秒 (1 年)
- デフォルト TTL: 0 秒
- キャッシュキーに含まれるヘッダー:
  - `Authorization`
  - `Accept`
  - `Host`
- キャッシュキーに含まれる Cookie: Cookies は使用されていません。
- キャッシュキーに含まれるクエリ文字列: すべてのクエリ文字列が含まれます。
- 圧縮オブジェクトのキャッシュ設定: Gzip と Brotli が対応しています。

## Amplify-StaticContent マネージドキャッシュポリシー設定

このポリシーを CloudFront コンソールで見る

このポリシーは、AWS Amplify ウェブアプリケーションであるオリジンで使用するように設計されています。

このポリシーの設定は以下のとおりです。

- 最小 TTL: 0 秒
- 最大 TTL: 31536000 秒 (1 年)
- デフォルト TTL: 0 秒
- キャッシュキーに含まれるヘッダー:
  - `Authorization`
  - `Host`
- キャッシュキーに含まれる Cookie: Cookies は使用されていません。

- キャッシュキーに含まれるクエリ文字列: クエリ文字列はありません。
- 圧縮オブジェクトのキャッシュ設定: Gzip と Brotli が対応しています。

# キャッシュキー Cookie の管理

Amplify にアプリをデプロイするときに、キャッシュキーに Cookie を含めるか除外するかを選択できます。Amplify コンソールでは、この設定は、[キャッシュキー設定] トグルを使用して [カスタムヘッダーとキャッシュ] ページで指定されます。手順については、「キャッシュキーからの Cookie を含めるまたは除外」を参照してください。

キャッシュキーに Cookie を含める

この設定では、Amplify は、提供されるコンテンツのタイプに基づいて、アプリに最適なキャッシュ設定を自動的に選択します。このキャッシュ構成タイプを明示的に選択する必要があります。

SDKs または を使用している場合 AWS CLI、この設定は `CreateApp`または UpdateApp APIs `AMPLIFY_MANAGED`で `cacheConfig.type`を に設定することに対応します。

キャッシュキーから Cookie を除外する

これはデフォルトのキャッシュ設定です。このキャッシュ構成は、キャッシュキーからすべての Cookie を除外する点を除いて、`AMPLIFY_MANAGED` 設定に似ています。

キャッシュキーから Cookie を除外することを選択すると、キャッシュパフォーマンスが向上します。ただし、このキャッシュ構成を選択する前に、アプリケーションが Cookie を使用して動的コンテンツを提供するかどうかを検討することが重要です。

SDKs または を使用している場合 AWS CLI、この設定は `CreateApp`または UpdateApp APIs `AMPLIFY_MANAGED_NO_COOKIES`を使用して `cacheConfig.type`を に設定することに対応します。

キャッシュキーの詳細については、「Amazon CloudFront デベロッパーガイド」の「キャッシュキーについて」を参照してください。

## キャッシュキーからの Cookie を含めるまたは除外

アプリのキャッシュキー Cookie 設定は、Amplify コンソール、 SDK または AWS CLIで設定できます。

次の手順に従って、Amplify コンソールを使用して新しいアプリをデプロイするときに、キャッシュキーに Cookie を含めるか除外するかを指定します。

Amplify にアプリをデプロイするときにキャッシュキー Cookie 設定を設定するには

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
2. [すべてのアプリ] ページで、[アプリの新規作成] を選択します。
3. [Amplify で構築を開始] ページで、自分の Git リポジトリプロバイダーを選択し、[次へ] を選択します。
4. [リポジトリブランチを追加] ページで、次の操作を行います。

   a. 接続するリポジトリの名前を選択します。

   b. 接続するリポジトリブランチの名前を選択します。

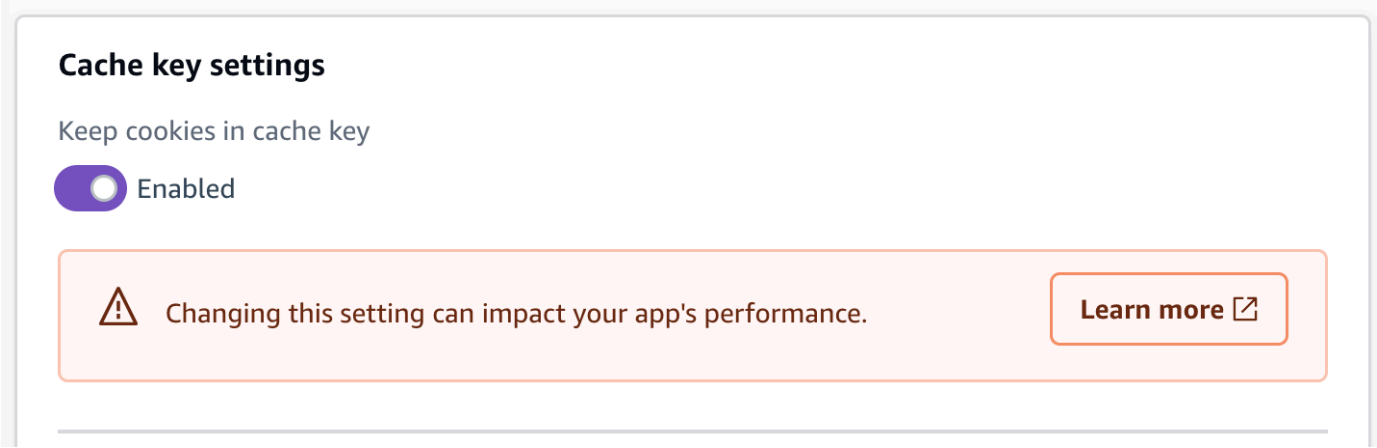
   c. [次へ] を選択します。
5. アプリに IAM サービスロールが必要な場合、Amplify ホスティングコンピューティングにサービスロールを自動的に作成させることも、作成したロールを指定することもできます。

   - Amplify が自動的にロールを作成してアプリにアタッチできるようにするには:

     - [新しいサービスロールの作成と使用] を選択します。
   - 以前に作成したサービスロールをアタッチするには:

     a. [既存のサービスロールを使用する] を選択します。

     b. リストから使用するロールを選択します。
6. [詳細設定] を選択し、[キャッシュキー設定] セクションを見つけます。
7. [キャッシュキー に Cookie を保持する] または [キャッシュキー から Cookie を削除する] を選択します。次のスクリーンショットは、コンソールの [キャッシュキー設定] トグルを示しています。



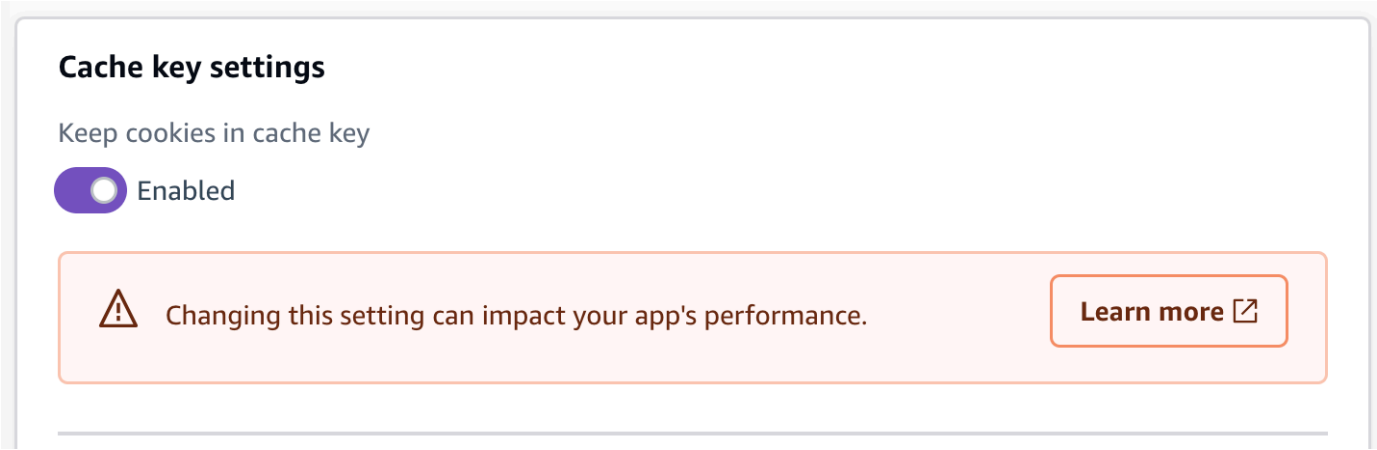

8. [次へ] を選択します。
9. [レビュー]ページで、[保存してデプロイ] を選択します。

## アプリケのキャッシュキー Cookie 設定の変更

Amplify にデプロイ済みのアプリのキャッシュキー Cookie 設定を変更できます。次の手順に従って、Amplify コンソールを使用してアプリケーションのキャッシュキーに Cookie を含めるか除外するかを変更します。

デプロイされたアプリのキャッシュキー Cookie 設定を変更するには

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
2. [すべてのアプリ] ページで、更新するアプリケーションを選択します。
3. ナビゲーションペインで、[ホスティング] を選択し、[カスタムヘッダーとキャッシュ] を選択します。
4. [カスタムヘッダーとキャッシュ] ページで、[キャッシュキー設定] セクションを見つけ、[編集] を選択します。
5. [キャッシュキー に Cookie を保持する] または [キャッシュキー から Cookie を削除する] を選択します。次のスクリーンショットは、コンソールの [キャッシュキー設定] トグルを示しています。




6. [保存] を選択します。

# アプリのパフォーマンスを向上させるような Cache-Control ヘッダーの使用

Amplify のデフォルトのホスティングアーキテクチャは、ホスティングパフォーマンスとデプロイの可用性のバランスを最適化します。ほとんどのお客様には、デフォルトのアーキテクチャを使用することをお勧めします。

アプリのパフォーマンスをより細かく制御する必要がある場合は、コンテンツ配信ネットワーク (CDN) エッジでキャッシュされたコンテンツをより長い間隔で保持することにより、ホスティングパフォーマンスを最適化するように HTTP `Cache-Control` ヘッダーを手動で設定できます。

HTTP `Cache-Control` ヘッダー の `max-age` と `s-maxage` のディレクティブは、アプリのコンテンツキャッシュ期間に影響します。`max-age` ディレクティブは、オリジンサーバーからコンテンツが更新されるまでにコンテンツをキャッシュに保持する期間 (秒単位) をブラウザに指示します。`s-maxage` ディレクティブでは `max-age` よりも優先され、オリジンサーバーからコンテンツが更新されるまでにコンテンツが CDN エッジに保持される期間 (秒単位) を指定できます。

Amplify でホストされるアプリは、オリジンから送信される `Cache-Control` ヘッダーを尊重します。ただし、定義したカスタムヘッダーで上書きする場合は除きます。Amplify は、`200 OK` ステータスコードで成功したレスポンスに対してのみ、`Cache-Control` カスタムヘッダーを適用します。これにより、エラー応答がキャッシュされ、同じリクエストを行う他のユーザーに送信されるのを防ぎます。

`s-maxage` ディレクティブを手動で調整して、アプリのパフォーマンスとデプロイの可用性をより細かく制御できます。たとえば、コンテンツがエッジにキャッシュされている時間を変更するには、

デフォルトの 31536000 秒 (1 年) 以外の値に `s-maxage` を更新することで、有効期間 (TTL) を手動で延長できます。

Amplify コンソールの「カスタムヘッダー」セクションで、アプリのカスタムヘッダーを定義できます。YAML 形式の例については、「キャッシュコントロールカスタムヘッダーの設定」を参照してください。

次の手順で、CDN エッジでコンテンツを 24 時間キャッシュされたままにするように `s-maxage` ディレクティブを設定します。

カスタム Cache-Control ヘッダーを設定するには

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
2. カスタムヘッダーを設定するアプリを選びます。
3. ナビゲーションペインで、[ホスティング]、[カスタムヘッダー] を選択します。
4. [カスタムヘッダー] ページで、[編集] を選択します。
5. [カスタムヘッダーの編集] ウィンドウで、カスタムヘッダーの情報を次のように入力します:
   a. `pattern` の場合、すべてのパスに **`**/*`** を入力します。
   b. `key` に「**`Cache-Control`**」と入力します。
   c. `value` に「**`s-maxage=86400`**」と入力します。
6. [保存] を選択します。
7. アプリを再デプロイして新しいカスタムヘッダーを適用します。

# Amplify デプロイのスキュー保護

デプロイスキュー保護は、ウェブアプリケーションのクライアントとサーバー間のバージョンスキューの問題を排除するために、Amplify アプリケーションで使用できます。Amplify アプリケーションにスキュー保護を適用すると、デプロイの発生時期に関係なく、クライアントが常に正しいバージョンのサーバー側アセットとやり取りできるようになります。

バージョンスキューは、ウェブデベロッパーにとって一般的な課題です。これは、ウェブブラウザが古いバージョンのアプリケーションを実行し、サーバーが新しいバージョンを実行している場合に発生します。この不一致により、アプリケーションのユーザーに予期しない動作、エラー、エクスペリエンスの低下が発生する可能性があります。Amplify デプロイスキュー保護機能は、ウェブブラウザで実行されているクライアントを特定のデプロイにピン留めします。これにより、Amplify は常にその特定のデプロイのアセットを提供し、クライアントとサーバーの同期を維持します。

Amplify のスキュー保護機能により、新しいデプロイをリリースする際のアプリケーションのユーザーのエラーを減らすことができます。また、後方互換性と前方互換性の問題の管理に費やす時間を短縮することで、デベロッパーのエクスペリエンスを向上させることもできます。

スキュー保護機能の詳細:

サポートされるアプリケーションタイプ

Amplify がサポートする任意のフレームワークで作成された静的アプリケーションと SSR アプリケーションにスキュー保護を追加できます。アプリケーションは、Git リポジトリまたは手動デプロイからデプロイできます。

`WEB_DYNAMIC` プラットフォームにデプロイされているアプリケーション (Next.js バージョン 11 以前) にスキュー保護を追加することはできません。

時間

静的アプリケーションの場合、Amplify は 1 週間のデプロイを提供します。SSR アプリケーションの場合、最大 8 つ前までのデプロイに対してスキュー保護が保証されます。

Cost

アプリケーションにスキュー保護を追加するために、追加コストはかかりません。

パフォーマンスに関する考慮事項

アプリケーションでスキュー保護が有効になっている場合、Amplify は CDN キャッシュ設定を更新する必要があります。したがって、スキュー保護を有効にした後の最初のデプロイには最大 10 分かかることが予想されます。

トピック

- Amplify アプリケーションのデプロイスキュー保護の設定
- スキュー保護の仕組み

# Amplify アプリケーションのデプロイスキュー保護の設定

Amplify コンソール、、または SDKs を使用して、アプリケーションのデプロイスキュー保護を追加 AWS Command Line Interfaceまたは削除できます。この機能はブランチレベルに適用されます。ブランチに対してスキュー保護が有効になった後に行われる新しいデプロイのみがスキュー保護されます。

AWS CLI または SDKs を使用してデプロイスキュー保護を追加または削除するには、 `CreateBranch.enableSkewProtection` および `UpdateBranch.enableSkewProtection`フィールドを使用します。詳細については、「API リファレンスドキュメント」の「CreateBranch」と「UpdateBranch」を参照してください。

特定のデプロイを削除してサービスを停止する場合は、`DeleteJob` API を使用します。詳細については、「Amplify API リファレンスドキュメント」の「DeleteJob」を参照してください。

現時点では、Amplify Hosting にデプロイ済みのアプリケーションでのみスキュー保護を有効にできます。Amplify コンソールを使用してブランチにスキュー保護を追加するには、次の手順に従います。

Amplify アプリケーションのブランチのスキュー保護を有効にする

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/amplify/ で Amplify コンソールを開きます。
2. [すべてのアプリ] ページで、スキュー保護を有効にするデプロイ済みアプリの名前を選択します。
3. ナビゲーションペインで [アプリの設定] を選択してから、[ブランチ設定] を選択します。

4. [ブランチ] セクションで、更新するブランチの名前を選択します。
5. [アクション] メニューで、[スキュー保護を有効にする] を選択します。
6. 確認ウィンドウで、[確認] を選択します。ブランチでスキュー保護が有効になりました。
7. アプリケーションブランチを再デプロイします。スキュー保護が有効になった後に行われたデプロイのみがスキュー保護されます。

Amplify コンソールを使用してアプリケーションのブランチからスキュー保護を除去するには、次の手順に従います。

Amplify アプリケーションのブランチからスキュー保護を除去する

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/amplify/ で Amplify コンソールを開きます。
2. [すべてのアプリ] ページで、スキュー保護を除去するデプロイ済みアプリの名前を選択します。
3. ナビゲーションペインで [アプリの設定] を選択してから、[ブランチ設定] を選択します。
4. [ブランチ] セクションで、更新するブランチの名前を選択します。
5. [アクション] メニューで、[スキュー保護を無効にする] を選択します。ブランチのスキュー保護が無効になり、最新のコンテンツのみが提供されるようになりました。

# スキュー保護の仕組み

ほとんどの場合、_dpl Cookie のデフォルトの動作は、スキュー保護のニーズに対応します。ただし、次のような高度なシナリオでは、`X-Amplify-Dpl` ヘッダーと `dpl` クエリパラメータを使用してスキュー保護を有効にするほうが適しています。

- 複数のブラウザタブにウェブサイトを同時にロードする
- サービスワーカーを使用する

Amplify は、クライアントに提供するコンテンツを決定するときに、受信リクエストを次の順序で評価します。

1. **`X-Amplify-Dpl`** ヘッダー – アプリケーションはこのヘッダーを使用して、特定の Amplify デプロイにリクエストを送信できます。このリクエストヘッダーは、`process.env.AWS_AMPLIFY_DEPLOYMENT_ID` の値を使用して設定できます。

2. **dpl** クエリパラメータ – Next.js アプリケーションは、フィンガープリントされたアセット (.js ファイルと .css ファイル) へのリクエストに対して、_dpl クエリパラメータを自動的に設定します。
3. _dpl Cookie – これは、スキュー保護されたすべてのアプリケーションのデフォルトです。特定のブラウザの場合、ドメインとやり取りするブラウザタブまたはインスタンスごとに同じ Cookie が送信されます。

   異なるブラウザタブに異なるバージョンのウェブサイトがロードされている場合、_dpl Cookie はすべてのタブで共有されることに注意してください。このシナリオでは、_dpl Cookie で完全なスキュー保護を達成することはできません。スキュー保護に `X-Amplify-Dpl` ヘッダーを使用することを検討する必要があります。

## X-Amplify-Dpl ヘッダーの例

次の例は、`X-Amplify-Dpl` ヘッダーを介してスキュー保護にアクセスする Next.js SSR ページのコードを示しています。このページは、その API ルートの 1 つに基づいてコンテンツをレンダリングします。API ルートに提供するデプロイは、`process.env.AWS_AMPLIFY_DEPLOYMENT_ID` の値に設定されている `X-Amplify-Dpl` ヘッダーを使用して指定します。

```
import { useEffect, useState } from 'react';

export default function MyPage({deploymentId}) {
    const [data, setData] = useState(null);

    useEffect(() => {
        fetch('/api/hello', {
            headers: {
                'X-Amplify-Dpl': process.env.AWS_AMPLIFY_DEPLOYMENT_ID
            },
        })
        .then(res => res.json())
        .then(data => setData(data))
        .catch(error => console.error("error", error))
    }, []);

    return <div>
        {data ? JSON.stringify(data) : "Loading ... " }
    </div>
}
```

# Amplify アプリケーションのモニタリング

AWS Amplify には、ホストされたアプリケーションをモニタリングするための以下の機能があります。

- CloudWatch メトリクス – Amplify は Amazon CloudWatch を介してメトリクスを出力します。このメトリクスを使用して、アプリケーションのトラフィック、エラー、データ転送、レイテンシーをモニタリングできます。
- アクセスログ – Amplify は、アプリに対して行われたリクエストに関する詳細情報を含むアクセスログを提供します。
- CloudTrail ログ記録 – Amplify は と統合 AWS CloudTrail されており、Amplify のユーザー、ロール、または AWS のサービスによって実行されたアクションの記録を提供します。CloudTrail コンソールでこれらのイベントを表示できます。

トピック



## Amazon CloudWatch による Amplify アプリケーションのモニタリング

AWS Amplify は Amazon CloudWatch と統合されているため、Amplify アプリケーションのメトリクスをほぼリアルタイムでモニタリングし、メトリクスが設定したしきい値を超えたときに通知を送信するアラームを作成できます。CloudWatchサービスの動作の詳細については、Amazon CloudWatch ユーザーガイドを参照してください。

### 「Supported CloudWatch metrics」(サポートされている CloudWatch メトリクス)

Amplify は、アプリのトラフィック、エラー、データ転送、レイテンシー、およびリクエストトークンを監視するために、`AWS/AmplifyHosting` の名前空間で 7 つの CloudWatch メトリクスをサポートしています。これらのメトリクスは1分間隔で集計されます。CloudWatch モニタリングメトリクスは無料で、CloudWatch Service Quotasにはカウントされません。

次の表には、サポートされている各メトリクスの説明とともに最も関連性の高い統計がまとめられています。利用可能なすべての統計が必ずしもすべてのメトリクスに適用可能であるとは限りません。

| メトリクス | 説明 |
| --- | --- |
| リクエスト | アプリが受信したビューアリクエストの合計数。<br><br>最も関連性の高い統計はSumです。リクエストの合計数を得るには、Sum 統計を使います。 |
| BytesDownloaded | リクエストに対して視聴者がアプリから転送 (ダウンロード) したデータGET、HEAD、OPTIONSの総量 (バイト単位) 。<br><br>最も関連性の高い統計はSumです。 |
| BytesUploaded | ヘッダーなどの、あらゆるリクエストに対してアプリに転送 (アップロード) されたデータの総量 (バイト単位)。<br><br>Amplify では、アプリケーションにアップロードされたデータに対する請求はありません。<br><br>最も関連性の高い統計はSumです。 |
| 4xxErrors | HTTP ステータスコード 400～499 の範囲のエラーを返したリクエストの数。<br><br>最も関連性の高い統計はSumです。これらのエラーの出現総数を取得するために、Sum統計を使用します。 |
| 5xxErrors | HTTPステータスコード500～599の範囲のエラーを返したリクエストの数。 |

| メトリクス | 説明 |
| --- | --- |
| | 最も関連性の高い統計はSumです。これらのエラーの出現総数を取得するために、Sum統計を使用します。 |
| レイテンシー | 最初のバイトまでの時間 (秒単位)。Amplify ホスティングがリクエストを受け取ってから、ネットワークにレスポンスを返すまでの総時間。視聴者のデバイスに到達するレスポンスに発生したネットワークレイテンシーは含まれません。<br><br>最も関連性の高い統計はAverage、Maximum、Minimum、p10、p50、p90p9す。<br><br>予測されるレイテンシーを評価するためにAverage統計を使用します。 |

| メトリクス | 説明 |
| --- | --- |
| TokensConsumed | アプリが消費するリクエストトークン。<br><br>Sum 統計は、リクエストトークンの合計消費量を表します。この統計を現在の Request tokens per second サービスクォータと比較して、将来の高トラフィックイベント中にスロットリングが発生する可能性を避けるためにクォータの引き上げをリクエストする必要があるかどうかを判断できます。<br><br>Average 統計は、通常の時間とピーク時のリクエストトークンの消費量を表します。通常、トークンの消費量が多いほど、最初のバイトまでの時間 (TTFB) が長くなります。したがって、アプリケーションのレイテンシーを評価するときにこの統計を使用できます。レイテンシーが不十分な場合は、ダウンストリーム API を改善してトークンの消費量を減らし、トークンの消費量がアプリケーションの Request tokens per second サービスクォータを超えたときに発生する可能性のあるスロットリングを回避できます。<br><br>Request tokens per second のサービスクォータの詳細については、「Amplify ホスティング Service Quotas」を参照してください。 |

Amplifyには、以下の CloudWatch メトリクスディメンションが用意されています。

| ディメンション | 説明 |
| --- | --- |
| アプリケーション | 指標データはアプリによって提供されます。 |

| ディメンション | 説明 |
| --- | --- |
| AWS アカウント | メトリクスデータは、 のすべてのアプリで提供されます AWS アカウント。 |

## CloudWatch メトリクスへのアクセス

次の手順を使用して、Amplify コンソールから直接 CloudWatch メトリクスにアクセスできます。

> **Note**
>
> https://console.aws.amazon.com/cloudwatch/ AWS マネジメントコンソール の で CloudWatch メトリクスにアクセスすることもできます。

Amplify コンソールを使用してメトリクスにアクセスするには

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
2. メトリクスを表示するアプリを選択します。
3. ナビゲーションペインで、[モニタリング]、[メトリクス] の順に選択します。

## CloudWatch アラームの作成

特定の基準が満たされた際に、通知を送信する CloudWatch アラームを Amplify コンソールで作成できます。アラームは単一の CloudWatch メトリクスを監視し、メトリクスが所定の評価期間の数にわたってしきい値に違反すると、Amazon Simple Notice Service 通知を送信します。

CloudWatch コンソールまたは CloudWatch API を使用して、メトリクスの数学式を用いたより高度なアラームを作成できます。例えば、4xxErrorsの割合が 3 つの連続期間で 15% を超えたときに通知するアラームを作成できます。詳細については、「Amazon CloudWatch ユーザーガイド」の「メトリクスの数式に基づく CloudWatch アラームの作成」を参照してください。

アラームには標準の CloudWatch 料金が適用されます。詳細については、「Amazon CloudWatch の料金」を参照してください。

次の手順に従って、Amplify コンソールを使用することで ドメインを作成します。

Amplify メトリクスの CloudWatch アラームを作成するには

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
2. アラームをセットアップするクォータを選択します。
3. ナビゲーションペインで、[モニタリング]、[アラーム] の順に選択します。
4. [アラーム] タブで、[アラームの作成] を選択します。
5. 「アラームの作成」ウィンドウで、アラームを次のように設定します。

   a. メトリックを監視するには、メトリック名をリストから選択します。

   b. [アラーム名] に、アラームに意味のある名前を入力します。例えば、リクエストを監視している場合、アラームに**`HighTraffic`**という名前を付けることができます。名前には ASCII 文字のみを使用します。

   c. [通知を設定]については、次のいずれかを実行します。

      - i. 次の手順に従って、新規を選択して Amazon SNS の新しいトピックを作成します。
        ii. [Eメールアドレス] には、通知の受信者の Eメールアドレスを入力します。
        iii. 受信者を追加するには、[新しいメールアドレスを追加]を選択します。
      - i. Amazon SNS のトピックを再度利用するには、既存を選択します。
        ii. SNS topic (SNS トピック) では、 リストから既存のAmazon SNS ピックの名前を選択します。

   d. 「Wheneverメトリックの統計」では、アラームの条件を次のように設定します。

      i. メトリクスがしきい値より大きい、小さい、またはしきい値と等しいのいずれかを指定します。
      ii. しきい値を指定します。
      iii. アラームを発生させるためにアラーム状態を維持する必要がある評価期間の数を指定します。
      iv. 評価期間の長さを指定します。

   e. [確認] を選択します。

> **Note**
>
> 指定した各 Amazon SNS 受信者には、 AWS 通知から確認メールが届きます。E メールには、受信者が購読を確認して通知を受け取るために必要なリンクが含まれています。

## SSR アプリの CloudWatch ログへのアクセス

Amplify は SSR ランタイムに関する情報を、 AWS アカウントの Amazon CloudWatch Logs に送信します。SSR アプリを Amplify ホスティングコンピューティングにデプロイする場合、アプリには、ユーザーの代わりに他のサービスを呼び出す際に Amplify が引き受ける IAM サービスロールが必要です。Amplify ホスティングコンピューティングにサービスロールを自動的に作成させることも、作成したロールを指定することもできます。

Amplify に IAM ロールの作成を許可することを選択した場合、そのロールにはすでに CloudWatch Logs を作成する権限が付与されています。独自の IAM ロールを作成する場合、Amplify が Amazon CloudWatch Logs にアクセスできるようにするには、ポリシーに次のアクセス権限を追加する必要があります。

```
logs:CreateLogStream
logs:CreateLogGroup
logs:DescribeLogGroups
logs:PutLogEvents
```

サービスロールの追加についての詳細は、「バックエンドリソースをデプロイするアクセス許可を持つサービスロールの追加」をご参照ください。サーバー側でレンダリングされたアプリを展開する詳細については、Amplify ホスティングでサーバーサイドレンダリングされたアプリのデプロイを参照してください。

SSR アプリケーションの Amplify ホスティングコンピューティングのログは、CloudWatch コンソールまたは Amplify コンソールで表示できます。Amplify コンソールでログを表示するには、次の手順に従います。

Amplify コンソールで SSR アプリケーションの CloudWatch ログを表示するには

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
2. CloudWatch ログを表示する SSR アプリを選択します。
3. ナビゲーションペインで、[モニタリング]、[ホスティングコンピューティングのログ] の順に選択します。

4. [ホスティングコンピューティングのログ] ページで、特定のブランチの CloudWatch ロググループを検索して選択します。

# Amplify アプリケーションのアクセスログの取得と分析

Amplify は、Amplify でホストしているすべてのアプリのアクセスログを保存します。アクセスログには、ホストされているアプリに対して行われたリクエストに関する情報が含まれています。Amplify は、アプリを削除するまで、アプリのすべてのアクセスログを保持します。アプリのすべてのアクセスログは、Amplify コンソールで使用できます。ただし、アクセスログに対する個々のリクエストは、指定した 2 週間に制限されます。

> **Warning**
>
> シークレット、認証情報、または機密データをパスまたはクエリパラメータとして URL に含めないでください。これらの値は、Amplify アプリケーションのアクセスログでプレーンテキストで表示できます。

Amplify は、お客様間で CloudFront ディストリビューションを再利用することはありません。Amplify は CloudFront ディストリビューションを事前に作成するので、新しいアプリをデプロイするときに CloudFront ディストリビューションが作成されるのを待つ必要はありません。これらのディストリビューションが Amplify アプリに割り当てられる前に、ボットからトラフィックを受信する可能性があります。ただし、割り当てられる前は常に「見つかりません」と応答するように設定されています。アプリのアクセスログにアプリを作成する前の期間のエントリーが含まれている場合、これらのエントリーはこのアクティビティに関連しています。

> **Important**
>
> ログは、すべてのリクエストを完全に課金するためのものではなく、コンテンツに対するリクエストの本質を把握するものとして使用することをお勧めします。CloudFront はベストエフォートベースでアクセスログを提供します。特定のリクエストのログエントリが、リクエストが実際に処理されてからかなり後に配信されることも、(まれに) 一切配信されないこともあります。ログエントリがアクセスログから省略された場合、アクセスログのエントリ数は、 AWS 請求レポートと使用状況レポートに表示される使用量と一致しません。

## アプリのアクセスログの取得

次の手順を使用して、Amplify アプリのアクセスログを取得します。

アクセスログを表示するには

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
2. ログを表示するアプリを選択します。
3. ナビゲーションペインで、[モニタリング]、[アクセスログ] の順に選択します。
4. [時間範囲の編集] を選択します。
5. [時間範囲の編集] ウィンドウで、以下の操作を行います。
   a. [開始日] では、ログを取得する 2 週間間隔の最初の日を指定します。
   b. [開始時間] では、ログの取得を開始する最初の日の時間を選択します。
   c. [確認] を選択します。
6. Amplify コンソールのアクセスログセクションには、指定した時間範囲のログが表示されます。[ダウンロード] を選択すると、ログが CSV 形式で保存されます。

## アクセスログの分析

アクセスログを分析するには、CSV ファイルを Amazon S3 バケットに保存します。アクセスログを分析する方法の 1 つとして Athena を使用する方法があります。Athena は、サービスのデータの分析に役立つインタラクティブなクエリ AWS サービスです。こちらの段階ごとの手順に従ってテーブルを作成できます。テーブルを作成した後、次のようにデータをクエリすることができます。

```
SELECT SUM(bytes) AS total_bytes
FROM logs
WHERE "date" BETWEEN DATE '2018-06-09' AND DATE '2018-06-11'
LIMIT 100;
```

# を使用した Amplify API コールのログ記録 AWS CloudTrail

AWS Amplify は AWS CloudTrail、Amplify のユーザー、ロール、または のサービスによって実行されたアクションを記録する AWS サービスである と統合されています。CloudTrail は、Amplify へのすべての API コールをイベントとしてキャプチャします。キャプチャされたコールには、Amplify コンソールからのコールと、Amplify API オペレーションへのコード呼び出しが含まれます。トレイル

を作成すると、Amplify のイベントを含む CloudTrail イベントの Amazon S3 バケットへの継続的な配信が可能になります。証跡を設定しない場合でも、CloudTrail コンソールの [イベント履歴] で最新のイベントを表示できます。CloudTrail で収集された情報を使用して、Amplify に対するリクエスト、リクエスト元の IP アドレス、リクエストしたユーザー、リクエスト日時、およびその他の詳細を確認できます。

CloudTrail の詳細については、AWS CloudTrail ユーザーガイドを参照してください。

## CloudTrail の Amplify 情報

CloudTrail はデフォルトで AWS アカウントで有効になっています。Amplify でアクティビティが発生すると、そのアクティビティはイベント履歴の他の AWS サービスイベントとともに CloudTrail イベントに記録されます。最近のイベントは、 AWS アカウントで表示、検索、ダウンロードできます。詳細については、AWS CloudTrail ユーザーガイドの「CloudTrail イベント履歴でのイベントの表示」を参照してください。

Amplify のイベントなど、 AWS アカウント内のイベントの継続的な記録については、証跡を作成します。証跡により、ログファイルを CloudTrail で Amazon S3 バケットに配信できます。デフォルトでは、コンソールで証跡を作成すると、すべての AWS リージョンに証跡が適用されます。証跡は、AWS パーティション内のすべてのリージョンからのイベントをログに記録し、指定した Amazon S3 バケットにログファイルを配信します。さらに、CloudTrail ログで収集したイベントデータをより詳細に分析し、それに基づいて対応するため、他の AWS サービスを構成できます。詳細については、AWS CloudTrail ユーザーガイドで次を参照してください。

- AWS アカウントの証跡の作成
- CloudTrail がサポートされているサービスと統合
- 「CloudTrail の Amazon SNS 通知の設定」
- 複数のリージョンから CloudTrail ログファイルを受け取るおよび複数のアカウントから CloudTrail ログファイルを受け取る

Amplify の操作はすべて CloudTrail によって記録され、AWS Amplify コンソール API リファレンス、AWS Amplify 管理 UI API リファレンス、および Amplify UI ビルダー API リファレンスに記録されます。例えば、`CreateApp`、`DeleteApp`、および `DeleteBackendEnvironment` オペレーションへの呼び出しによって CloudTrail ログファイルにエントリが生成されます。

各イベントまたはログエントリには、誰がリクエストを生成したかという情報が含まれます。アイデンティティ情報は、以下を判別するのに役立ちます。

- リクエストは root または AWS Identity and Access Management (IAM) ユーザー認証情報を使用して行われましたか。
- リクエストが、ロールまたはフェデレーションユーザーの一時的なセキュリティ認証情報を使用して送信されたか。
- リクエストは別の AWS サービスによって行われました。

詳細については、「AWS CloudTrail ユーザーガイド」の [CloudTrail userIdentity element] (CloudTrail userIdentity 要素) を参照してください。

## Amplify ログファイルエントリの概要

「トレイル」は、指定した Amazon S3 バケットにイベントをログファイルとして配信するように設定できます。CloudTrail のログファイルは、単一か複数のログエントリを含みます。イベントは、任意の出典からの単一のリクエストを表し、リクエストされたアクション、アクションの日時、リクエストパラメータなどに関する情報が含まれます。CloudTrail ログファイルは、パブリック API 呼び出しの順序付けられたスタックトレースではないため、特定の順序では表示されません。

次の例は、 AWS Amplify コンソール API リファレンス`ListApps`オペレーションを示す CloudTrail ログエントリを示しています。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::444455556666:user/Mary_Major",
        "accountId": "444455556666",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "Mary_Major",
        "sessionContext": {
            "sessionIssuer": {},
            "webIdFederationData": {},
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2021-01-12T05:48:10Z"
            }
        }
    },
    "eventTime": "2021-01-12T06:47:29Z",
    "eventSource": "amplify.amazonaws.com",
```

```
    "eventName": "ListApps",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.255",
    "userAgent": "aws-internal/3 aws-sdk-java/1.11.898
 Linux/4.9.230-0.1.ac.223.84.332.metal1.x86_64 OpenJDK_64-Bit_Server_VM/25.275-b01
 java/1.8.0_275 vendor/Oracle_Corporation",
    "requestParameters": {
        "maxResults": "100"
    },
    "responseElements": null,
    "requestID": "1c026d0b-3397-405a-95aa-aa43aexample",
    "eventID": "c5fca3fb-d148-4fa1-ba22-5fa63example",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "444455556666"
}
```

次の例は、 AWS Amplify Admin UI API Reference ListBackendJobsオペレーションを示す CloudTrail ログエントリを示しています。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::444455556666:user/Mary_Major",
        "accountId": "444455556666",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "Mary_Major",
        "sessionContext": {
            "sessionIssuer": {},
            "webIdFederationData": {},
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2021-01-13T00:47:25Z"
            }
        }
    },
    "eventTime": "2021-01-13T01:15:43Z",
    "eventSource": "amplifybackend.amazonaws.com",
    "eventName": "ListBackendJobs",
```

```
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.255",
    "userAgent": "aws-internal/3 aws-sdk-java/1.11.898
 Linux/4.9.230-0.1.ac.223.84.332.metal1.x86_64 OpenJDK_64-Bit_Server_VM/25.275-b01
 java/1.8.0_275 vendor/Oracle_Corporation",
    "requestParameters": {
        "appId": "d23mv2oexample",
        "backendEnvironmentName": "staging"
    },
    "responseElements": {
        "jobs": [
            {
                "appId": "d23mv2oexample",
                "backendEnvironmentName": "staging",
                "jobId": "ed63e9b2-dd1b-4bf2-895b-3d5dcexample",
                "operation": "CreateBackendAuth",
                "status": "COMPLETED",
                "createTime": "1610499932490",
                "updateTime": "1610500140053"
            },
            {
                "appId": "d23mv2oexample",
                "backendEnvironmentName": "staging",
                "jobId": "06904b10-a795-49c1-92b7-185dfexample",
                "operation": "CreateBackend",
                "status": "COMPLETED",
                "createTime": "1610499657938",
                "updateTime": "1610499704458"
            }
        ],
        "appId": "d23mv2oexample",
        "backendEnvironmentName": "staging"
    },
    "requestID": "7adfabd6-98d5-4b11-bd39-c7deaexample",
    "eventID": "68769310-c96c-4789-a6bb-68b52example",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "444455556666"
}
```

# Amplify アプリケーションでの IAM ロールの使用

IAM ロールは、特定の許可を持つ、IAM アイデンティティです。ロールのアクセス許可によって、アイデンティティが AWSでできることとできないことが決まります。 AWS アカウント で IAM ロールを作成し、それらを使用して Amplify ホスティングにアクセス許可を委任できます。ロールの詳細については、「IAM ユーザーガイド」の「IAM ロール」を参照してください。

次のタイプの IAM ロールを使用して、ユーザーに代わってアクションを実行するために必要なアクセス許可を Amplify ホスティングに付与したり、他の AWS リソースにアクセスするコンピューティングコードを実行したりすることができます。

IAM サービスロール

Amplify はこの IAM ロールを引き受け、ユーザーに代わってアクションを実行します。このロールは、バックエンドリソースを持つアプリケーションに必要です。

IAM SSR コンピューティングロール

サーバー側レンダリング (SSR) アプリケーションが特定の AWS リソースに安全にアクセスできるようにします。

IAM SSR CloudWatch Logs ロール

SSR アプリをデプロイする場合、アプリには、Amplify が Amazon CloudWatch Logs へのアクセスを許可するために Amplify が引き受ける IAM サービスロールが必要です。

トピック

- バックエンドリソースをデプロイするアクセス許可を持つサービスロールの追加
- AWS リソースへのアクセスを許可する SSR コンピューティングロールの追加
- CloudWatch Logs へのアクセス許可を持つサービスロールの追加

## バックエンドリソースをデプロイするアクセス許可を持つサービスロールの追加

Amplify では、フロントエンドでバックエンドリソースをデプロイするためのアクセス許可が必要です。このアクセス許可を付与するには、サービスロールを使用します。サービスロールは、ユーザー

に代わってバックエンドをデプロイ、作成、管理するアクセス許可を Amplify ホスティングに提供する AWS Identity and Access Management (IAM) ロールです。

IAM サービスロールが必要な新しいアプリを作成する場合、Amplify ホスティングにサービスロールを自動的に作成させることも、既に作成している IAM ロールを指定することもできます。このセクションでは、アカウント管理権限を持ち、Amplify アプリケーションが バックエンドをデプロイ、作成、管理に必要とするリソースへの直接アクセスを明示的に許可する、Amplify サービスロールを作成できるようになります。

## IAM コンソールで Amplify サービスロールを作成する

サービスロールを作成する

1. IAM コンソールを開き、左側のナビゲーションバーから [ロール] を選択して、[ロールの作成] を選択します。
2. [信頼されたエンティティを選択] ページで、[AWS サービス] を選択します。[ユースケース] で [Amplify - バックエンドデプロイ] (Amplify - Backend Deployment) を選択し、[次へ] を選択します。
3. [アクセス許可を追加] ページで [次へ] を選択してください。
4. [名前、表示、作成] ページで、[ロール名] に **`AmplifyConsoleServiceRole-AmplifyRole`** などのわかりやすい名前を入力します。
5. デフォルトをすべて受け入れて [ロールの作成] を選択します。
6. Amplify コンソールに戻り、ロールをアプリにアタッチします。
   - 新しいアプリをデプロイしている場合は、次の操作を行います:
     a. サービスロールのリストを更新します。
     b. 先ほど作成したロールを選択します。この例については、AmplifyConsoleServiceRole-AmplifyRole のようになります。
     c. [次へ] を選択し、手順に従ってアプリのデプロイを完了します。
   - 既存のアプリがある場合は、次の操作を行います:
     a. ナビゲーションペインで、[アプリの設定] を選択して、[IAM ロール] を選択します。
     b. [IAM ロール] ページの [サービスロール] セクションで、[編集] を選択します。
     c. [サービスロール] ページで、[サービスロール] リストから作成したロールを選択します。
     d. [保存] を選択します。

7. アプリのバックエンドリソースをデプロイするアクセス許可が Amplify に付与されました。

## 混乱した代理の問題を防ぐためのサービスロールの信頼ポリシーの編集

混乱した代理問題とは、アクションを実行する許可を持たないエンティティが、より高い特権を持つエンティティにそのアクションの実行を強制できるというセキュリティ問題です。詳細については、「サービス間の混乱した代理の防止」を参照してください。

現在、Amplify-Backend Deploymentサービスロールのデフォルトの信頼ポリシーでは、代理の混乱を防ぐためにaws:SourceArnとaws:SourceAccountのグローバルコンテキスト条件キーが適用されています。ただし、以前にアカウントにAmplify-Backend Deploymentロールを作成したことがある場合は、ロールの信頼ポリシーを更新してこれらの条件を追加することで、代理が混乱するのを防ぐことができます。

次の例を使用して、アカウント内のアプリへのアクセスを制限します。リージョンおよびアプリケーション ID をユーザー自身の情報などに置き換えます。

```
"Condition": {
      "ArnLike": {
        "aws:SourceArn": "arn:aws:amplify:us-east-1:123456789012:apps/*"
      },
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      }
    }
```

を使用してロールの信頼ポリシーを編集する手順については AWS マネジメントコンソール、IAM ユーザーガイドの「ロールの変更 (コンソール)」を参照してください。

# AWS リソースへのアクセスを許可する SSR コンピューティングロールの追加

この統合により、Amplify SSR コンピューティングサービスに IAM ロールを割り当てて、サーバー側レンダリング (SSR) アプリケーションがロールのアクセス許可に基づいて特定の AWS リソースに安全にアクセスできるようにします。たとえば、アプリケーションの SSR コンピューティング関数が、割り当てられた IAM ロールで定義されたアクセス許可に基づいて、 や Amazon S3 バケットなどの他の AWS サービス Amazon Bedrock やリソースに安全にアクセスすることを許可できます。

IAM SSR コンピューティングロールは一時的な認証情報を提供するため、環境変数で存続期間の長いセキュリティ認証情報をハードコードする必要はありません。IAM SSR コンピューティングロールの使用は、最小特権のアクセス許可を付与し、可能であれば短期認証情報を使用する AWS というセキュリティのベストプラクティスに沿ったものです。

このセクションの後半の手順では、カスタムアクセス許可を持つポリシーを作成し、そのポリシーをロールにアタッチする方法について説明します。ロールを作成するときは、ロールを引き受けるアクセス許可を Amplify に付与するカスタム信頼ポリシーをアタッチする必要があります。信頼関係が正しく定義されていない場合、ロールを追加しようとするとエラーが発生します。次のカスタム信頼ポリシーはロールを継承するための許可を Amplify に付与します。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Statement1",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "amplify.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Amplify コンソール、 AWS SDKs、または を使用して、 の IAM ロールを AWS アカウント 既存の SSR アプリケーションに関連付けることができます AWS CLI。アタッチしたロールは、Amplify SSR コンピューティングサービスに自動的に関連付けられ、他の AWS リソースにアクセスするために指定したアクセス許可が付与されます。アプリケーションのニーズが時間の経過とともに変化するにつれて、アプリケーションを再デプロイすることなく、アタッチされた IAM ロールを変更できます。これにより、柔軟性を得られ、アプリケーションのダウンタイムが短縮されます。

> **Important**
>
> セキュリティとコンプライアンスの目的を達成するようにアプリケーションを設定するのはお客様の責任です。これには、SSR コンピューティングロールの管理が含まれます。これは、ユースケースのサポートに必要な最小限のアクセス許可セットを持つように設定する必要があります。詳細については、「IAM SSR コンピューティングロールのセキュリティの管理」を参照してください。

## IAM コンソールでの SSR コンピューティングロールの作成

IAM SSR コンピューティングロールを Amplify アプリケーションにアタッチする前に、ロールが AWS アカウントに既に存在している必要があります。このセクションでは、IAM ポリシーを作成し、Amplify が特定の AWS リソースにアクセスするために引き受けることができるロールにアタッチする方法について説明します。

IAM ロールを作成するときは、最小特権のアクセス許可を付与する AWS ベストプラクティスに従うことをお勧めします。IAM SSR コンピューティングロールは SSR コンピューティング関数からのみ呼び出されるため、コードの実行に必要なアクセス許可のみを付与する必要があります。

AWS マネジメントコンソール、 AWS CLI、または SDKs を使用して、IAM でポリシーを作成できます。詳細については、「IAM ユーザーガイド」の「「カスタマー管理ポリシーでカスタム IAM アクセス許可を定義する」を参照してください。

次の手順では、IAM コンソールを使用して、Amplify コンピューティングサービスに付与するアクセス許可を定義する IAM ポリシーを作成する方法を示します。

IAM コンソール JSON ポリシーエディタでポリシーを作成するには

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。
2. 左側のナビゲーションペインで、[ポリシー] を選択します。
3. [Create policy] (ポリシーの作成) を選択します。
4. [ポリシーエディタ] セクションで、[JSON] オプションを選択します。
5. JSON ポリシードキュメントを入力するか貼り付けます。
6. ポリシーにアクセス権限を追加し終えたら、[次へ] を選択します。

7. [確認と作成] ページで、作成するポリシーの [ポリシー名] と [説明] (オプション) を入力します。このポリシーで定義されているアクセス許可 を確認して、ポリシーによって付与されたアクセス許可を確認します。
8. ポリシーを作成 をクリックして、新しいポリシーを保存します。

ポリシーを作成したら、次の手順を使用してポリシーを IAM ロールにアタッチします。

特定の AWS リソースに Amplify アクセス許可を付与するロールを作成するには

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。
2. コンソールのナビゲーションペインで、[ロール]、[ロールの作成] の順に選択します。
3. [Custom trust policy] (カスタム信頼ポリシー) ロールタイプを選択してください。
4. [Custom trust policy] (カスタム信頼ポリシー) セクションで、ロールのカスタム信頼ポリシーを入力します。ロールの信頼ポリシーが必要です。これは、ロールを引き受けるために信頼するプリンシパルを定義します。

   次の信頼ポリシーをコピーして貼り付け、このロールを引き受けるアクセス許可を Amplify サービスに付与します。

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Statement1",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "amplify.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

5. ポリシーの検証中に生成されたセキュリティ警告、エラー、または一般的な警告を解決してから、[次へ] を選択します。
6. [許可を追加] ページで、前の手順で作成したポリシーの名前を探し、選択します。次いで、[次へ] を選択します。
7. [ロール名] に、ロールの名前を入力します。ロール名は 内で一意である必要があります AWS アカウント。大文字と小文字は区別されません。例えば、**PRODROLE** と **prodrole** というロール名を両方作成することはできません。他の AWS リソースはロールを参照する可能性があるため、ロールの作成後にロールの名前を編集することはできません。
8. (オプション) [説明] には、新しいロールの説明を入力します。
9. (オプション) [ステップ 1: 信頼されたエンティティを選択する] または [ステップ 2: 許可を追加する] セクションで [編集] を選択し、ロールのカスタムポリシーと許可を編集します。
10. ロール情報を確認し、ロールの作成 を選択します。

## Amplify アプリへの IAM SSR コンピューティングロールの追加

で IAM ロールを作成したら AWS アカウント、Amplify コンソールでアプリに関連付けることができます。

Amplify コンソールで SSR コンピューティングロールをアプリに追加するには

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/amplify/ で Amplify コンソールを開きます。
2. [すべてのアプリ] ページで、コンピューティングロールを追加するアプリの名前を選択します。
3. ナビゲーションペインで、[アプリの設定] を選択して、[IAM ロール] を選択します。
4. [コンピューティングロール] セクションで、[編集] を選択します。
5. [デフォルトロール] リストで、アタッチするロールの名前を検索して選択します。この例では、前の手順で作成したロールの名前を選択できます。デフォルトでは、選択したロールはアプリのすべてのブランチに関連付けられます。

   ロールの信頼関係が正しく定義されていない場合、エラーが表示され、ロールを追加できなくなります。
6. (オプション) アプリケーションがパブリックリポジトリにあり、自動ブランチ作成を使用している場合、またはプルリクエストのウェブプレビューが有効になっている場合は、アプリケーションレベルのロールを使用することはお勧めしません。代わりに、コンピューティングロールは、特定のリソースへのアクセスを必要とするブランチにのみアタッチします。デフォルトのアプリ

ケーションレベルの動作を上書きし、特定のブランチにロールをアタッチするには、次の手順を実行します。

a. [ブランチ] には、使用するブランチの名前を選択します。

b. [コンピューティングロール]で、ブランチに関連付けるロールの名前を選択します。

7. [保存] を選択します。

## IAM SSR コンピューティングロールのセキュリティの管理

セキュリティは、 AWS とお客様の間の責任共有です。セキュリティとコンプライアンスの目的を達成するようにアプリケーションを設定するのはお客様の責任です。これには、SSR コンピューティングロールの管理が含まれます。これは、ユースケースのサポートに必要な最小限のアクセス許可セットを持つように設定する必要があります。指定する SSR コンピューティングロールの認証情報は、SSR 関数のランタイムですぐに使用できます。SSR コードが意図的に、バグのために、またはリモートコード実行 (RCE) を許可することによってこれらの認証情報を公開した場合、権限のないユーザーは SSR ロールとそのアクセス許可にアクセスできます。

パブリックリポジトリのアプリケーションが SSR コンピューティングロールを使用し、プルリクエストに自動ブランチ作成またはウェブプレビューを使用する場合は、ロールにアクセスできるブランチを慎重に管理する必要があります。アプリケーションレベルのロールを使用しないことをお勧めします。代わりに、ブランチレベルでコンピューティングロールをアタッチする必要があります。これにより、特定のリソースへのアクセスを必要とするブランチにのみアクセス許可を付与できます。

ロールの認証情報が公開されている場合は、次のアクションを実行して、ロールの認証情報へのすべてのアクセスを削除します。

1. すべてのセッションを取り消す

   ロールの認証情報に対するすべてのアクセス許可をすぐに取り消す手順については、「IAM ロールの一時的なセキュリティ認証情報を取り消す」を参照してください。

2. Amplify コンソールからロールを削除する

   このアクションはすぐに有効になります。アプリケーションを再デプロイする必要はありません。

Amplify コンソールでコンピューティングロールを削除するには

1. にサインイン AWS マネジメントコンソール し、https://console.aws.amazon.com/amplify/ で Amplify コンソールを開きます。
2. [すべてのアプリ] ページで、コンピューティングロールを削除するアプリの名前を選択します。
3. ナビゲーションペインで、[アプリの設定] を選択して、[IAM ロール] を選択します。
4. [コンピューティングロール] セクションで、[編集] を選択します。
5. デフォルトロールを削除するには、ロール名の右側にある X を選択します。
6. [保存] を選択します。

# CloudWatch Logs へのアクセス許可を持つサービスロールの追加

Amplify は SSR ランタイムに関する情報を、 AWS アカウントの Amazon CloudWatch Logs に送信します。SSR アプリをデプロイする場合、アプリには、ユーザーの代わりに他のサービスを呼び出す際に Amplify が引き受けるIAMサービスロールが必要です。Amplify ホスティングコンピューティングにサービスロールを自動的に作成させることも、作成したロールを指定することもできます。

Amplify に IAM ロールの作成を許可することを選択した場合、そのロールにはすでに CloudWatch Logs を作成する権限が付与されています。独自の IAM ロールを作成する場合、Amplify が Amazon CloudWatch Logs にアクセスできるようにするには、ポリシーに次のアクセス権限を追加する必要があります。

```
logs:CreateLogStream
logs:CreateLogGroup
logs:DescribeLogGroups
logs:PutLogEvents
```

# Git リポジトリの統合ウェブフック

Amplify ホスティングはウェブフックを使用して、Git リポジトリへの新しいコミット後にビルドを自動的に開始します。統合ウェブフック機能により、Amplify と Git プロバイダーの統合が改善され、より多くの Amplify アプリケーションを 1 つのリポジトリに接続できます。統合ウェブフックにより、Amplify はリポジトリ内のすべての関連アプリケーションに対してリージョンごとに 1 つのウェブフックを使用するようになりました。たとえば、リポジトリが米国東部 (バージニア北部) リージョンと米国西部 (オレゴン) リージョンの両方のアプリケーションに接続されている場合、2 つの統合ウェブフックがあります。

このリリース以前は、Amplify はリポジトリに関連付けられたアプリケーションごとに新しいウェブフックを作成していました。1 つのリポジトリに複数のアプリケーションがある場合、個々の Git プロバイダーによって適用されるウェブフック制限に達し、アプリケーションの追加を防ぐことができます。これは、1 つのリポジトリに複数のプロジェクトが存在するモノレポジトリで作業するチームにとって特に困難でした。

統合ウェブフックには以下の利点があります。

- Git プロバイダーのウェブフック制限を克服: 必要な数の Amplify アプリケーションを 1 つのリポジトリに接続できます。
- モノレポのサポート強化: 複数のプロジェクトが 1 つのリポジトリを共有するモノレポジトリを使用すると、柔軟性と効率が向上します。
- 管理の簡素化: 単一のリポジトリウェブフックで複数の Amplify アプリを管理すると、複雑さと潜在的な障害ポイントが軽減されます。
- ワークフロー統合の改善: Git プロバイダーによって割り当てられたウェブフックを、開発プロセスの他の重要なワークフローに使用できます。

## 統合ウェブフックの開始方法

新しいアプリケーションの作成

Git リポジトリから Amplify ホスティングに新しいアプリケーションをデプロイすると、統合ウェブフック機能がリポジトリに自動的に実装されます。新しいアプリケーションを作成する手順については、「Amplify ホスティングへのアプリのデプロイの概要」を参照してください。

既存のアプリケーションの更新

既存の Amplify アプリケーションの場合、Git リポジトリをアプリケーションに再接続して、既存のウェブフックを統一されたウェブフックに置き換える必要があります。Git プロバイダーで許可されているウェブフックの最大数にすでに達している場合、統合ウェブフックへの移行は成功しない可能性があります。この場合、再接続する前に少なくとも 1 つの既存のウェブフックを手動で削除します。

リポジトリには、異なる AWS リージョンにデプロイされる複数のアプリケーションを含めることができます。Amplify オペレーションはリージョンベースであるため、統合ウェブフックへの移行は、Amplify アプリを再接続したリージョンのウェブフックに対してのみ行われます。その結果、アプリケーション ID ベースのウェブフックとリージョンベースの統合ウェブフックの両方がリポジトリに表示される場合があります。

以下の手順に従って、既存の Amplify アプリを統合ウェブフックに移行します。

既存の Amplify アプリを統合ウェブフックに移行するには

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
2. 統合ウェブフックに移行するアプリを選択します。
3. ナビゲーションペインで [アプリの設定] を選択してから、[ブランチ設定] を選択します。
4. [ブランチ設定] ページで、[リポジトリの再接続] を選択します。
5. 統合ウェブフックへの移行が成功したことを確認するには、Git リポジトリのウェブフック設定に移動します。1 つのウェブフック URL が `https://amplify-webhooks.`*`Region`*`.amazonaws.com/`*`git-provider`* の形式で表示されます。

# Amplify のセキュリティ

でのクラウドセキュリティが最優先事項 AWS です。お客様は AWS 、セキュリティを最も重視する組織の要件を満たすように構築されたデータセンターとネットワークアーキテクチャからメリットを得られます。

セキュリティは、 AWS とお客様の間の責任共有です。責任共有モデルではこれをクラウドのセキュリティおよびクラウド内のセキュリティと説明しています。

- クラウドのセキュリティ – AWS クラウドで AWS サービスを実行するインフラストラクチャを保護する AWS 責任があります。 AWS また、 では、安全に使用できるサービスも提供しています。サードパーティーの監査者は、AWS コンプライアンスプログラムコンプライアンスプログラムの一環として、当社のセキュリティの有効性を定期的にテストおよび検証。が適用されるコンプライアンスプログラムの詳細については AWS Amplify、「コンプライアンスプログラムAWS による対象範囲内のサービスコンプライアンスプログラム」を参照してください。
- クラウドのセキュリティ – お客様の責任は、使用する AWS サービスによって決まります。また、ユーザーは、データの機密性、会社の要件、適用される法律や規制など、その他の要因についても責任を負います。

このドキュメントは、Amplify 使用時における責任共有モデルの適用法を理解するのに役立ちます。以下のトピックでは、セキュリティとコンプライアンスの目標を達成するように Amplify を設定する方法について説明します。また、Amplify リソースのモニタリングや保護に役立つ他の AWS サービスの使用方法についても説明します。

トピック

- Amplify の Identity and Access Management
- Amplify のデータ保護
- のコンプライアンス検証 AWS Amplify
- のインフラストラクチャセキュリティ AWS Amplify
- Amplify でのセキュリティイベントのログ記録とモニタリング
- サービス間の混乱した代理の防止
- Amplify のセキュリティベストプラクティス

## Amplify の Identity and Access Management

AWS Identity and Access Management (IAM) は、管理者が AWS リソースへのアクセスを安全に制御 AWS のサービス するのに役立つ です。IAM 管理者は、誰が認証(サインイン) され、Amplify リソースを使用する認可を受ける (アクセス許可がある) ことができるかを管理します。IAM は、追加料金なしで使用できる AWS のサービス です。

トピック



## オーディエンス

AWS Identity and Access Management (IAM) の使用方法は、ロールによって異なります。

- サービスユーザー - 機能にアクセスできない場合は、管理者にアクセス許可をリクエストします (「Amplify アイデンティティとアクセスのトラブルシューティング」を参照)。
- サービス管理者 - ユーザーアクセスを決定し、アクセス許可リクエストを送信します (「Amplify が IAM で機能する仕組み」を参照)
- IAM 管理者 - アクセスを管理するためのポリシーを作成します (「Amplify のアイデンティティベースのポリシー例」を参照)

## アイデンティティを使用した認証

認証とは、ID 認証情報 AWS を使用して にサインインする方法です。、IAM ユーザー AWS アカウントのルートユーザー、または IAM ロールを引き受けることで認証される必要があります。

AWS IAM Identity Center (IAM Identity Center)、シングルサインオン認証、Google/Facebook 認証情報などの ID ソースからの認証情報を使用して、フェデレーティッド ID としてサインインできます。サインインの詳細については、「AWS サインイン ユーザーガイド」の「AWS アカウントにサインインする方法」を参照してください。

プログラムによるアクセスの場合、 は SDK と CLI AWS を提供してリクエストを暗号化して署名します。詳細については、「IAM ユーザーガイド」の「API リクエストに対するAWS 署名バージョン 4」を参照してください。

### AWS アカウント ルートユーザー

を作成するときは AWS アカウント、すべての AWS のサービス および リソースへの完全なアクセス権を持つ AWS アカウント root ユーザーと呼ばれる 1 つのサインインアイデンティティから始めます。日常的なタスクには、ルートユーザーを使用しないことを強くお勧めします。ルートユーザー認証情報を必要とするタスクについては、「IAM ユーザーガイド」の「ルートユーザー認証情報が必要なタスク」を参照してください。

### フェデレーテッドアイデンティティ

ベストプラクティスとして、人間のユーザーが一時的な認証情報 AWS のサービス を使用して にアクセスするには、ID プロバイダーとのフェデレーションを使用する必要があります。

フェデレーティッド ID は、エンタープライズディレクトリ、ウェブ ID プロバイダー、または ID ソースの認証情報 AWS のサービス を使用して Directory Service にアクセスするユーザーです。フェデレーテッドアイデンティティは、一時的な認証情報を提供するロールを引き受けます。

アクセスを一元管理する場合は、 AWS IAM Identity Centerをお勧めします。詳細については、「AWS IAM Identity Center ユーザーガイド」の「IAM アイデンティティセンターとは」を参照してください。

### IAM ユーザーとグループ

IAM ユーザーは、特定の個人やアプリケーションに対する特定のアクセス許可を持つアイデンティティです。長期認証情報を持つ IAM ユーザーの代わりに一時的な認証情報を使用することをお勧めします。詳細については、IAM ユーザーガイドの「ID プロバイダーとのフェデレーションを使用して にアクセスする必要がある AWS」を参照してください。

IAM グループは、IAM ユーザーの集合を指定し、大量のユーザーに対するアクセス許可の管理を容易にします。詳細については、「IAM ユーザーガイド」の「IAM ユーザーに関するユースケース」を参照してください。

### IAM ロール

IAM ロールは、特定のアクセス許可を持つアイデンティであり、一時的な認証情報を提供します。ユーザーから IAM ロール (コンソール) に切り替えるか、 または API オペレーションを呼び出すこ

とで、ロールを引き受けることができます。 AWS CLI AWS 詳細については、「IAM ユーザーガイド」の「ロールを引き受けるための各種方法」を参照してください。

IAM ロールは、フェデレーションユーザーアクセス、一時的な IAM ユーザーのアクセス許可、クロスアカウントアクセス、クロスサービスアクセス、および Amazon EC2 で実行するアプリケーションに役立ちます。詳細については、IAM ユーザーガイド の IAM でのクロスアカウントリソースアクセス を参照してください。

## ポリシーを使用したアクセスの管理

でアクセスを制御する AWS には、ポリシーを作成し、ID AWS またはリソースにアタッチします。ポリシーは、アイデンティティまたはリソースに関連付けられたときにアクセス許可を定義します。は、プリンシパルがリクエストを行うときにこれらのポリシー AWS を評価します。ほとんどのポリシーは JSON ドキュメント AWS として に保存されます。JSON ポリシードキュメントの詳細については、「IAM ユーザーガイド」の「JSON ポリシー概要」を参照してください。

管理者は、ポリシーを使用して、どのプリンシパルがどのリソースに対して、どのような条件でアクションを実行できるかを定義することで、誰が何にアクセスできるかを指定します。

デフォルトでは、ユーザーやロールにアクセス許可はありません。IAM 管理者は IAM ポリシーを作成してロールに追加し、このロールをユーザーが引き受けられるようにします。IAM ポリシーは、オペレーションの実行方法を問わず、アクセス許可を定義します。

### アイデンティティベースのポリシー

アイデンティティベースのポリシーは、アイデンティティ (ユーザー、グループ、またはロール) にアタッチできる JSON アクセス許可ポリシードキュメントです。これらのポリシーは、アイデンティティがどのリソースに対してどのような条件下でどのようなアクションを実行できるかを制御します。アイデンティティベースポリシーの作成方法については、IAM ユーザーガイド の カスタマー管理ポリシーでカスタム IAM アクセス許可を定義する を参照してください。

アイデンティティベースのポリシーは、インラインポリシー (単一の ID に直接埋め込む) または管理ポリシー (複数の ID にアタッチされたスタンドアロンポリシー) にすることができます。管理ポリシーとインラインポリシーのいずれかを選択する方法については、「IAM ユーザーガイド」の「管理ポリシーとインラインポリシーのいずれかを選択する」を参照してください。

### リソースベースのポリシー

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。例としては、IAM ロール信頼ポリシーや Amazon S3 バケットポリシーなどがあります。リソースベースのポ

リシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。

リソースベースのポリシーは、そのサービス内にあるインラインポリシーです。リソースベースのポリシーでは、IAM の AWS マネージドポリシーを使用できません。

### その他のポリシータイプ

AWS は、より一般的なポリシータイプによって付与されるアクセス許可の最大数を設定できる追加のポリシータイプをサポートしています。

- アクセス許可の境界 – アイデンティティベースのポリシーで IAM エンティティに付与することのできるアクセス許可の数の上限を設定します。詳細については、「IAM ユーザーガイド」の「IAM エンティティのアクセス許可境界」を参照してください。
- サービスコントロールポリシー (SCP) - AWS Organizations内の組織または組織単位の最大のアクセス許可を指定します。詳細については、「AWS Organizations ユーザーガイド」の「サービスコントロールポリシー」を参照してください。
- リソースコントロールポリシー (RCP) – は、アカウント内のリソースで利用できる最大数のアクセス許可を定義します。詳細については、「AWS Organizations ユーザーガイド」の「リソースコントロールポリシー (RCP)」を参照してください。
- セッションポリシー – ロールまたはフェデレーションユーザーの一時セッションを作成する際にパラメータとして渡される高度なポリシーです。詳細については、「IAM ユーザーガイド」の「セッションポリシー」を参照してください。

### 複数のポリシータイプ

1 つのリクエストに複数のタイプのポリシーが適用されると、結果として作成されるアクセス許可を理解するのがさらに難しくなります。が複数のポリシータイプが関与する場合にリクエストを許可するかどうか AWS を決定する方法については、「IAM ユーザーガイド」の「ポリシー評価ロジック」を参照してください。

## Amplify が IAM で機能する仕組み

IAM を使用して Amplify へのアクセスを管理する前に、Amplify で利用できる IAM の機能について学びます。

Amplify で使用できる IAM の機能

| IAM 機能 | Amplify サポート |
| --- | --- |
| アイデンティティベースのポリシー | あり |
| リソースベースのポリシー | なし |
| ポリシーアクション | あり |
| ポリシーリソース | あり |
| ポリシー条件キー | あり |
| ACL | なし |
| ABAC (ポリシー内のタグ) | 部分的 |
| 一時認証情報 | あり |
| 転送アクセスセッション (FAS) | あり |
| サービスロール | あり |
| サービスリンクロール | いいえ |

Amplify およびその他の AWS のサービスがほとんどの IAM 機能と連携する方法の概要については、IAM ユーザーガイドのAWS 「IAM と連携する のサービス」を参照してください。

## Amplify のアイデンティティベースの ポリシー

アイデンティティベースのポリシーのサポート: あり

アイデンティティベースポリシーは、IAM ユーザー、ユーザーグループ、ロールなど、アイデンティティにアタッチできる JSON 許可ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。アイデンティティベースポリシーの作成方法については、「IAM ユーザーガイド」の「カスタマー管理ポリシーでカスタム IAM アクセス許可を定義する」を参照してください。

IAM アイデンティティベースのポリシーでは、許可または拒否するアクションとリソース、およびアクションを許可または拒否する条件を指定できます。JSON ポリシーで使用できるすべての要素に

ついて学ぶには、「IAM ユーザーガイド」の「IAM JSON ポリシーの要素のリファレンス」を参照してください。

Amplify のアイデンティティベースのポリシー例

Amplify のアイデンティティベースポリシーの例を確認するには、「Amplify のアイデンティティベースのポリシー例」を参照してください。

## Amplify 内のリソースベースのポリシー

リソースベースのポリシーのサポート: なし

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。リソースベースのポリシーには例として、IAM ロールの信頼ポリシーや Amazon S3 バケットポリシーがあげられます。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスをコントロールできます。ポリシーがアタッチされているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーで、プリンシパルを指定する必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーティッドユーザー、または を含めることができます AWS のサービス。

クロスアカウントアクセスを有効にするには、全体のアカウント、または別のアカウントの IAM エンティティを、リソースベースのポリシーのプリンシパルとして指定します。詳細については、IAM ユーザーガイドのIAM でのクロスアカウントリソースアクセスを参照してください。

## Amplify のポリシーアクション

ポリシーアクションのサポート: あり

管理者は JSON AWS ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

JSON ポリシーの `Action` 要素にはポリシー内のアクセスを許可または拒否するために使用できるアクションが記述されます。このアクションは関連付けられたオペレーションを実行するためのアクセス許可を付与するポリシーで使用されます。

Amplify アクションのリストを確認するには、「サービス認可リファレンス」の「AWS Amplifyで定義されるアクション」を参照してください。

Amplify のポリシーアクションは、アクションの前に次のプレフィックスを使用します。

```
amplify
```

単一のステートメントで複数のアクションを指定するには、アクションをカンマで区切ります。

```
"Action": [
      "amplify:action1",
      "amplify:action2"
         ]
```

Amplify のアイデンティティベースポリシーの例を確認するには、「Amplify のアイデンティティベースのポリシー例」を参照してください。

## Amplify のポリシーリソース

ポリシーリソースのサポート: あり

管理者は JSON AWS ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

`Resource` JSON ポリシー要素はアクションが適用されるオブジェクトを指定します。ベストプラクティスとして、Amazon リソースネーム (ARN) を使用してリソースを指定します。リソースレベルのアクセス許可をサポートしないアクションの場合は、ステートメントがすべてのリソースに適用されることを示すために、ワイルドカード (*) を使用します。

```
"Resource": "*"
```

Amplify リソースのタイプとその ARN のリストを確認するには、「サービス認可リファレンス」の「AWS Amplifyで定義されるリソース」を参照してください。どのアクションで各リソースの ARN を指定できるかについては、「AWS Amplifyで定義されるアクション」を参照してください。

Amplify のアイデンティティベースポリシーの例を確認するには、「Amplify のアイデンティティベースのポリシー例」を参照してください。

## Amplify のポリシー条件キー

サービス固有のポリシー条件キーのサポート: あり

管理者は JSON AWS ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

Condition 要素は、定義された基準に基づいてステートメントが実行される時期を指定します。イコールや未満などの条件演算子を使用して条件式を作成して、ポリシーの条件とリクエスト内の値を一致させることができます。すべての AWS グローバル条件キーを確認するには、「IAM ユーザーガイド」のAWS 「グローバル条件コンテキストキー」を参照してください。

Amplify の条件キーのリストを確認するには、「サービス認可リファレンス」の「AWS Amplifyの条件キー」を参照してください。条件キーを使用できるアクションとリソースについては、「 で定義されるアクション AWS Amplify」を参照してください。

Amplify のアイデンティティベースポリシーの例を確認するには、「Amplify のアイデンティティベースのポリシー例」を参照してください。

## Amplify のアクセスコントロールリスト (ACL)

ACL のサポート: なし

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするためのアクセス許可を持つかを制御します。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

## Amplify での属性ベースのアクセス制御 (ABAC)

ABAC (ポリシー内のタグ) のサポート: 一部

属性ベースのアクセスコントロール (ABAC) は、タグと呼ばれる属性に基づいてアクセス許可を定義する認可戦略です。IAM エンティティと AWS リソースにタグをアタッチし、プリンシパルのタグがリソースのタグと一致するときにオペレーションを許可するように ABAC ポリシーを設計できます。

タグに基づいてアクセスを管理するには、aws:ResourceTag/*key-name*、aws:RequestTag/*key-name*、または aws:TagKeys の条件キーを使用して、ポリシーの条件要素でタグ情報を提供します。

サービスがすべてのリソースタイプに対して 3 つの条件キーすべてをサポートする場合、そのサービスの値はありです。サービスが一部のリソースタイプに対してのみ 3 つの条件キーのすべてをサポートする場合、値は「部分的」になります。

ABAC の詳細については、「IAM ユーザーガイド」の「ABAC 認可でアクセス許可を定義する」を参照してください。ABAC をセットアップする手順を説明するチュートリアルについては、「IAM ユーザーガイド」の「属性ベースのアクセスコントロール (ABAC) を使用する」を参照してください。

## Amplify での一時的な認証情報の使用

一時的な認証情報のサポート: あり

一時的な認証情報は、 AWS リソースへの短期的なアクセスを提供し、フェデレーションまたはスイッチロールの使用時に自動的に作成されます。長期的なアクセスキーを使用する代わりに、一時的な認証情報を動的に生成 AWS することをお勧めします。詳細については、「IAM ユーザーガイド」の「IAM の一時的な認証情報」および「AWS のサービス と IAM との連携」を参照してください。

## Amplify の転送アクセスセッション

転送アクセスセッション (FAS) のサポート: あり

転送アクセスセッション (FAS) は、 を呼び出すプリンシパルのアクセス許可と AWS のサービス、ダウンストリームサービス AWS のサービス へのリクエストをリクエストする を使用します。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。

## Amplify のサービスロール

サービスロールのサポート: あり

サービスロールとは、サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除できます。詳細については、IAM ユーザーガイドの AWS のサービスに許可を委任するロールを作成するを参照してください。

> **Warning**
>
> サービスロールの許可を変更すると、の機能が破損する可能性があります。Amplify が指示する場合以外は、サービスロールを編集しないでください。

## Amplify のサービスリンクロール

サービスにリンクされたロールのサポート: なし

サービスにリンクされたロールは、 にリンクされたサービスロールの一種です AWS のサービス。サービスは、ユーザーに代わってアクションを実行するロールを引き受けることができます。サービスにリンクされたロールは に表示され AWS アカウント 、サービスによって所有されます。IAM 管理者は、サービスリンクロールのアクセス許可を表示できますが、編集することはできません。

サービスリンクロールの作成または管理の詳細については、「IAM ユーザーガイド」の「IAM と提携するAWS サービス」を参照してください。表の中から、[サービスにリンクされたロール] 列に Yes と記載されたサービスを見つけます。サービスリンクロールに関するドキュメントをサービスで表示するには、[あり]リンクを選択します。

## Amplify のアイデンティティベースのポリシー例

デフォルトでは、ユーザーとロールには Amplify リソースを作成または変更するアクセス許可がありません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。

これらのサンプルの JSON ポリシードキュメントを使用して IAM アイデンティティベースのポリシーを作成する方法については、「IAM ユーザーガイド」の「IAM ポリシーを作成する (コンソール)」を参照してください。

ACM が定義するアクションとリソースタイプ (リソースタイプごとの ARN の形式を含む) の詳細については、「サービス認可リファレンス」の「AWS Amplifyのアクション、リソース、および条件キー」を参照してください。

トピック

- ポリシーに関するベストプラクティス
- Amplify コンソールの使用
- 自分の権限の表示をユーザーに許可する

### ポリシーに関するベストプラクティス

ID ベースのポリシーは、ユーザーのアカウント内で誰かが Amplify リソースを作成、アクセス、または削除できるどうかを決定します。これらのアクションでは、 AWS アカウントに費用が発生する場合があります。アイデンティティベースポリシーを作成したり編集したりする際には、以下のガイドラインと推奨事項に従ってください:

- AWS 管理ポリシーを開始し、最小特権のアクセス許可に移行する – ユーザーとワークロードにアクセス許可の付与を開始するには、多くの一般的なユースケースにアクセス許可を付与するAWS 管理ポリシーを使用します。これらは で使用できます AWS アカウント。ユースケースに固有の AWS カスタマー管理ポリシーを定義することで、アクセス許可をさらに減らすことをお勧めします。詳細については、IAM ユーザーガイド の AWS マネージドポリシー または ジョブ機能のAWS マネージドポリシー を参照してください。
- 最小特権を適用する – IAM ポリシーでアクセス許可を設定する場合は、タスクの実行に必要な許可のみを付与します。これを行うには、特定の条件下で特定のリソースに対して実行できるアクションを定義します。これは、最小特権アクセス許可とも呼ばれています。IAM を使用して許可を適用する方法の詳細については、IAM ユーザーガイド の IAM でのポリシーとアクセス許可 を参照してください。
- IAM ポリシーで条件を使用してアクセスをさらに制限する - ポリシーに条件を追加して、アクションやリソースへのアクセスを制限できます。たとえば、ポリシー条件を記述して、すべてのリクエストを SSL を使用して送信するように指定できます。条件を使用して、サービスアクションが などの特定の を通じて使用されている場合に AWS のサービス、サービスアクションへのアクセスを許可することもできます CloudFormation。詳細については、IAM ユーザーガイド の IAM JSON ポリシー要素:条件 を参照してください。
- IAM アクセスアナライザー を使用して IAM ポリシーを検証し、安全で機能的な権限を確保する - IAM アクセスアナライザー は、新規および既存のポリシーを検証して、ポリシーが IAM ポリシー言語 (JSON) および IAM のベストプラクティスに準拠するようにします。IAM アクセスアナライザーは 100 を超えるポリシーチェックと実用的な推奨事項を提供し、安全で機能的なポリシーの作成をサポートします。詳細については、IAM ユーザーガイド の IAM Access Analyzer でポリシーを検証する を参照してください。
- 多要素認証 (MFA) を要求する – で IAM ユーザーまたはルートユーザーを必要とするシナリオがある場合は AWS アカウント、MFA をオンにしてセキュリティを強化します。API オペレーションが呼び出されるときに MFA を必須にするには、ポリシーに MFA 条件を追加します。詳細については、IAM ユーザーガイド の MFA を使用した安全な API アクセス を参照してください。

IAM でのベストプラクティスの詳細については、IAM ユーザーガイド の IAM でのセキュリティのベストプラクティス を参照してください。

## Amplify コンソールの使用

AWS Amplify コンソールにアクセスするには、最小限のアクセス許可のセットが必要です。これらのアクセス許可により、 の Amplify リソースの詳細を一覧表示および表示できます AWS アカウント。最小限必要な許可よりも制限が厳しいアイデンティティベースのポリシーを作成すると、そのポ

リシーを持つエンティティ (ユーザーまたはロール) に対してコンソールが意図したとおりに機能しません。

AWS CLI または AWS API のみを呼び出すユーザーには、最小限のコンソールアクセス許可を付与する必要はありません。代わりに、実行しようとしている API オペレーションに一致するアクションのみへのアクセスが許可されます。

Amplify Studio のリリースに伴い、アプリまたはバックエンドを削除するには `amplify` と `amplifybackend` 権限の両方が必要になりました。IAM ポリシーが `amplify` の権限のみを提供している場合、ユーザーがアプリを削除しようとすると権限エラーが発生します。ポリシーを作成する管理者の場合は、適切な権限を決定して、削除アクションを実行する必要があるユーザーに付与します。

ユーザーとロールが引き続き Amplify コンソールを使用できるようにするには、エンティティに Amplify `ConsoleAccess`または `ReadOnly` AWS 管理ポリシーもアタッチします。詳細については、「IAM ユーザーガイド」の「ユーザーへの許可の追加」を参照してください。

## 自分の権限の表示をユーザーに許可する

この例では、ユーザーアイデンティティにアタッチされたインラインおよびマネージドポリシーの表示を IAM ユーザーに許可するポリシーの作成方法を示します。このポリシーには、コンソールで、または AWS CLI または AWS API を使用してプログラムでこのアクションを実行するアクセス許可が含まれています。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
```

```
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

## AWS の マネージドポリシー AWS Amplify

AWS 管理ポリシーは、 によって作成および管理されるスタンドアロンポリシーです AWS。 AWS 管理ポリシーは、ユーザー、グループ、ロールにアクセス許可の割り当てを開始できるように、多くの一般的なユースケースにアクセス許可を付与するように設計されています。

AWS 管理ポリシーは、すべての AWS お客様が使用できるため、特定のユースケースに対して最小特権のアクセス許可を付与しない場合があることに注意してください。ユースケースに固有のカスタマー管理ポリシーを定義して、アクセス許可を絞り込むことをお勧めします。

AWS 管理ポリシーで定義されているアクセス許可は変更できません。が AWS マネージドポリシーで定義されたアクセス許可 AWS を更新すると、ポリシーがアタッチされているすべてのプリンシパル ID (ユーザー、グループ、ロール) に影響します。 AWS は、新しい が起動されるか、新しい API オペレーション AWS のサービス が既存のサービスで使用できるようになったときに、 AWS マネージドポリシーを更新する可能性が高くなります。

詳細については、「IAM ユーザーガイド」の「AWS マネージドポリシー」を参照してください。

### AWS マネージドポリシー: AdministratorAccess-Amplify

`AdministratorAccess-Amplify` ポリシーを IAM アイデンティティにアタッチできます。Amplify はまた、ユーザーに代わって Amplify がアクションを実行するのを許可するサービスロールにも、このポリシーをアタッチします。

Amplify コンソールでバックエンドをデプロイする場合は、Amplify が AWS リソースの作成と管理に使用する Amplify-Backend Deploymentサービスロールを作成する必要があります。IAM は AdministratorAccess-Amplify マネージドポリシーを Amplify-Backend Deployment サービスロールにアタッチします。

このポリシーは、アカウントに管理者権限を付与すると同時に、Amplify のアプリケーションがバックエンドの作成と管理に必要なリソースへの直接アクセスを明示的に許可します。

アクセス許可の詳細

このポリシーは、IAM アクションを含む複数の AWS サービスへのアクセスを提供します。これらのアクションにより、このポリシーを持つ ID は を使用して、任意のアクセス許可を持つ他の ID AWS Identity and Access Management を作成できます。これにより権限の昇格が可能になるため、このポリシーは AdministratorAccess ポリシーと同じくらい強力であると見なす必要があります。

このポリシーは、すべてのリソースに iam:PassRole アクション許可を付与します。これは Amazon Cognito のユーザープールの設定をサポートするために必要です。

このポリシーのアクセス許可を確認するには、「AWS マネージドポリシーリファレンス」の「AdministratorAccess-Amplify」を参照してください。

## AWS マネージドポリシー: AmplifyBackendDeployFullAccess

AmplifyBackendDeployFullAccess ポリシーを IAM アイデンティティにアタッチできます。

このポリシーは、 を使用して Amplify バックエンドリソースをデプロイするためのフルアクセス許可を Amplify に付与します AWS Cloud Development Kit (AWS CDK)。アクセス許可は、必要なAdministratorAccessポリシーアクセス許可を持つ AWS CDK ロールに委ねられます。

アクセス許可の詳細

このポリシーには以下を実行するためのアクセス許可が含まれています。

- Amplify– デプロイされたアプリケーションに関するメタデータを取得します。
- CloudFormation– Amplify マネージド型スタックを作成、更新、削除します。
- SSM– Amplify マネージド SSM パラメーターストアの String と SecureString のパラメーターを作成、更新、および削除します。
- AWS AppSync– AWS AppSync スキーマ、リゾルバー、関数のリソースを更新して取得します。この目的は、Gen 2 サンドボックスのホットスワップ機能をサポートすることです。

- Lambda– Amplify マネージド関数の設定を更新して取得します。この目的は、Gen 2 サンドボックスのホットスワップ機能をサポートすることです。

  Lambda 関数のタグを取得します。この目的は、顧客が定義した Lambda 関数をサポートすることです。
- Amazon S3– Amplify デプロイアセットを取得します。
- AWS Security Token Service– CLI AWS Cloud Development Kit (AWS CDK) がデプロイロールを引き受けることを有効にします。
- Amazon RDS– DB インスタンス、クラスター、プロキシのメタデータを読み取ります。
- Amazon EC2– サブネットのアベイラビリティーゾーン情報を読み取ります。
- CloudWatch Logs– 顧客の Lambda 関数のログを取得します。目的は、Amplify クラウド開発サンドボックス環境が Lambda 関数のログを顧客のターミナルにストリーミングできるようにすることです。

このポリシーのアクセス許可を確認するには、「AWS マネージドポリシーリファレンス」の「AmplifyBackendDeployFullAccess」を参照してください。

## AWS 管理ポリシーの Amplify 更新

このサービスがこれらの変更の追跡を開始してからの Amplify の AWS マネージドポリシーの更新に関する詳細を表示します。このページへの変更に関する自動アラートについては、のドキュメント履歴 AWS Amplify ページの RSS フィードを購読してください。

| 変更 | 説明 | 日付 |
| --- | --- | --- |
| AmplifyBackendDeployFullAccess – 既存のポリシーに更新します | logs:FilterLogEvents リソースに読み取りアクセスを追加して、Amplify が、カスタムロググループが作成された関数からログをストリーミングできるようにします。これは、Lambda 関数のログをストリーミングする既存の機能の拡張機能です。 | 2024 年 11 月 14 日 |

| 変更 | 説明 | 日付 |
| --- | --- | --- |
| AmplifyBackendDeployFullAccess – 既存のポリシーに更新します | `lambda:ListTags` および `logs:FilterLogEvents` のリソースに読み取りアクセスを追加して、顧客が定義した Lambda 関数をサポートします。これらのアクセス許可により、Amplify クラウド開発サンドボックス環境は Lambda 関数の口グを顧客のターミナルにストリーミングできます。 | 2024 年 7 月 18 日 |
| AmplifyBackendDeployFullAccess – 既存のポリシーに更新します | `arn:aws:ssm:*:*:parameter/cdk-bootstrap/*` リソースに読み取りアクセスを追加して、Amplify が顧客のアカウントで CDK ブートストラップバージョンを検出できるようにします。 | 2024 年 5 月 31 日 |

| 変更 | 説明 | 日付 |
| --- | --- | --- |
| AmplifyBackendDeployFullAccess – 既存のポリシーに更新します | Amazon RDS および Amazon EC2 の読み取り専用アクセス許可が、リソースとアカウントの両方の条件によってスコープされる新しい `AmplifyDiscoverRDSVpcConfig` ポリシーステートメントを追加します。これらのアクセス許可は、顧客が既存の SQL データベースから Typescript データスキーマを生成できるようにする Amplify Gen 2 `npx amplify generate schema-from-database` コマンドをサポートしています。<br><br>`rds:DescribeDBProxies` 、`rds:DescribeDBInstances` 、`rds:DescribeDBClusters` 、`rds:DescribeDBSubnetGroups` 、`ec2:DescribeSubnets` のアクセス許可を追加します。`npx amplify generate schema-from-database` コマンドでは、指定された DB ホストが Amazon RDS でホストされているかどうかをチェックし、SQL データベースにバックアップされた AWS AppSync API をセットアップ | 2024 年 4 月 17 日 |

| 変更 | 説明 | 日付 |
| --- | --- | --- |
| | するために必要な他のリソースをプロビジョニングするために必要な Amazon VPC 設定を自動生成するために、これらのアクセス許可が必要です。 | |
| AmplifyBackendDeployFullAccess – 既存のポリシーに更新します | `DeleteBranch` API が呼び出されたときにスタックの削除をサポートする `cloudformation:DeleteStack` ポリシーアクションを追加します。<br><br>`lambda:GetFunction` ポリシーアクションを追加して、ホットスワップ機能をサポートします。<br><br>Lambda 関数の更新をサポートする `lambda:UpdateFunctionConfiguration` ポリシーアクションを追加します。 | 2024 年 4 月 5 日 |
| AdministratorAccess-Amplify – 既存ポリシーの更新 | CloudFormation APIs への呼び出しをサポートする `cloudformation:TagResource` および アクセス`cloudformation:UnTagResource` 許可を追加します。 | 2024 年 4 月 4 日 |

| 変更 | 説明 | 日付 |
| --- | --- | --- |
| AmplifyBackendDeployFullAccess – 既存のポリシーに更新します | `lambda:InvokeFunction` ポリシーアクションを追加して、 AWS Cloud Development Kit (AWS CDK) ホットスワップをサポートします。 AWS CDK は Lambda 関数を直接呼び出して、Amazon S3 アセットのホットスワップを実行します。<br><br>`lambda:UpdateFunctionCode` ポリシーアクションを追加して、ホットスワップ機能をサポートします。 | 2024 年 1 月 2 日 |
| AmplifyBackendDeployFullAccess – 既存のポリシーに更新します | `UpdateApiKey` オペレーションをサポートするポリシーアクションを追加します。これは、リソースを削除することなく、サンドボックスを終了して再起動した後、アプリケーションを正常にデプロイできるようにするために必要です。 | 2023 年 11 月 17 日 |
| AmplifyBackendDeployFullAccess – 既存のポリシーに更新します | Amplify のアプリのデプロイをサポートする `amplify:GetBackendEnvironment` 権限を追加します。 | 2023 年 11 月 6 日 |

| 変更 | 説明 | 日付 |
| --- | --- | --- |
| AmplifyBackendDeployFullAccess – 新しいポリシー | Amplify は、Amplify のバックエンドリソースのデプロイに必要な最小限の権限を持つ新しいポリシーを追加しました。 | 2023 年 10 月 8 日 |
| AdministratorAccess-Amplify – 既存ポリシーの更新 | Amplify コマンドラインインターフェイス (CLI)に必要な `ecr:DescribeRepositories` 権限を追加します。 | 2023 年 6 月 1 日 |

| 変更 | 説明 | 日付 |
| --- | --- | --- |
| AdministratorAccess-Amplify – 既存ポリシーの更新 | AWS AppSync リソースからのタグの削除をサポートするポリシーアクションを追加します。<br><br>Amazon Polly のリソースをサポートするポリシーアクションを追加します。<br><br>OpenSearch のドメイン設定の更新をサポートするポリシーアクションを追加します。<br><br>AWS Identity and Access Management ロールからのタグの削除をサポートするポリシーアクションを追加します。<br><br>Amazon DynamoDB リソースからタグの削除をサポートするポリシーアクションを追加します。<br><br>Amplify の公開およびホスティングワークフローをサポートするには、`CLISDKCalls` ステートメントブロックに `cloudfront:GetCloudFrontOriginAccessIdentity` および `cloudfront:GetCloudFrontOriginAccessIdentityConfig` 権限を追加します。 | 2023 年 2 月 24 日 |

| 変更 | 説明 | 日付 |
| --- | --- | --- |
| | `CLIManageviaCFNPolicy` ステートメントブロックに `s3:PutBucketPublicAccessBlock` 許可を追加して、 AWS CLI が内部バケットで Amazon S3 パブリックアクセスブロック機能を有効にするという Amazon S3 セキュリティのベストプラクティスをサポートできるようにします。<br><br>`CLISDKCalls` ステートメントブロックに アクセス`cloudformation:DescribeStacks` 許可を追加して、Amplify バックエンドプロセッサでの再試行時の顧客の CloudFormation スタックの取得をサポートし、スタックの更新時に実行が重複しないようにします。<br><br>`cloudformation:ListStacks` 権限を `CLICloudformationPolicy` ステートメントブロックに追加します。この権限は、CloudFormation DescribeStacks のアクションを完全にサポートするために必要です。 | |

| 変更 | 説明 | 日付 |
| --- | --- | --- |
| AdministratorAccess-Amplify – 既存ポリシーの更新 | ポリシーアクションを追加して、Amplify のサーバー側レンダリング機能がアプリケーションメトリクスを顧客の AWS アカウントの CloudWatch にプッシュできるようにします。 | 2022 年 8 月 30 日 |
| AdministratorAccess-Amplify – 既存ポリシーの更新 | Amplify デプロイ Amazon S3 バケットへのパブリックアクセスをブロックするポリシーアクションを追加します。 | 2022 年 4 月 27 日 |
| AdministratorAccess-Amplify – 既存ポリシーの更新 | ユーザーがサーバーサイドレンダリング (SSR) されたアプリを削除できるようにするアクションを追加します。これにより、対応する CloudFront ディストリビューションを正常に削除することもできます。<br><br>顧客が Amplify CLI を使用して既存のイベントソースからのイベントを処理する別の Lambda 関数を指定できるようにするアクションを追加します。これらの変更 AWS Lambda により、 は UpdateEventSourceMapping アクションを実行できるようになります。 | 2022 年 4 月 17 日 |

| 変更 | 説明 | 日付 |
| --- | --- | --- |
| AdministratorAccess-Amplify – 既存ポリシーの更新 | すべてのリソースでAmplify UI Builder のアクションを有効にするためのポリシーアクションを追加します。 | 2021 年 12 月 2 日 |
| AdministratorAccess-Amplify – 既存ポリシーの更新 | ソーシャル ID プロバイダーを使用する Amazon Cognito の認証機能をサポートするポリシーアクションを追加します。<br><br>Lambda レイヤーをサポートするポリシーアクションを追加します。<br><br>Amplify Storage カテゴリをサポートするポリシーアクションを追加します。 | 2021 年 11 月 8 日 |

| 変更 | 説明 | 日付 |
| --- | --- | --- |
| AdministratorAccess-Amplify – 既存ポリシーの更新 | Amplify Interaction のカテゴリをサポートする Amazon Lex のアクションを追加します。<br><br>Amplify Predictions カテゴリをサポートする Amazon Rekognition アクションを追加します。<br><br>Amazon Cognito のユーザープールでの MFA 設定をサポートする Amazon Cognito アクションを追加します。<br><br>CloudFormation StackSetsをサポートする CloudFormation アクションを追加します。<br><br>Amplify Geo カテゴリをサポートする Amazon Location Service アクションを追加します。<br><br>Amplify の Lambda レイヤーをサポートする Lambda アクションを追加します。<br><br>CloudWatch Events をサポートする CloudWatch ログアクションを追加します。<br><br>Amplify Storage カテゴリをサポートする Amazon S3 アクションを追加します。<br><br>サーバー側レンダリング (SSR) アプリをサポートする | 2021 年 9 月 27 日 |

| 変更 | 説明 | 日付 |
| --- | --- | --- |
| | ポリシーアクションを追加します。 | |

| 変更 | 説明 | 日付 |
| --- | --- | --- |
| AdministratorAccess-Amplify – 既存ポリシーの更新 | すべての Amplify アクションを 1 つの `amplify:*` アクションに統合します。<br><br>顧客の Amazon S3 バケットの暗号化をサポートする Amazon S3 アクションを追加します。<br><br>権限境界が有効になっている Amplify のアプリをサポートする、IAM 権限境界アクションを追加します。<br><br>発信元の電話番号の表示、ならびに宛先の電話番号の表示、作成、検証、および削除をサポートする Amazon SNS アクションを追加します。<br><br>Amplify Studio: Amazon Cognito、 AWS Lambda、IAM、 CloudFormation ポリシーアクションを追加して、Amplify コンソールと Amplify Studio でバックエンドを管理できるようにします。<br><br>( AWS Systems Manager SSM) ポリシーステートメントを追加して、Amplify 環境シークレットを管理します。<br><br>Amplify アプリの Lambda レイヤーをサポートするアクションを追加します | 2021 年 7 月 28 日 |

| 変更 | 説明 | 日付 |
| --- | --- | --- |
| | CloudFormation `ListResources` 。 | |
| Amplify が変更の追跡を開始しました | Amplify は AWS 、管理ポリシーの変更の追跡を開始しました。 | 2021 年 7 月 28 日 |

# Amplify アイデンティティとアクセスのトラブルシューティング

以下の情報を使用して、Amplify と IAM の使用時に発生する可能性がある一般的な問題の診断と修正に役立てます。

トピック

- Amplify でアクションを実行する権限がない
- iam:PassRole を実行する権限がない
- AWS アカウント外のユーザーに Amplify リソースへのアクセスを許可したい

## Amplify でアクションを実行する権限がない

アクションを実行する権限がないというエラーが表示された場合は、そのアクションを実行できるようにポリシーを更新する必要があります。

次のエラー例は、`mateojackson` IAM ユーザーがコンソールを使用して、ある *`my-example-widget`* リソースに関する詳細情報を表示しようとしたことを想定して、その際に必要な `amplify:`*`GetWidget`* アクセス許可を持っていない場合に発生するものです。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 amplify:GetWidget on resource: my-example-widget
```

この場合、`amplify:`*`GetWidget`* アクションを使用して *`my-example-widget`* リソースへのアクセスを許可するように、`mateojackson` ユーザーのポリシーを更新する必要があります。

サポートが必要な場合は、 AWS 管理者にお問い合わせください。サインイン資格情報を提供した担当者が管理者です。

Amplify Studio のリリースに伴い、アプリまたはバックエンドを削除するには amplify と amplifybackend 権限の両方が必要になりました。管理者が amplify の権限のみを提供する IAM ポリシーを作成した場合、アプリを削除しようとすると権限エラーが発生します。

以下のエラー例は、mateojackson IAM ユーザーがコンソールを使用して架空の *example-amplify-app* リソースを削除しようとしているが、amplifybackend:*RemoveAllBackends* 権限がない場合に発生します。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 amplifybackend;:RemoveAllBackends on resource: example-amplify-app
```

この場合、Mateo は、amplifybackend:*RemoveAllBackends* アクションを使用して *example-amplify-app* リソースへのアクセスが許可されるように、管理者にポリシーの更新を依頼します。

## iam:PassRole を実行する権限がない

iam:PassRole アクションを実行する権限がないというエラーが表示された場合は、ポリシーを更新して Amplify にロールを渡せるようにする必要があります。

一部の AWS のサービス では、新しいサービスロールまたはサービスにリンクされたロールを作成する代わりに、既存のロールをそのサービスに渡すことができます。そのためには、サービスにロールを渡すアクセス許可が必要です。

次の例のエラーは、marymajor という IAM ユーザーがコンソールを使用して Amplify でアクションを実行しようとする場合に発生します。ただし、このアクションをサービスが実行するには、サービスロールから付与されたアクセス許可が必要です。Mary には、ロールをサービスに渡すアクセス許可がありません。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

この場合、Mary のポリシーを更新してメアリーに iam:PassRole アクションの実行を許可する必要があります。

サポートが必要な場合は、 AWS 管理者にお問い合わせください。サインイン資格情報を提供した担当者が管理者です。

## AWS アカウント外のユーザーに Amplify リソースへのアクセスを許可したい

他のアカウントのユーザーや組織外の人が、リソースにアクセスするために使用できるロールを作成できます。ロールの引き受けを委託するユーザーを指定できます。リソースベースのポリシーまた

はアクセスコントロールリスト (ACL) をサポートするサービスの場合、それらのポリシーを使用して、リソースへのアクセスを付与できます。

詳細については、以下を参照してください:

- Amplify がこれらの機能をサポートしているかどうかを確認するには、「Amplify が IAM で機能する仕組み」を参照してください。
- 所有 AWS アカウント している のリソースへのアクセスを提供する方法については、「IAM ユーザーガイド」の「所有 AWS アカウント している別の の IAM ユーザーへのアクセスを提供する」を参照してください。
- リソースへのアクセスをサードパーティーに提供する方法については AWS アカウント、IAM ユーザーガイドの「サードパーティー AWS アカウント が所有する へのアクセスを提供する」を参照してください。
- ID フェデレーションを介してアクセスを提供する方法については、IAM ユーザーガイド の 外部で認証されたユーザー (ID フェデレーション) へのアクセスの許可 を参照してください。
- クロスアカウントアクセスにおけるロールとリソースベースのポリシーの使用方法の違いについては、IAM ユーザーガイド の IAM でのクロスアカウントのリソースへのアクセス を参照してください。

# Amplify のデータ保護

AWS Amplify 責任 AWS 共有モデルに準拠し、 には、データ保護に関する規制とガイドラインが含まれています。 AWS は、すべての AWS サービスを実行するグローバルインフラストラクチャを保護する責任があります。 AWS は、このインフラストラクチャでホストされるデータの制御を維持します。 顧客コンテンツと個人データを処理するためのセキュリティ設定コントロールを含めます。 AWS 顧客および APN パートナー。 データコントローラーまたはデータ処理者として動作する は、AWS クラウドに保存した個人データについて責任を負います。

データ保護の目的で、 AWS アカウント 認証情報を保護し、 AWS IAM Identity Center または AWS Identity and Access Management (IAM) を使用して個々のユーザーを設定することをお勧めします。この方法により、それぞれのジョブを遂行するために必要なアクセス許可のみを各ユーザーに付与できます。また、次の方法でデータを保護することをお勧めします。

- 各アカウントで多要素認証 (MFA) を使用します。
- SSL/TLS を使用して AWS リソースと通信します。
- で API とユーザーアクティビティのログ記録を設定します AWS CloudTrail。

- AWS 暗号化ソリューションと、 サービス内のすべての AWS デフォルトのセキュリティコントロールを使用します。
- Amazon Macie などの高度なマネージドセキュリティサービスを使用します。これにより、Amazon S3 に保存される個人データの検出と保護が支援されます。

顧客のアカウント番号などの機密の識別情報は、[名前] フィールドなどの自由形式のフィールドに配置しないことを強くお勧めします。これは、コンソール、API、 AWS CLIまたは SDK を使用して Amplify または他の AWS のサービスを使用する場合も同様です。 AWS SDKs Amplify や他のサービスに入力したすべてのデータは、診断ログに取り込まれる可能性があります。外部サーバーへの URL を指定するときは、そのサーバーへのリクエストを検証するための認証情報を URL に含めないでください。

データ保護の詳細については、AWS セキュリティブログ のブログ投稿「AWS の責任共有モデルと GDPR」を参照してください。

## 保管中の暗号化

保存時の暗号化とは、保存中にデータを暗号化することで、不正なアクセスからデータを保護することです。Amplify は、 によって管理される Amazon S3 AWS KMS keys 用の を使用して、デフォルトでアプリケーションのビルドアーティファクトを暗号化します AWS Key Management Service。

Amplify は Amazon CloudFront を使用してアプリを顧客に提供します。CloudFront は、エッジロケーション POP (Point Of Presence) 用に暗号化された SSD、およびリージョナルエッジキャッシュ (REC) 用に暗号化された EBS ボリュームを使用します。CloudFront Functions の関数コードと設定は、エッジロケーション POP の暗号化された SSD や、CloudFront で使用されるその他のストレージロケーションに、常に暗号化された形式で保存されます。

## 転送中の暗号化

転送中の暗号化とは、通信エンドポイント間の移動中にデータが傍受されるのを防ぐことです。Amplify ホスティングは、デフォルトで転送中のデータの暗号化を提供します。顧客と Amplify 間、および Amplify とそのダウンストリーム依存関係間のすべての通信は、および ととのすべての通信は、署名バージョン 4 の署名プロセスで署名された TLS 接続を使用して保護されます。すべての Amplify ホスティングエンドポイントは AWS Private Certificate Authorityが管理する SHA-256 証明書を使用します。詳細については、「署名バージョン 4 の署名プロセス」および「AWS Private Certificate Authorityとは」を参照してください。

## 暗号化キーの管理

AWS Key Management Service (KMS) は AWS KMS keys、顧客データの暗号化に使用される暗号化キーを作成および制御するためのマネージドサービスです。 は、顧客に代わってデータを暗号化するための暗号化キー AWS Amplify を生成および管理します。お客様が管理する必要のある暗号化キーはありません。

# のコンプライアンス検証 AWS Amplify

サードパーティーの監査者は、複数のコンプライアンスプログラムの一環として AWS Amplify のセキュリティと AWS コンプライアンスを評価します。これには、SOC、PCI、ISO、HIPAA、MTCS、C5、K-ISMS、ENS High、OSPAR、HITRUST CSF、および FINMA が含まれます。

AWS のサービス が特定のコンプライアンスプログラムの対象であるかどうかを確認するには、「コンプライアンスAWS のサービス プログラムによる対象範囲内」の「コンプライアンス」を参照し、関心のあるコンプライアンスプログラムを選択します。一般的な情報については、AWS 「コンプライアンスプログラム」を参照してください。

を使用して、サードパーティーの監査レポートをダウンロードできます AWS Artifact。詳細については、「Downloading Reports in AWS Artifact」を参照してください。

を使用する際のお客様のコンプライアンス責任 AWS のサービス は、お客様のデータの機密性、貴社のコンプライアンス目的、適用される法律および規制によって決まります。を使用する際のコンプライアンス責任の詳細については AWS のサービス、AWS 「 セキュリティドキュメント」を参照してください。

# のインフラストラクチャセキュリティ AWS Amplify

マネージドサービスである AWS Amplify は、 AWS グローバルネットワークセキュリティで保護されています。 AWS セキュリティサービスと がインフラストラクチャ AWS を保護する方法については、AWS 「 クラウドセキュリティ」を参照してください。インフラストラクチャセキュリティのベストプラクティスを使用して環境を AWS 設計するには、「Security Pillar AWS Well-Architected Framework」の「Infrastructure Protection」を参照してください。

AWS 公開された API コールを使用して、ネットワーク経由で Amplify にアクセスします。クライアントは以下をサポートする必要があります。

- Transport Layer Security (TLS)。TLS 1.2 が必須で、TLS 1.3 をお勧めします。
- DHE (楕円ディフィー・ヘルマン鍵共有) や ECDHE (楕円曲線ディフィー・ヘルマン鍵共有) などの完全前方秘匿性 (PFS) による暗号スイート。これらのモードは Java 7 以降など、ほとんどの最新システムでサポートされています。

# Amplify でのセキュリティイベントのログ記録とモニタリング

モニタリングは、Amplify およびその他の AWS ソリューションの信頼性、可用性、パフォーマンスを維持する上で重要な部分です。 には、Amplify をモニタリングし、問題が発生したときに報告し、必要に応じて自動アクションを実行するための以下のモニタリングツール AWS が用意されています。

- Amazon CloudWatch は、リソース AWS と で実行するアプリケーションをリアルタイムでモニタリングします AWS。特定のメトリクスの収集と追跡、カスタマイズしたダッシュボードの作成、および指定したしきい値に達したときに通知したりアクションを実行したりするアラームの設定を行うことができます。例えば、CloudWatch で Amazon Elastic Compute Cloud (Amazon EC2) インスタンスの CPU 使用率などのメトリクスを追跡し、必要に応じて新しいインスタンスを自動的に起動することができます。Amplify での CloudWatch メトリクスとアラームの使用の詳細については、「Amplify アプリケーションのモニタリング」を参照してください。
- Amazon CloudWatch Logs によって Amazon EC2 インスタンス、 AWS CloudTrail、などからのログファイルのモニタリング、保存、アクセスができます。CloudWatch Logs は、ログファイル内の情報をモニタリングし、特定のしきい値が満たされたときに通知します。高い耐久性を備えたストレージにログデータをアーカイブすることも可能です。詳細については、「Amazon CloudWatch Logs ユーザーガイド」を参照してください。
- AWS CloudTrail は、 AWS アカウントによって、またはアカウントに代わって行われた API コールおよび関連イベントをキャプチャし、指定した Amazon Simple Storage Service (Amazon S3) バケットにログファイルを配信します。呼び出し元のユーザーとアカウント AWS、呼び出し元の送信元 IP アドレス、呼び出しの発生日時を特定できます。詳細については、「を使用した Amplify API コールのログ記録 AWS CloudTrail」を参照してください。
- Amazon EventBridge は、アプリケーションをさまざまなイベントソースのデータに簡単に接続できるようにするサーバーレスイベントバスサービスです。EventBridge が、お客様独自のアプリケーション、Software-as-a-Service (SaaS) アプリケーション、および AWS のサービスからリアルタイムデータのストリームを配信し、そのデータを AWS Lambdaなどのターゲットにルーティングします。これにより、サービスで発生したイベントをモニタリングし、イベント駆動型アーキテクチャを構築できます。詳細については、Amazon EventBridge ユーザーガイドを参照してください。

# サービス間の混乱した代理の防止

混乱した代理問題は、アクションを実行する許可を持たないエンティティが、より特権のあるエンティティにアクションを実行するように強制できるセキュリティの問題です。では AWS、サービス間のなりすましにより、混乱した代理問題が発生する可能性があります。サービス間でのなりすましは、1 つのサービス (呼び出し元サービス) が、別のサービス (呼び出し対象サービス) を呼び出すときに発生する可能性があります。呼び出し元サービスは、本来ならアクセスすることが許可されるべきではない方法でその許可を使用して、別のお客様のリソースに対する処理を実行するように操作される場合があります。これを防ぐため、 AWS では、アカウントのリソースへのアクセス権が付与されたサービスプリンシパルで、すべてのサービスのデータを保護するために役立つツールを提供しています。

リソースポリシーで aws:SourceArnおよび aws:SourceAccount グローバル条件コンテキストキーを使用して、 が別のサービス AWS Amplify に付与するアクセス許可をリソースに制限することをお勧めします。両方のグローバル条件コンテキストキーを使用しており、それらが同じポリシーステートメントで使用されるときは、aws:SourceAccount 値と、aws:SourceArn 値のアカウントが同じアカウント ID を使用する必要があります。

aws:SourceArn の値は Amplify アプリのブランチ ARN でなければなりません。この値を arn:*Partition*:amplify:*Region*:*Account*:apps/*AppId*/branches/*BranchName* 形式で指定します。

混乱した代理問題から保護するための最も効果的な方法は、リソースの完全な ARN を指定しながら、aws:SourceArn グローバル条件コンテキストキーを使用することです。リソースの完全な ARN が不明な場合や、複数のリソースを指定する場合は、aws:SourceArn グローバルコンテキスト条件キーを使用して、ARN の未知部分をワイルドカード (*) で表します。例えば、arn:aws:*servicename*::*123456789012*:* です。

以下の例は、アカウント内の Amplify アプリへのアクセスを制限し、混乱を招く代理問題を防ぐために適用できるロール信頼ポリシーを示しています。このポリシーを使用するには、ポリシー例の赤の斜体のテキストを、自分の情報に置き換えます。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Sid": "ConfusedDeputyPreventionExamplePolicy",
```

```
    "Effect": "Allow",
    "Principal": {
      "Service": [
          "amplify.me-south-1.amazonaws.com",
          "amplify.eu-south-1.amazonaws.com",
          "amplify.ap-east-1.amazonaws.com",
          "amplifybackend.amazonaws.com",
          "amplify.amazonaws.com"
        ]
    },
    "Action": "sts:AssumeRole",
    "Condition": {
      "ArnLike": {
        "aws:SourceArn": "arn:aws:amplify:us-east-1:123456789012:apps/*"
      },
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      }
    }
  }
}
```

次の例は、アカウント内の特定の Amplify のアプリへのアクセスを制限し、混乱を招く代理問題を防ぐために適用できるロール信頼ポリシーを示しています。このポリシーを使用するには、ポリシー例の赤の斜体のテキストを、自分の情報に置き換えます。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Sid": "ConfusedDeputyPreventionExamplePolicy",
    "Effect": "Allow",
    "Principal": {
      "Service": [
          "amplify.me-south-1.amazonaws.com",
          "amplify.eu-south-1.amazonaws.com",
          "amplify.ap-east-1.amazonaws.com",
          "amplifybackend.amazonaws.com",
          "amplify.amazonaws.com"
        ]
```

```
    },
    "Action": "sts:AssumeRole",
    "Condition": {
      "ArnLike": {
        "aws:SourceArn": "arn:aws:amplify:us-east-1:123456789012:apps/d123456789/
branches/*"
      },
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      }
    }
  }
}
```

# Amplify のセキュリティベストプラクティス

Amplify には、独自のセキュリティポリシーを開発および実装する際に考慮する必要のあるいくつかのセキュリティ機能が用意されています。以下のベストプラクティスは一般的なガイドラインであり、完全なセキュリティソリューションを提供するものではありません。これらのベストプラクティスは顧客の環境に必ずしも適切または十分でない可能性があるので、処方箋ではなく、あくまで有用な推奨事項とお考えください。

## Amplify のデフォルトドメインでの cookie の使用

Amplify を使用してウェブアプリをデプロイすると、Amplify はそれをデフォルトの `amplifyapp.com` ドメインでホストします。アプリは、`https://branch-name.d1m7bkiki6tdw1.amplifyapp.com` という形式の URL で表示できます。

Amplify のアプリケーションのセキュリティを強化するために、amplifyapp.com ドメインはパブリックサフィックスリスト (PSL) に登録されています。セキュリティを強化するために、Amplify アプリケーションのデフォルトドメイン名に機密性の高いCookieを設定する必要がある場合は、__Host-プレフィックスの付いたCookieを使用することをお勧めします。このプラクティスは、クロスサイトリクエストフォージェリ (CSRF) 攻撃からドメインを防ぐ際に役立ちます。詳細については、Mozilla 開発者ネットワークの「Set-Cookie」ページを参照してください。

# Amplify ホスティング Service Quotas

AWS Amplify ホスティングのサービスクォータは次のとおりです。Service Quotas (以前は制限とも呼ばれていました) は、 AWS アカウントのサービスリソースまたはオペレーションの最大数です。

新しい AWS アカウント では、アプリケーションと同時ジョブのクォータが減少しました。 は、使用量に基づいてこれらのクォータを自動的に AWS 引き上げます。また、クォータの引き上げをリクエストすることも可能です。

Service Quotas コンソールには、アカウントのクォータに関する情報が表示されます。Service Quotas コンソールを使用して、デフォルトのサービスクォータを表示したり、調整可能なクォータのクォータの引き上げをリクエストしたりすることができます。詳細については、「Service Quotas ユーザーガイド」の「クォータ引き上げのリクエスト」を参照してください。

| 名前 | デフォルト | 引き上げ可能 | 説明 |
| --- | --- | --- | --- |
| アプリケーション | サポートされている各リージョン: 25 | 可能 | このアカウントの現在のリージョンで Amplify AWS コンソールで作成できるアプリの最大数。 |
| アプリケーションごとのブランチ | サポートされている各リージョン: 50 | 不可 | このアカウントで現在のリージョンに作成できるアプリケーションごとのブランチの最大数 |
| アーティファクトサイズの構築 | サポートされている各リージョン: 5 GB | 不可 | アプリケーションビルドアーティファクトの最大サイズ (GB 単位)。ビルドアーティファクトは、ビルド後に Amplify AWS |

| 名前 | デフォルト | 引き上げ可能 | 説明 |
| --- | --- | --- | --- |
| | | | コンソールによってデプロイされます。 |
| キャッシュアーティファクトのサイズ | サポートされている各リージョン: 5 GB | 不可 | キャッシュアーティファクトの最大サイズ (GB 単位)。 |
| 同時ジョブ | サポートされている各リージョン：5 | あり | このアカウントで現在のリージョンに作成できる同時ジョブの最大数。 |
| アプリケーションごとのドメイン | サポートされている各リージョン：5 | あり | このアカウントで現在のリージョンに作成できるアプリケーションごとのドメインの最大数 |
| 環境キャッシュアーティファクトのサイズ | サポートされている各リージョン: 5 GB | 不可 | 環境キャッシュアーティファクトの最大サイズ (GB 単位)。 |
| 手動デプロイの ZIP ファイルサイズ | サポートされている各リージョン: 5 GB | 不可 | 手動デプロイ ZIP ファイルの最大サイズ (GB 単位)。 |
| 1 時間あたりのアプリ作成の最大数 | サポートされている各リージョン: 25 | 不可 | 現在のリージョンで、このアカウントで Amplify コンソールで 1 AWS 時間あたりに作成できるアプリの最大数。 |

| 名前 | デフォルト | 引き上げ可能 | 説明 |
| --- | --- | --- | --- |
| 1 秒あたりのリクエストトークン数 | サポートされている各リージョン: 20,000 | あり | アプリの 1 秒あたりのリクエストトークンの最大数。Amplify ホスティングは、消費するリソースの量 (処理時間とデータ転送) に基づいて、リクエストにトークンを割り当てます。 |
| ドメインごとのサブドメイン | サポートされている各リージョン: 50 | 不可 | 現在のリージョンのこのアカウントで作成できる、ドメインごとのサブドメインの最大数。 |
| アプリケーションごとのウェブフック | サポートされている各リージョン: 50 | 可能 | このアカウントで現在のリージョンに作成できるアプリケーションごとのウェブフックの最大数 |

Amplify Service Quotasに関する詳細については、「AWS 全般のリファレンス」の「AWS Amplify エンドポイントとクォータ」を参照してください。

# Amplify ホスティングのトラブルシューティング

Amplify ホスティングの操作中にエラーやデプロイの問題が発生した場合は、このセクションのトピックを参照してください。

トピック



## Amplify の一般的な問題のトラブルシューティング

以下の情報は、Amplify ホスティングの一般的な問題のトラブルシューティングに役立ちます。

トピック



### HTTP 429 ステータスコード (リクエストが多すぎる)

Amplify は、受信リクエストが消費する処理時間とデータ転送に基づいて、ウェブサイトへの 1 秒あたりのリクエスト数 (RPS) を制御します。アプリケーションが HTTP 429 ステータスコードを返した場合、受信リクエストは、アプリケーションに割り当てられた処理時間とデータ転送の時間を超えています。このアプリケーション制限は、Amplify の REQUEST_TOKENS_PER_SECOND サー

ビスクォータによって管理されます。クォータの詳細については、「Amplify ホスティング Service Quotas」を参照してください。

この問題を修正するには、アプリを最適化してリクエスト期間とデータ転送を短縮し、アプリの RPS を増やすことをお勧めします。例えば、同じ 20,000 トークンでは、100 ミリ秒以内に応答する高度に最適化された SSR ページは、レイテンシーが 200 ミリ秒を超えるページと比較しても、より高い RPS をサポートできます。

同様に、1 MB の応答サイズを返すアプリケーションは、250 KB の応答サイズを返すアプリケーションよりも多くのトークンを消費します。

また、特定の応答がキャッシュに保持される時間を最大化するように Cache-Control ヘッダーを設定して、Amazon CloudFront キャッシュを活用することをお勧めします。CloudFront キャッシュから送信されるリクエストは、レート制限にはカウントされません。各 CloudFront ディストリビューションは 1 秒あたり最大 250,000 件のリクエストを処理できるため、キャッシュを使用してアプリケーションを大幅にスケールアップすることができます。CloudFront キャッシュの詳細については、「Amazon CloudFront デベロッパーガイド」の「キャッシュの最適化と可用性」を参照してください。

## Amplify コンソールにアプリのビルドステータスと最終更新時間が表示されない

Amplify コンソールで [すべてのアプリ] ページに移動すると、現在のリージョンのアプリごとにタイルが表示されます。Deployed などのビルドステータスや、アプリに関して表示される最終更新時間などが表示されない場合、アプリには `Production` ステージブランチが関連付けられていません。

コンソールでアプリケーションを一覧表示するために、Amplify は `ListApps` API を使用します。Amplify は `ProductionBranch.status` 属性を使用してビルドステータスを表示し、`ProductionBranch.lastDeployTime` 属性を使用して最終更新時間を表示します。この API の詳細については、「Amplify Hosting API のドキュメント」の「ProductionBranch」を参照してください。

次の手順に従って、`Production` ステージをアプリケーションのブランチに関連付けます。

1. Amplify コンソールにサインインします。
2. [すべてのアプリ] ページで、更新するアプリケーションを選択します。
3. ナビゲーションペインで [アプリの設定] を選択してから、[ブランチ設定] を選択します。
4. [ブランチ設定] セクションで、[編集] を選択します。
5. [プロダクションブランチ] で、使用するブランチの名前を選択します。

6. [保存] を選択します。
7. [すべてのアプリ] ページに戻ります。これで、アプリのビルドステータスと最終更新時間が表示されます。

## 新しいプルリクエストのウェブプレビューが作成されていない

ウェブプレビュー機能を使用すると、プルリクエストからの変更を、統合ブランチにマージする前にプレビューできます。Web プレビューは、リポジトリに対して行われたすべてのプルリクエストを、メインサイトが使用している URL とはまったく異なる固有のプレビュー URL にデプロイします。

アプリのウェブプレビューを有効にしても、新しい PR 用に作成されていない場合は、次のいずれかが問題の原因であるかどうかを調べます。

1. アプリが最大 `Branches per app` のサービスクォータに達したかどうかを確認します。クォータの詳細については、「Amplify ホスティング Service Quotas」を参照してください。

   アプリごとに 50 ブランチのデフォルトのクォータ内に収まるようにするには、アプリで自動ブランチ削除を有効にすることを検討してください。これにより、リポジトリに存在しなくなったブランチがアカウントに蓄積されなくなります。
2. パブリック GitHub リポジトリを使用していて、Amplify アプリに IAM サービスロールがアタッチされている場合、Amplify はセキュリティ上の理由からプレビューを作成しません。例えば、バックエンドを備えたアプリや`WEB_COMPUTE`ホスティングプラットフォームにデプロイされるアプリには IAM サービスロールが必要です。そのため、これらの種類のアプリのリポジトリが公開されている場合、ウェブプレビューを有効にすることはできません。

   ウェブプレビューをアプリで機能させるには、サービスロールの関連付けを解除するか (アプリにバックエンドがないか、`WEB_COMPUTE` アプリでない場合)、GitHub リポジトリをプライベートにすることができます。

## 手動デプロイが Amplify コンソールで保留中のステータスでスタックしている

手動デプロイを使用すると、Git プロバイダーに接続しなくても、Amplify ホスティングでウェブアプリケーションを公開できます。次の 4 つのデプロイオプションの 1 つを使用できます。

1. Amplify コンソールでアプリケーションフォルダをドラッグアンドドロップする。

2. Amplify コンソールで .zip ファイル (サイトのビルドアーティファクトを含む) をドラッグアンドドロップする。
3. .zip ファイル (サイトのビルドアーティファクトを含む) を Amazon S3 バケットにアップロードし、そのバケットを Amplify コンソールのアプリに接続する。
4. Amplify コンソールで、.zip ファイル (サイトのビルドアーティファクトを含む) を指すパブリック URL を使用する。

Amplify コンソールで手動デプロイにアプリケーションフォルダを使用する場合、ドラッグアンドドロップ機能に関する問題が認識されています。これらのデプロイは、次の理由で失敗する可能性があります。

- 一時的なネットワークの問題が発生する。
- アップロード中にファイルに対するローカルの変更がある。
- ブラウザセッションが、大量の静的アセットを同時にアップロードしようとする。

ドラッグアンドドロップによるアップロードの信頼性の向上に取り組んでいますが、アプリケーションフォルダをドラッグアンドドロップするのではなく、.zip ファイルを使用することをお勧めします。

Amplify コンソールからのファイルのアップロードを回避し、手動デプロイの信頼性を高めるため、.zip ファイルを Amazon S3 バケットにアップロードすることを強くお勧めします。Amplify と Amazon S3 の統合により、このプロセスが簡素化されます。詳細については、「Amazon S3 バケットから Amplify への静的ウェブサイトのデプロイ」を参照してください。

## アプリケーションの Node.js バージョンを更新する必要がある

Node.js バージョン 14、16、18 を使用するアプリケーションの Amplify サポートは、2025 年 9 月 15 日に終了します。この日付以降の動作は、アプリケーションタイプによって異なります。

- SSR アプリケーション: 非推奨の Node.js バージョンを使用すると、ビルドエラーが発生します。Node.js 20 以降にアップグレードするまで、更新をデプロイすることはできません。
- SSR アプリケーション以外: buildspec またはライブパッケージの更新を通じて手動でインストールした場合、非推奨の Node.js バージョンを引き続き使用できます。

既にデプロイされているアプリケーションは、Node.js のバージョンに関係なく引き続き実行されます。

Amazon Linux 2023 ビルドイメージを使用している場合、デフォルトで Node.js バージョン 20 がサポートされています。2025 年 9 月 15 日以降、AL2023 イメージは Node.js 22 を自動的にサポートし、デフォルトの Node.js バージョンを 18 から 22 に変更します。

Amazon Linux 2 (AL2) は、Node.js バージョン 20 以降を自動的にはサポートしません。現在 AL2 を使用している場合は、AL2023 に移行することをお勧めします。Amplify コンソールでビルドイメージを変更できます。指定した Node.js バージョンをサポートするカスタムビルドイメージを使用することもできます。

アップグレードする前に、新しいブランチでアプリケーションをテストして、正しく機能することを確認することをお勧めします。

アップグレードオプション

Amplify コンソール

Amplify コンソールの [ライブパッケージのアップデート] 機能を使用して、使用する Node.js のバージョンを指定できます。手順については、「ビルドイメージでの特定のパッケージバージョンと依存関係バージョンの使用」を参照してください。

カスタムビルドイメージ

カスタムビルドイメージを使用していて、NVM がイメージにインストールされている場合は、Dockerfile に `nvm install 20` を追加できます。カスタムビルドイメージの要件と設定手順の詳細については、「ビルドイメージのカスタマイズ」を参照してください。

ビルド設定

preBuild コマンドセクションに `nvm use` コマンドを追加することで、アプリケーションの `amplify.yml` ビルド設定で使用する Node.js バージョンを指定できます。アプリケーションのビルド設定を更新する手順については、「Amplify アプリケーションのビルド設定の構成」を参照してください。

次の例は、ビルド設定をカスタマイズしてデフォルトの Node.js バージョンを Node.js 18 に設定し、`node-20` という名前のテストブランチで Node.js バージョン 20 にアップグレードする方法を示しています。

```
frontend:
  phases:
    preBuild:
      commands:
        - nvm use 18
```

```
        - if [ "${AWS_BRANCH}" = "node-20" ]; then nvm use 20; fi
```

> **Warning**
>
> preBuild コマンドは、ライブパッケージの更新後に実行されることに注意してください。nvm use コマンドで指定された Node.js バージョンは、ライブパッケージの更新で設定された Node.js バージョンを上書きします。

# Amazon Linux 2023 ビルドイメージの問題のトラブルシューティング

以下の情報は、Amazon Linux 2023 (AL2023) のビルドイメージで発生した問題をトラブルシューティングする際に役立ちます。

トピック



## Python ランタイムで Amplify 関数を実行したい

Amplify ホスティングは、新しいアプリケーションをデプロイするときに、デフォルトで Amazon Linux 2023 ビルドイメージを使用するようになりました。AL2023 には、Python バージョン 3.8、3.9、3.10、3.11 がプリインストールされています。

Amazon Linux 2 イメージと後方互換性があるので、AL2023 ビルドイメージには、古いバージョンの Python がプリインストールされたシンボリックリンクがあります。

デフォルトでは、Python バージョン 3.10 がグローバルに使用されます。特定の Python バージョンを使用して関数を構築するには、アプリケーションのビルド仕様ファイルで次のコマンドを実行します。

```
version: 1
backend:
  phases:
    build:
      commands:
        # use a python version globally
```

```
        - pyenv global 3.11
        # verify python version
        - python --version
        # install pipenv
        - pip install --user pipenv
        # add to path
        - export PATH=$PATH:/root/.local/bin
        # verify pipenv version
        - pipenv --version
        - amplifyPush --simple
```

## スーパーユーザーまたはルート権限を必要とするコマンドを実行したい

Amazon Linux 2023 のビルドイメージを使用していて、スーパーユーザーまたはルート権限を必要とするシステムコマンドを実行するときにエラーが発生した場合は、Linux sudo コマンドを使用してこれらのコマンドを実行する必要があります。例えば、`yum install -y gcc` の実行中にエラーが発生した場合は、`sudo yum install -y gcc` を使用します。

Amazon Linux 2 のビルドイメージでは、ルートユーザーを使用しましたが、Amplify の AL2023 イメージは、カスタム `amplify` ユーザーでコードを実行します。Amplify は、Linux sudo コマンドを使用してコマンドを実行する権限をこのユーザーに付与します。スーパーユーザー権限が必要なコマンドには sudo を使用することをお勧めします。

# ビルドの問題のトラブルシューティング

Amplify アプリケーションの作成またはビルド時に問題が発生した場合は、このセクションのトピックを参照してください。

トピック



## リポジトリへの新しいコミットが Amplify ビルドをトリガーしない

Git リポジトリへの新しいコミットが Amplify ビルドをトリガーしない場合は、ウェブフックがまだリポジトリに存在することを確認します。存在する場合は、ウェブフックリクエストの履歴をチェッ

クして、障害がないかどうかを確認します。Amplify の受信ウェブフックのペイロードサイズ制限は 256 KB です。変更されたファイルが多数あるリポジトリにコミットをプッシュすると、この制限を超え、ビルドがトリガーされない可能性があります。

## 新しいアプリケーションを作成するときに、リポジトリ名が Amplify コンソールに表示されない

Amplify コンソールで新しいアプリケーションを作成するときに、[リポジトリとブランチの追加] ページで、組織で使用可能なリポジトリから選択できます。ターゲットリポジトリが最近更新されていない場合、リストに表示されないことがあります。これは、組織に多数のリポジトリがある場合に発生する可能性があります。この問題を解決するには、コミットをリポジトリにプッシュし、コンソールでリポジトリリストを更新します。これにより、リポジトリが表示されます。

## ビルドが **Cannot find module aws-exports** エラーで失敗する (Gen 1 アプリのみ)

ビルド中にアプリケーションが `aws-exports.js` ファイルを見つけられない場合、次のエラーが返されます。

```
TS2307: Cannot find module 'aws-exports'
```

Amplify コマンドラインインターフェイス (CLI) は、バックエンドビルド中に `aws-exports.js` ファイルを生成します。このエラーを解決するには、ビルドで使用する `aws-exports.js` ファイルを作成する必要があります。ビルド仕様に次のコードを追加して、ファイルを作成します。

```
backend:
  phases:
    build:
      commands:
        - "# Execute Amplify CLI with the helper script"
        - amplifyPush --simple
```

Amplify アプリのビルド仕様設定の完全な例については、「ビルド仕様 YAML 構文のリファレンス」を参照してください。

## ビルドタイムアウトを上書きしたい

デフォルトのタイムアウト値は 30 分です。_BUILD_TIMEOUT 環境変数を使用して、デフォルトのビルドタイムアウトを上書きできます。ビルドの最小タイムアウトは 5 分 です。ビルドの最大タイムアウトは 120 分です。

Amplify コンソールでアプリケーションの環境変数を設定する手順については、「環境変数の設定」を参照してください。

# カスタムドメインのトラブルシューティング

Amplify コンソールのアプリケーションにカスタムドメインを接続する際に問題が発生した場合は、このセクションのトピックを参考にしてください。

ここで問題の解決策が見つからない場合は、 サポートにお問い合わせください。詳細については、「AWS サポート ユーザーガイド」の「サポートケースの作成」を参照してください。

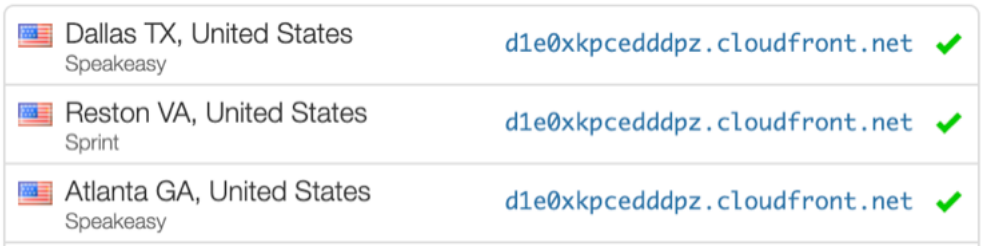
トピック

- CNAME が解決されることを確認する必要がある
- サードパーティーでホストされているドメインが [Pending Verification] (検証待ち) 状態のままになっている
- Amazon Route 53 でホストされているドメインが [Pending Verification] (検証待ち) 状態のままになっている
- マルチレベルサブドメインを持つアプリが [Pending Verification] (検証待ち) 状態でスタックしている
- DNS プロバイダーが完全修飾ドメイン名の A レコードをサポートしていない
- 「CNAMEAlreadyExistsException」エラーが発生した
- 「追加の検証が必要です」というエラーが表示されます。
- CloudFront URL に 404 エラーが出る
- 自分のドメインにアクセスしたときに SSL 証明書または HTTPS エラーが発生する。
- ドメインリダイレクトでサポートされていないパスコンポーネント
- クロスアカウントドメインの関連付けで 400 エラーが発生する

## CNAME が解決されることを確認する必要がある

1. サードパーティのドメインプロバイダーで DNS レコードを更新したら、dig などのツールや https://www.whatsmydns.net/ などの無料ウェブサイトを使用して、CNAME レコードが正しく解決されているかどうかを確認できます。次のスクリーンショットは、whatsmydns.net を使用して www.example.com というドメインの CNAME レコードを確認する方法を示しています。


2. [検索] を選択すると、whatsmydns.net に CNAME の検索結果が表示されます。次のスクリーンショットは、CNAME が cloudfront.net URL に正しく解決されることを確認する結果のリストの例です。




## サードパーティーでホストされているドメインが [Pending Verification] (検証待ち) 状態のままになっている

1. カスタムドメインが「検証待ち」の状態のままになっている場合は、CNAME レコードが解決中であることを確認してください。このタスクの実行方法については、前述のトラブルシューティングのトピック「CNAME 解決を確認する方法」を参照してください。
2. CNAME レコードが解決されない場合は、ドメインプロバイダーの DNS 設定に CNAME エントリが存在することを確認してください。

> **Important**
>
> カスタムドメインを作成したらすぐに CNAME レコードを更新することが重要です。アプリが Amplify コンソールで作成されると、CNAME レコードが数分ごとにチェックされ、解決したかどうかを判別します。1 時間経っても解決しない場合は、数時間ごとに

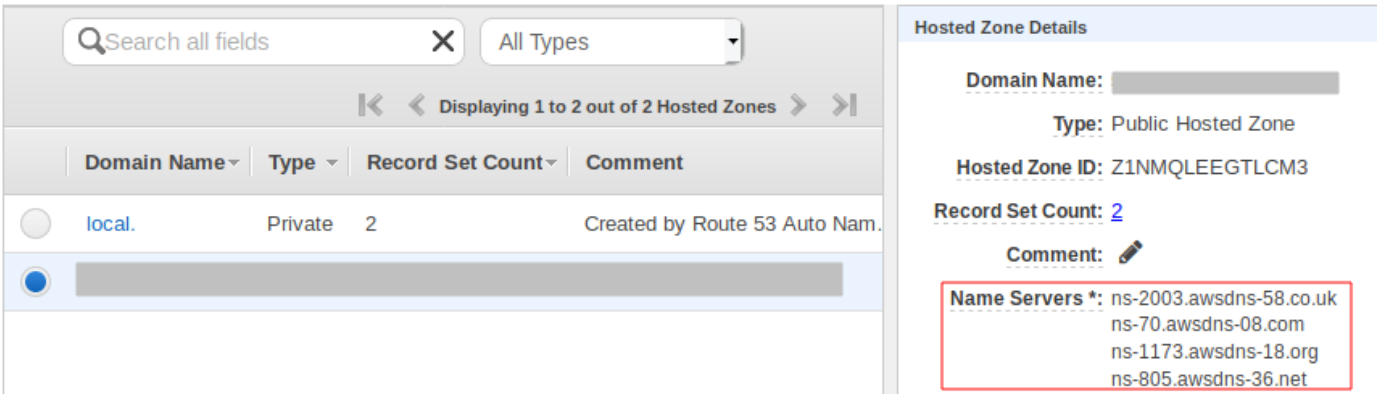
チェックが行われるため、ドメインを使用する準備ができるまでに時間がかかる可能性があります。アプリを作成してから数時間後に CNAME レコードを追加または更新した場合、これがアプリが「検証保留中」の状態で停止する最も可能性の高い原因です。

3. CNAME レコードが存在することが確認できた場合は、DNS プロバイダーに問題がある可能性があります。DNS 検証 CNAME が解決しない理由を診断するには、DNS プロバイダーに連絡するか、DNS を Route 53 に移行することができます。詳細については、「Amazon Route 53 を既存ドメインの DNS サービスにする」を参照してください。

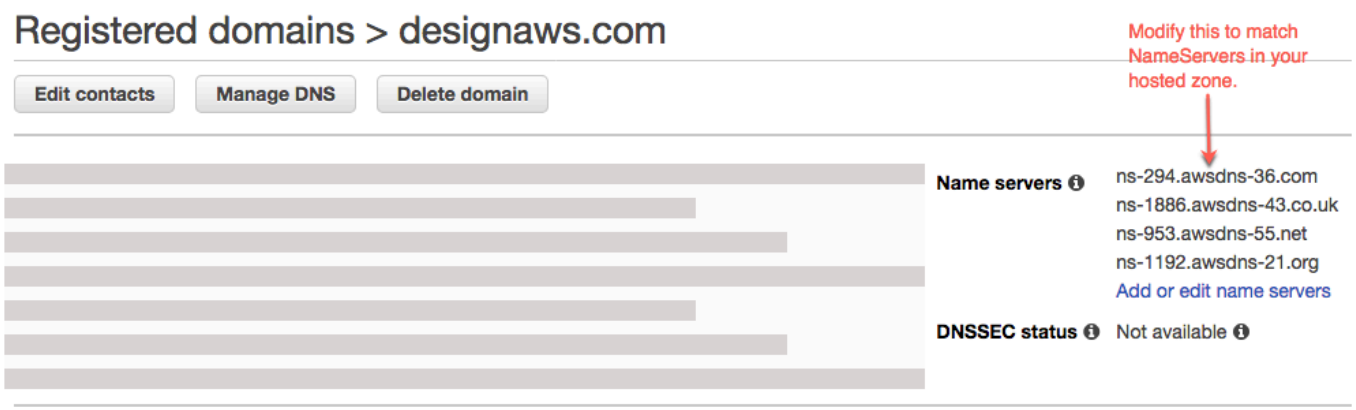
## Amazon Route 53 でホストされているドメインが [Pending Verification] (検証待ち) 状態のままになっている

ドメインを Amazon Route 53 に移行した場合、ドメインのネームサーバーが、アプリの作成時に Amplify によって発行されたものとは異なる可能性があります。エラーの原因を診断するには、次の手順を実行します。

1. Amazon Route 53 コンソールにサインインします
2. ナビゲーションペインで、[ホストゾーン] をクリックし、検証する必要のあるドメインの名前を選択します。
3. 「ホストゾーンの詳細」セクションのネームサーバーの値を記録します。次のステップを完了するには、これらの値が必要です。次の Route 53 コンソールのスクリーンショットでは、右下隅にネームサーバー値の場所が表示されています。




4. ナビゲーションペインで [Registered Domains] をクリックします。[登録済みドメイン] セクションに表示されるネームサーバーが、前のステップで [ホストゾーンの詳細] セクションに記録したネームサーバーの値と一致することを確認します。一致しない場合は、ネームサーバーの値をホストゾーンの値と一致するように編集します。次の Route 53 コンソールのスクリーンショットでは、右側にネームサーバー値の場所が表示されています。




5. これでも問題が解決しない場合は、 サポートにお問い合わせください。詳細については、「AWS サポート ユーザーガイド」の「サポートケースの作成」を参照してください。

## マルチレベルサブドメインを持つアプリが [Pending Verification] (検証待ち) 状態でスタックしている

サードパーティーの DNS プロバイダーに接続するときに、マルチレベルサブドメインを持つアプリが [Pending Verification] (検証待ち) 状態でスタックしている場合、DNS レコードの形式に問題がある可能性があります。一部の DNS プロバイダーは、セカンドレベルドメイン (SLD) とトップレベルドメイン (TLD) のドメインのサフィックスをレコードに自動的に追加します。SLD と TLD を含む形式でもドメインを指定すると、ドメイン検証の問題が発生する可能性があります。

ドメインを接続するときは、まず Amplify が提供する完全な形式、たとえば `_hash.docs.backend.example.com` を使用してドメイン名を指定してみてください。SSL 設定が [Pending Verification] (検証待ち) 状態でスタックする場合は、レコードから TLD と SLD を削除してみてください。たとえば、完全な形式が `_hash.docs.backend.example.com` の場合、`_hash.docs.backend` を指定します。レコードが伝播されるまで 15～30 分待ちます。次に、MX Toolbox などのツールを使用して、検証プロセスが機能しているかどうかを確認します。

## DNS プロバイダーが完全修飾ドメイン名の A レコードをサポートしていない

一部の DNS プロバイダーは、`example.cloudfront.net` などの完全修飾ドメイン名 (FQDN) の A レコードをサポートしていません。たとえば、Cloudflare A records は IPv4 アドレスのみを書き込み、FQDNs をサポートしていません。この制限を回避するには、DNS 設定で A records ではなく CNAME レコードを使用することをお勧めします。

たとえば、次の DNS 設定では A record が使用されています。

```
A     | @ | ***.cloudfront.net
CNAME | www | ***.cloudfront.net
```

CNAME レコードのみを使用するように、次の DNS 設定を変更します。

```
CNAME | @ | ***.cloudfront.net
CNAME | www | ***.cloudfront.net
```

この回避策により、Cloudflare のシステムで A records の IPv4-onlyの制限を回避しながら、apex ドメイン (@ レコード) を CloudFront などのサービスに適切にポイントできます。

## 「CNAMEAlreadyExistsException」エラーが発生した

CNAMEAlreadyExistsException エラーが発生した場合、接続しようとしたホスト名のいずれか (サブドメイン、または apex ドメインなど) が別の Amazon CloudFront ディストリビューションに既にデプロイされていることを意味します。エラーの原因は、現在のホスティングプロバイダーと DNS プロバイダーによって異なります。

`example.com` や `sub.example.com` などの CNAME エイリアスは、一度に 1 つの CloudFront ディストリビューションにのみ関連付けることができます。CNAMEAlreadyExistsException は、ドメインが同じ 内または別のアカウントにある AWS アカウント可能性のある別の CloudFront ディストリビューションに既に関連付けられていることを示します。Amplify ホスティングによって作成された新しいディストリビューションを機能させるには、ドメインと以前の CloudFront ディストリビューションとの関連付けを解除する必要があります。お客様またはお客様の組織が複数の を所有している場合は、複数のアカウントをチェックする必要がある場合があります AWS アカウント。

CNAMEAlreadyExistsException エラーの原因を診断するには、次の手順を実行します。

1. Amazon CloudFront コンソールにサインインし、このドメインが他のディストリビューションにデプロイされていないことを確認します。CloudFront ディストリビューションには、一度に 1 つの CNAME レコードのみをアタッチすることができます。
2. 以前に CloudFront ディストリビューションにドメインをデプロイしていた場合は、削除する必要があります。

    a. 左のナビゲーションペインで、[ディストリビューション]を選択します。

    b. 編集するディストリビューションの名前を選択します。

    c. [General] (全般) タブを選択します。設定 セクションで、編集 を選択します。

d. 代替ドメイン名 (CNAME) からドメイン名を削除します。次に、[変更を保存する]を選択します。

3. 現在 AWS アカウント または他の でこのドメインを使用している他の CloudFront ディストリビューションが存在しないことを確認します AWS アカウント。現在実行中のサービスを中断しない場合は、ホストゾーンを削除して再作成してみてください。
4. このドメインが、所有する別の Amplify アプリに接続されているかどうか確認します。接続されている場合は、ホスト名のいずれかを再利用していないことを確認します。別のアプリに `www.example.com` を使用している場合、現在接続しているアプリで `www.example.com` を使用することはできません。`blog.example.com` などの他のサブドメインを使用できます。
5. このドメインが別のアプリに正常に接続され、過去 1 時間以内に削除された場合は、1 時間以上経過してからもう一度試してください。6 時間後にこの例外が表示された場合は、 にお問い合わせください サポート。詳細については、「AWS サポート ユーザーガイド」の「サポートケースの作成」を参照してください。
6. Route 53 を介してドメインを管理する場合は、古い CloudFront ディストリビューションを指すホストゾーン CNAME または ALIAS レコードを必ずクリーンアップしてください。
7. 前のステップを完了したら、Amplify ホスティングからカスタムドメインを削除し、ワークフローから始めて、Amplify コンソールでカスタムドメインを接続します。

## 「追加の検証が必要です」というエラーが表示されます。

追加検証必須エラーが発生した場合、 AWS Certificate Manager (ACM) はこの証明書リクエストを処理するために追加情報を必要とすることを意味します。この状況は不正保護対策として生じることがあります。たとえば、ドメインが「Alexa の上位 1,000 のウェブサイト」内にランク付けされている場合です。要求された情報を提供するには、サポートセンターから サポートにお問い合わせください。サポートプランを契約していない場合は、ACM ディスカッションフォーラム に新しいスレッドを投稿してください。

> **Note**
>
> 末尾が amazonaws.com、cloudfront.net、または elasticbeanstalk.com などの Amazon が所有するドメイン名に証明書をリクエストすることはできません。

## CloudFront URL に 404 エラーが出る

トラフィックを処理するために、Amplify ホスティングは CNAME レコードを介して CloudFront URL を指します。アプリをカスタムドメインに接続する過程で、Amplify コンソールにはアプリの CloudFront URL が表示されます。ただし、この CloudFront URL を使用してアプリケーションに直接アクセスすることはできません。404 エラーが返される。アプリケーションは、Amplify アプリの URL (例: `https://main.d5udybEXAMPLE.amplifyapp.com`) またはカスタムドメイン (例: `www.example.com`) を使用してのみ解決されます。

Amplify は、デプロイされた正しいブランチにリクエストをルーティングする必要があり、ホスト名を使用してこれを行います。たとえば、アプリのメインラインブランチを指すドメイン `www.example.com` を設定できるだけでなく、同じアプリの dev ブランチを指すドメイン `dev.example.com` も設定できます。そのため、Amplify がそれに応じてリクエストをルーティングできるように、設定されているサブドメインに基づいてアプリケーションにアクセスする必要があります。

## 自分のドメインにアクセスしたときに SSL 証明書または HTTPS エラーが発生する。

サードパーティーの DNS プロバイダーで設定された認証機関認可 (CAA) DNS レコードがある場合、 AWS Certificate Manager (ACM) はカスタムドメイン SSL 証明書の中間証明書を更新または再発行できない場合があります。これを解決するには、Amazon の認証局ドメインの少なくとも 1 つを信頼する CAA レコードを追加する必要があります。次の手順では、実行する必要のあるステップについて説明します。

Amazon 認証局を信頼する CAA レコードを追加するには

1. Amazon の認証局ドメインの少なくとも 1 つを信頼するように、ドメインプロバイダーに CAA レコードを設定します。CAA レコードの設定について詳しくは、「AWS Certificate Manager ユーザーガイド」の「Certificate Authority Authorization (CAA) の問題」を参照してください。
2. SSL 証明書を更新するには、次のいずれかの方法を使用します。
   - Amplify コンソールを使用して手動で更新します。

     > **Note**
     >
     > この方法では、カスタムドメインのダウンタイムが発生します。

a. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
b. CAA レコードを追加するアプリを選択します。
c. ナビゲーションペインで、[アプリ設定]、[ドメイン管理] の順に選択します。
d. 「ドメイン管理」ページで、カスタムドメインを削除します。
e. アプリをカスタムドメインに再接続します。このプロセスにより新しい SSL 証明書が発行され、その中間証明書を ACM で管理できるようになりました。

   アプリをカスタムドメインに再接続するには、使用しているドメインプロバイダーに対応する以下のいずれかの手順を使用してください。

   - Amazon Route 53 が管理するカスタムドメインの追加.
   - サードパーティーの DNS プロバイダーが管理するカスタムドメインの追加.
   - GoDaddy が管理するドメインの DNS レコードの更新.

- SSL 証明書を再発行 サポート するには、 にお問い合わせください。

## ドメインリダイレクトでサポートされていないパスコンポーネント

ドメインリダイレクトはホスト名部分のみに一致します。ドメインベースのソースルール ("`https://domain.com/path`" など) のパスコンポーネントはサポートされていないため、ルールはエラーなしで無視されます。詳細については、「サンプルリファレンスをリダイレクトして書き換える」を参照してください。

## クロスアカウントドメインの関連付けで 400 エラーが発生する

同じリージョン内の他の AWS アカウント (複数可) の異なる Amplify App (複数可) に既または以前に関連付けられているドメインを持つ Amplify アプリの DomainAssociation リクエストを開始する場合、これはクロスアカウントドメインの関連付けと見なされます。このエラーが表示される場合は、クロスアカウントドメインの関連付けを試みていることを意味します。これには手動検証が必要です。クロスアカウントドメインの関連付けを続行する場合は、AWS サポートにお問い合わせください。

# サーバーサイドレンダリングされたアプリケーションのトラブルシューティング

Amplify ホスティングコンピューティングで SSR アプリをデプロイする際に予期しない問題が発生した場合は、以下のトラブルシューティングトピックを確認してください。ここで問題の解決策が見つからない場合は、Amplify Hosting GitHub Issues リポジトリのSSR ウェブコンピュートトラブルシューティングガイドを参照してください。

トピック



## フレームワークアダプターの使い方を知りたい

フレームワークアダプターを使用する SSR アプリケーションのデプロイで問題が発生する場合は、「SSR フレームワークのオープンソースアダプターの使用」を参照してください。

## Edge API ルートが原因で Next.js ビルドが失敗する

現在、Amplify は Next.js エッジ API ルートをサポートしていません。Amplify でアプリをホストするときは、エッジ以外の API とミドルウェアを使用する必要があります。

## オンデマンドの Incremental Static Regeneration がアプリで機能しない

バージョン 12.2.0 以降、Next.js は特定のページの Next.js キャッシュを手動で削除するインクリメンタル・スタティック・リジェネレーション (ISR) をサポートしています。ただし、Amplify は現在オンデマンド ISR をサポートしていません。アプリが Next.js のオンデマンド再検証を使用している場合、アプリを Amplify にデプロイしてもこの特徴量は機能しません。

# アプリケーションのビルド出力が最大許容サイズを超えています

現在、Amplify が SSR アプリでサポートしているビルドの最大出力サイズは 220 MB です。アプリのビルド出力のサイズが最大許容サイズを超えていることを示すエラーメッセージが表示された場合は、削減する手順を実行する必要があります。

アプリのビルド出力のサイズを減らすには、アプリのビルドアーティファクトを検査し、更新または削除すべき大きな依存関係を特定します。まず、ビルドアーティファクトをローカルコンピュータにダウンロードします。次に、ディレクトリのサイズを確認します。例えば、node_modules ディレクトリには、Next.js サーバーのランタイムファイルによって参照される @swc や @esbuild などのバイナリがある場合があります。これらのバイナリは、ランタイムに必須ではないため、ビルドの後に 削除できます。

以下の手順に従って、アプリのビルド出力をダウンロードし、 AWS Command Line Interface (CLI) を使用してディレクトリのサイズを検査します。

Next.js アプリのビルド出力をダウンロードして検査するには

1. ターミナルウィンドウを開いて、次のコマンドを実行します。アプリ ID、ブランチ名、ジョブ ID をユーザー独自の情報に変更します。ジョブ ID には、調査中の失敗したビルドのビルド番号を使用します。

```
aws amplify get-job --app-id abcd1234 --branch-name main --job-id 2
```

2. ターミナル出力で、job、steps、stepName: "BUILD" セクションで署名付きアーティファクト URL を見つけます。URL は、次の出力例では赤で強調表示されます。

```
"job": {
    "summary": {
        "jobArn": "arn:aws:amplify:us-west-2:111122223333:apps/abcd1234/main/
jobs/0000000002",
        "jobId": "2",
        "commitId": "HEAD",
        "commitTime": "2024-02-08T21:54:42.398000+00:00",
        "startTime": "2024-02-08T21:54:42.674000+00:00",
        "status": "SUCCEED",
        "endTime": "2024-02-08T22:03:58.071000+00:00"
    },
    "steps": [
        {
            "stepName": "BUILD",
```

```
            "startTime": "2024-02-08T21:54:42.693000+00:00",
            "status": "SUCCEED",
            "endTime": "2024-02-08T22:03:30.897000+00:00",
            "logUrl": "https://aws-amplify-prod-us-west-2-artifacts.s3.us-
west-2.amazonaws.com/abcd1234/main/0000000002/BUILD/log.txt?X-Amz-Security-
Token=IQoJb3JpZ2luX2V...Example
```

3. URL をコピーしてブラウザウィンドウに貼り付けます。artifacts.zip ファイルはローカルコンピュータにダウンロードされます。これがユーザーのビルド出力です。
4. du ディスク使用量コマンドを実行して、ディレクトリのサイズを検査します。次のコマンド例では、compute および static のディレクトリのサイズを返します。

```
du -csh compute static
```

compute および static のディレクトリのサイズ情報を含む出力の例を次に示します。

```
 29M    compute
3.8M    static
 33M    total
```

5. compute ディレクトリを開き、node_modules フォルダを見つけます。ファイルの依存関係を確認し、フォルダのサイズを小さくするために更新または削除します。
6. アプリにランタイムに必要のないバイナリがある場合は、アプリの amplify.yml ファイルのビルドセクションに次のコマンドを追加して、ビルド後にそれらを削除します。

```
- rm -f node_modules/@swc/core-linux-x64-gnu/swc.linux-x64-gnu.node
- rm -f node_modules/@swc/core-linux-x64-musl/swc.linux-x64-musl.node
```

次の例は、これらのコマンドを本番用ビルドの実行の 後に 追加した amplify.yml ファイルのビルドコマンドセクションを示します。

```
frontend:
  phases:
    build:
      commands:
         -npm run build

         // After running a production build, delete the files
         - rm -f node_modules/@swc/core-linux-x64-gnu/swc.linux-x64-gnu.node
```

```
        - rm -f node_modules/@swc/core-linux-x64-musl/swc.linux-x64-musl.node
```

## ビルドがメモリ不足エラーで失敗します

Next.js では、ビルドアーティファクトをキャッシュして、以降のビルドのパフォーマンスを向上させることができます。さらに、Amplify の AWS CodeBuild コンテナは、後続のビルドパフォーマンスを向上させるために、ユーザーに代わってこのキャッシュを圧縮して Amazon S3 にアップロードします。これにより、ビルドがメモリ不足エラーで失敗する可能性があります。

ビルドフェーズ中にアプリがメモリ制限を超えないようにするには、次のアクションを実行します。まず、ビルド設定の cache.paths セクションから`.next/cache/**/*`を削除します。次に、ビルド設定ファイルから環境変数`NODE_OPTIONS`を削除します。代わりに、Amplify コンソールで環境変数`NODE_OPTIONS`を設定して、ノードの最大メモリ制限を定義します。Amplify コンソールを使用した環境変数を設定する方法の詳細については、「環境変数の設定」を参照してください。

これらの変更を加えたら、ビルドをやり直してください。成功したら、ビルド設定ファイルの cache.paths セクションに`.next/cache/**/*`を追加し直してください。

ビルドパフォーマンスを向上させるための Next.js キャッシュ設定の詳細については、Next.js のウェブサイトの「AWS CodeBuild」を参照してください。

## アプリケーションの HTTP レスポンスサイズが大きすぎる

現在、Web Compute プラットフォームを使用する Next.js 12 以降のアプリで、Amplify がサポートしている最大応答サイズは 5.72 MB です。この制限を超える応答は、コンテンツなしで504 個のエラーをクライアントに返します。

## コンピューティングアプリの起動時間をローカルで測定するにはどうすればよいですか?

次の手順を使用して、Next.js 12 以降の Compute アプリのローカル初期化/起動時間を決定します。アプリのパフォーマンスをローカルと Amplify ホスティングで比較し、その結果を使用してアプリのパフォーマンスを向上させることができます。

Next.js コンピューティングアプリケーションの初期化時間をローカルで測定するには

1. アプリの `next.config.js` ファイルを開き、次のように `output`オプションを `standalone` に設定します。

```
** @type {import('next').NextConfig} */
const nextConfig = {
  // Other options
  output: "standalone",
};

module.exports = nextConfig;
```

2. ターミナルウィンドウを開いて、次のコマンドを実行し、アプリをビルドします。

```
next build
```

3. 次のコマンドを実行し、.next/static フォルダを .next/standalone/.next/static にコピーします。

```
cp -r .next/static .next/standalone/.next/static
```

4. 次のコマンドを実行し、public フォルダを .next/standalone/public にコピーします。

```
cp -r public .next/standalone/public
```

5. 次のコマンドを実行して、Next.js サーバーを起動します。

```
node .next/standalone/server.js
```

6. ステップ 5 でコマンドを実行してからサーバーが起動するまでにかかる時間に注意してください。サーバーがポートでリッスンしている場合、次のメッセージが出力されます。

```
Listening on port 3000
```

7. ステップ 6 でサーバーを起動した後、他のモジュールがロードされるまでにかかる時間に注意してください。たとえば、bugsnag のようなライブラリのロードには 10 ~ 12 秒かかります。ロードされると、確認メッセージ [bugsnag] loaded が表示されます。
8. ステップ 6 とステップ 7 の所要時間を一緒に追加します。この結果は、Compute アプリケーションのローカル初期化/起動時間です。

## ビルドが非推奨の Node.js バージョンエラーで失敗する

問題: SSR アプリケーションビルドが Node.js バージョンがサポートされていませんというエラーで失敗します。

```
# NODE.JS VERSION NOT SUPPORTED
================================================================================
Your application uses Node.js v18.x.x, which is no longer supported.
AWS Amplify Console has ended support for Node.js 14, Node.js 16 and Node.js 18.

To deploy your application, please upgrade to Node.js 20 or later.

For detailed migration guidelines, visit: https://docs.aws.amazon.com/amplify/latest/
userguide/troubleshooting-general.html#update-node-version
================================================================================
```

原因: SSR アプリケーションは、非推奨の Node.js バージョン (14.x、16.x、または 18.x) を使用してビルドされました。2025 年 9 月 15 日以降、Amplify はビルドプロセス中にこれらの非推奨バージョンを使用する SSR アプリケーションのデプロイをブロックします。

Node.js 20 以降を使用するようにビルド環境を更新してください。詳細な手順については、「アプリケーションの Node.js バージョンを更新する必要がある」を参照してください。

# リダイレクトと書き換えのトラブルシューティング

Amplify アプリケーションのリダイレクトと書き換えの設定時に問題が発生した場合は、このセクションのトピックを参照してください。

トピック

- SPA リダイレクトルールを使用しても、特定のルートへのアクセスは拒否されます。
- API へのリバースプロキシをセットアップしたい

## SPA リダイレクトルールを使用しても、特定のルートへのアクセスは拒否されます。

SPA リダイレクトルールを持つ特定のルートでアクセス拒否エラーが発生した場合、アプリのビルド設定で `baseDirectory` が正しく設定されていない可能性があります。たとえば、アプリケー

ションのフロントエンドが build ディレクトリにビルドされている場合、ビルド設定も build ディレクトリを指す必要があります。次のビルド仕様の例は、この設定を示しています。

```
frontend:
  artifacts:
    baseDirectory: build
    files:
        - "**/*"
```

Amplify アプリのビルド仕様設定の完全な例については、「ビルド仕様 YAML 構文のリファレンス」を参照してください。

## API へのリバースプロキシをセットアップしたい

次の JSON を使用して、動的エンドポイントへのリバースプロキシを設定できます。

```
[
  {
    "source": "/documents/<*>",
    "target": "https://otherdomain/resource/<*>",
    "status": "200",
    "condition": null
  }
]
```

Amplify アプリからサードパーティー API へのリバースプロキシを作成する基本的な例については、「リバースプロキシの書き換え」を参照してください。

# キャッシュのトラブルシューティング

Amplify アプリケーションのキャッシュに問題が発生した場合は、このセクションのトピックを参照してください。

トピック

- アプリのキャッシュのサイズを縮小したい
- アプリケーションのキャッシュからの読み取りを無効にしたい

### アプリのキャッシュのサイズを縮小したい

キャッシュを使用している場合は、ビルド間でクリーンアップされていない中間ファイルをキャッシュしている可能性があります。これらの使用頻度の低いファイルをキャッシュすると、キャッシュのサイズが大きくなります。これを防ぐために、アプリケーションのビルド仕様の cache セクションの！ディレクティブを使用して、特定のフォルダがキャッシュされないようにすることができます。

次のビルド設定の例は、！ディレクティブを使用して、キャッシュしないフォルダを指定する方法を示しています。

```
cache:
  paths:
    - node_modules/**/*
    - "!node_modules/path/not/to/cache"
```

node_modules フォルダをキャッシュすると、node_modules/.cache はデフォルトで省略されます。

Amplify アプリのビルド仕様設定の完全な例については、「ビルド仕様 YAML 構文のリファレンス」を参照してください。

### アプリケーションのキャッシュからの読み取りを無効にしたい

アプリケーションのキャッシュからの読み取りを無効にする場合は、アプリケーションのビルド仕様からキャッシュセクションを削除します。

## GitHub リポジトリへの Amplify アクセスの設定

Amplify は GitHub アプリの機能を使用して、Amplify に GitHub リポジトリへの読み取り専用アクセスを許可するようになりました。Amplify GitHub アプリでは、権限がより細かく調整され、指定したリポジトリにのみ Amplify にアクセス権を付与できます。GitHub アプリの詳細については、GitHub ウェブサイトの「GitHub アプリについて」を参照してください。

GitHub リポジトリに保存されている新しいアプリに接続すると、デフォルトでは Amplify は GitHub アプリを使用してリポジトリにアクセスします。ただし、以前に GitHub リポジトリから接続した既存の Amplify アプリは、アクセスに OAuth を使用します。CI/CD はこれらのアプリでも引き続き機能しますが、新しい Amplify GitHub アプリを使用するように移行することを強くお勧めします。

Amplify コンソールを使用して新しいアプリをデプロイしたり、既存のアプリを移行したりすると、Amplify GitHub アプリのインストール場所に自動的に誘導されます。アプリのインストールランディングページに手動でアクセスするには、ウェブブラウザを開いて地域別にアプリケーションに移動します。`https://github.com/apps/aws-amplify-`*`REGION`* 形式を使用し、*`REGION`* を Amplify アプリをデプロイするリージョンに置き換えてください。例えば、Amplify GitHub アプリを米国西部 (オレゴン) リージョンにインストールするには、https://github.com/apps/aws-amplify-us-west-2 に移動します。

トピック

- 新規デプロイ用の Amplify Github App のインストールと承認
- 既存の OAuth アプリを Amplify GitHub アプリに移行する
- 、CLI CloudFormation、および SDK デプロイ用の Amplify GitHub アプリのセットアップ
- Amplify Github アプリを使ったウェブプレビューの設定

## 新規デプロイ用の Amplify Github App のインストールと承認

GitHub リポジトリ内の既存のコードから新しいアプリを Amplify にデプロイするときは、以下の手順に従って GitHub アプリをインストールして認可します。

Amplify Github アプリをインストールして認可するには

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
2. 「すべてのアプリ」ページから [新規アプリ]、[ウェブアプリをホスト] の順に選択します。
3. 「Amplify ホスティングを始める」ページで、[GitHub] を選択し、[続行] を選択します。
4. GitHub リポジトリに初めて接続する場合、ブラウザの GitHub.com に新しいページが開き、GitHub アカウントでの AWS Amplify の認可を求められます。[承認] を選択します。
5. 次に、Amplify GitHub アプリを GitHub アカウントにインストールする必要があります。GitHub.com で、GitHub アカウントへの AWS Amplify のインストールと認可の許可を求めるページが開きます。
6. Amplify GitHub アプリをインストールする GitHub アカウントを選択します。
7. 次のいずれかを行います。
   - インストールをすべてのリポジトリに適用するには、「全てのリポジトリ」を選択します。
   - 選択した特定のリポジトリのみにインストールを制限するには、[選択したリポジトリのみ] を選択します。選択したリポジトリには、移行するアプリのリポジトリを必ず含めてください。

8. [インストールして承認] を選択します。
9. Amplify コンソールのアプリの [リポジトリブランチを追加] ページにリダイレクトされます。
10. 「最近更新されたリポジトリ」リストで、接続するリポジトリの名前を選択します。
11. ブランチリストで、接続するリポジトリブランチの名前を選択します。
12. [次へ] を選択します。
13. [ビルド設定の構成]ページで、[次へ]を選択します。
14. [レビュー]ページ で、[保存してデプロイ]を選択します。

## 既存の OAuth アプリを Amplify GitHub アプリに移行する

以前 GitHub リポジトリから接続した既存の Amplify アプリは、リポジトリアクセスに OAuth を使用します。GitHub アプリを使用するには、これらのアプリを移行することを強くお勧めします。

以下の手順に従ってアプリを移行し、GitHub アカウント内の対応する OAuth Webhook を削除します。移行の手順は、Amplify GitHub アプリが既にインストールされているかどうかによって異なることに注意してください。最初のアプリを移行し、GitHub アプリをインストールして認可したら、後続のアプリケーション移行のためにリポジトリ権限を更新するだけで済みます。

アプリを OAuth から GitHub アプリに移行するには

1. にサインイン AWS マネジメントコンソール し、Amplify コンソールを開きます。
2. 移行するアプリを選択します。
3. アプリの情報ページで、青い「GitHub アプリに移行」メッセージを見つけて、[移行を開始] を選択します。
4. [GitHub アプリのインストールと承認] ページで、[GitHub アプリの構成] を選択します。
5. ブラウザに GitHub.com の新しいページを開くと、GitHub アカウントで AWS Amplify を認可する許可を求められます。[承認] を選択します。
6. Amplify GitHub アプリをインストールする GitHub アカウントを選択します。
7. 次のいずれかを行います。
   - インストールをすべてのリポジトリに適用するには、「全てのリポジトリ」を選択します。
   - 選択した特定のリポジトリのみにインストールを制限するには、[選択したリポジトリのみ] を選択します。移行するアプリのリポジトリを、選択したリポジトリに必ず含めてください。
8. [インストールして承認] を選択します。

9. Amplify コンソールのアプリの [GitHub アプリのインストールと認可] ページにリダイレクトされます。GitHub の認可が成功すると、成功メッセージが表示されます。[次へ]をクリックします。
10. 「インストールの完了」ページで [インストール完了] を選択します。このステップにより、既存のウェブフックが削除され、新しい webhook が作成され、移行が完了します。

## 、CLI CloudFormation、および SDK デプロイ用の Amplify GitHub アプリのセットアップ

以前 GitHub リポジトリから接続した既存の Amplify アプリは、リポジトリアクセスに OAuth を使用します。これには、Amplify コマンドラインインターフェイス (CLI) CloudFormation、または SDKs を使用してデプロイしたアプリケーションが含まれます。ただし、GitHub アプリを使用するにはれらのアプリを移行することを強くお勧めします。移行は、 AWS マネジメントコンソールの Amplify コンソールで実行する必要があります。手順については、「既存の OAuth アプリを Amplify GitHub アプリに移行する」を参照してください。

CloudFormation、Amplify CLI、および SDKs を使用して、リポジトリアクセスに GitHub アプリを使用する新しい Amplify アプリをデプロイできます。このプロセスでは、まず Amplify Github アプリを GitHub アカウントにインストールする必要があります。次に、GitHub アカウントで個人アクセストークンを生成する必要があります。最後に、アプリをデプロイし、個人アクセストークンを指定します。

Amplify GitHub App をアカウントにインストールします

1. ウェブブラウザを開き、アプリをデプロイする AWS リージョンの Amplify GitHub アプリのインストール場所に移動します。

   `https://github.com/apps/aws-amplify-`*`REGION`*`/installations/new` 形式を使用し、「*`REGION`*」を独自の入力に置き換えてください。たとえば、米国西部 (オレゴン) リージョンにアプリをインストールする場合は、`https://github.com/apps/aws-amplify-us-west-2/installations/new` を指定します。
2. Amplify GitHub アプリをインストールする GitHub アカウントを選択します。
3. 次のいずれかを行います。
   - インストールをすべてのリポジトリに適用するには、「全てのリポジトリ」を選択します。

- 選択した特定のリポジトリのみにインストールを制限するには、[選択したリポジトリのみ] を選択します。選択したリポジトリには、移行するアプリのリポジトリを必ず含めてください。

4. [インストール] を選択します。

GitHub アカウントで個人アクセストークンを生成する

1. GitHub アカウントにサインインします。
2. 右上隅にあるプロフィール写真を探し、メニューから [設定] を選択します。
3. 左側のナビゲーションメニューから、[デベロッパー設定] を選択します。
4. 「GitHub アプリ」ページの左側のナビゲーションメニューで、[個人アクセストークン] を選択します。
5. 「個人アクセストークン」ページで、[新規トークンを生成] を選択します。
6. 「新規個人アクセストークン」ページの「メモ」に、トークンのわかりやすい名前を入力します。
7. 「スコープの選択」セクションで、「admin: repo_hook」を選択します。
8. [Generate token] を選択します。
9. 個人アクセストークンをコピーして保存します。CLI、または SDKs を使用して Amplify アプリをデプロイするときに CloudFormation、これを提供する必要があります。

Amplify GitHub アプリが GitHub アカウントにインストールされ、個人用アクセストークンを生成したら、Amplify CLI CloudFormationまたは SDKs を使用して新しいアプリをデプロイできます。`accessToken` フィールドを使用して、前の手順で作成した個人アクセストークンを指定します。詳細については、Amplify API リファレンスの「CreateApp」と、ユーザーガイドAWS CloudFormation の「AWS:: Amplify:: App」を参照してください。

次の CLI コマンドは、リポジトリへのアクセスに GitHub アプリを使用する新しい Amplify アプリをデプロイします。*myapp-using-githubapp*、*https://github.com/Myaccount/react-app*、および *MY_TOKEN* を自分の情報に置き換えてください。

```
aws amplify create-app --name myapp-using-githubapp --repository https://github.com/
Myaccount/react-app --access-token MY_TOKEN
```

# Amplify Github アプリを使ったウェブプレビューの設定

ウェブプレビューは、GitHub リポジトリに対して行われたすべてのプルリクエスト (PR) を固有のプレビュー URL にデプロイします。プレビューでは、Amplify Github アプリを使用して GitHub リポジトリにアクセスできるようになりました。ウェブプレビュー用の GitHub アプリのインストールと承認の手順については、プルリクエストの Web プレビューを有効にする を参照してください。

# AWS Amplify ホスティングリファレンス

このセクションのトピックを使用して、詳細な参考資料を見つけます AWS Amplify。

トピック



## AWS CloudFormation サポート

AWS CloudFormation テンプレートを使用して Amplify リソースをプロビジョニングし、繰り返し可能で信頼性の高いウェブアプリケーションのデプロイを可能にします。 は、クラウド環境内のすべてのインフラストラクチャリソースを記述してプロビジョニングするための共通言語 AWS CloudFormation を提供し、数回のクリックで複数の AWS アカウントやリージョンへのロールアウトを簡素化します。

Amplify ホスティングについては、Amplify CloudFormation ドキュメントを参照してください。Amplify Studio については、Amplify UI Builder CloudFormation ドキュメントを参照してください。

## AWS Command Line Interface サポート

を使用して AWS Command Line Interface 、コマンドラインからプログラムで Amplify アプリを作成します。詳細については、AWS CLI ドキュメントを参照してください。

## リソースタグ付けのサポート

を使用して Amplify リソース AWS Command Line Interface にタグを付けることができます。詳細については、AWS CLI タグリソースに関するドキュメントを参照してください。

## Amplify ホスティング API

このリファレンスには、Amplify ホスティング API のアクションとデータ型の説明があります。詳細については、Amplify API リファレンスドキュメントを参照してください。

# のドキュメント履歴 AWS Amplify

次の表は、 の前回のリリース以降のドキュメントの重要な変更点を示しています AWS Amplify。

- 前回のドキュメント更新日: 2025 年 9 月 8 日

| 変更 | 説明 | 日付 |
| --- | --- | --- |
| Node.js でサポートされているバージョンの更新 | 「Amplify ホスティングでサーバーサイドレンダリングされたアプリのデプロイ」の Node.js でサポートされるバージョンに関する情報を更新しました。 | 2025 年 9 月 8 日 |
| ビルド設定と構成の章の更新 | アプリケーションに必要な CPU、メモリ、ディスク容量リソースを提供するインスタンスタイプを選択できる、設定可能な新しいビルドインスタンスタイプ機能について説明する「Amplify アプリケーションのビルド設定の管理」の章を更新しました。 | 2025 年 5 月 28 日 |
| Firewall の章の更新 | Amplify ホストサイトのファイアウォールサポート 章を更新して、GA 機能や料金構造など AWS WAF、Amplify と の統合の一般提供 (GA) について説明しました。 | 2025 年 3 月 26 日 |
| 新しいスキュー保護の章 | Amplify ウェブアプリケーションのクライアントとサーバー間のバージョンスキューの問題を排除するスキュー保護機 | 2025 年 3 月 10 日 |

| 変更 | 説明 | 日付 |
| --- | --- | --- |
| | 能を説明する「Amplify デプロイのスキュー保護」の章を追加しました。 | |
| Webhooks の章の更新 | 単一の Git リポジトリに関連付けられているすべての Amplify アプリケーションに 1 つの包括的なウェブフックを使用する Amplify の新しい統合ウェブフック機能について説明する「Git リポジトリの統合ウェブフック」の章を追加しました。 | 2025 年 3 月 10 日 |
| 新規 SSR コンピューティングロールを追加して AWS リソースへのアクセスを許可する トピック | Amplify SSR コンピューティングロールを作成してアプリに関連付け、Amplify コンピューティングサービスに AWS リソースへのアクセスを許可する方法を説明する「AWS リソースへのアクセスを許可する SSR コンピューティングロールの追加」のトピックを追加しました。 | 2025 年 2 月 17 日 |
| Amplify アプリを保護するために AWS WAF を使用する新しい章 | ウェブアクセスコントロールリスト AWS WAF (ウェブ ACL) を使用してウェブアプリケーションを保護できる Amplify と (プレビューで) の統合について説明するAmplify ホストサイトのファイアウォールサポート章を追加しました。 | 2024 年 12 月 18 日 |

| 変更 | 説明 | 日付 |
| --- | --- | --- |
| 更新されたマネージドポリシーのトピック | Amplify の管理ポリシーの最近の AWS への変更について説明するようにAWS の マネージドポリシー AWS Amplifyトピックを更新しました。 | 2024 年 11 月 14 日 |
| Next.js に対する Amplifyの サポートに関するトピックの更新 | Next.js バージョン 15 に対する Amplify のサポートについて説明する「Next.js 向けの Amplify サポート」のトピックを更新しました。 | 2024 年 11 月 6 日 |
| Amazon S3 の章からの Amplify への静的ウェブサイトのデプロイ | Amplify と Amazon S3 の新しい統合について説明する Amazon S3 バケットから Amplify への静的ウェブサイトのデプロイ の章を追加しました。この章では、S3 に保存されている静的ウェブサイトコンテンツを数回のクリックでホストできます。 | 2024 年 10 月 16 日 |
| 新しいキャッシュ設定の管理の章 | Amplify のデフォルトのキャッシュ動作と、コンテンツにマネージドキャッシュポリシーを適用する方法について説明する アプリケーションのキャッシュ設定の管理 の章を追加しました。 | 2024 年 8 月 13 日 |
| 更新されたマネージドポリシーのトピック | Amplify の管理ポリシーの最近の AWS への変更について説明するようにAWS の マネージドポリシー AWS Amplifyトピックを更新しました。 | 2024 年 7 月 18 日 |

| 変更 | 説明 | 日付 |
| --- | --- | --- |
| 更新されたマネージドポリシーのトピック | Amplify の管理ポリシーの最近の AWS への変更について説明するようにAWS の マネージドポリシー AWS Amplifyトピックを更新しました。 | 2024 年 5 月 31 日 |
| 更新されたマネージドポリシーのトピック | Amplify の管理ポリシーの最近の AWS への変更について説明するようにAWS の マネージドポリシー AWS Amplifyトピックを更新しました。 | 2024 年 4 月 17 日 |
| 改訂済みの概要の章 | チュートリアルで Next.js サンプルアプリケーションを使用する Amplify ホスティングへのアプリのデプロイの概要 の章を更新しました。 | 2024 年 4 月 12 日 |
| 更新されたマネージドポリシーのトピック | Amplify の管理ポリシーの最近の AWS への変更について説明するようにAWS の マネージドポリシー AWS Amplifyトピックを更新しました。 | 2024 年 4 月 5 日 |
| 更新されたマネージドポリシーのトピック | Amplify の管理ポリシーの最近の AWS への変更について説明するようにAWS の マネージドポリシー AWS Amplifyトピックを更新しました。 | 2024 年 4 月 4 日 |
| 新しいトラブルシューティングの章 | Amplify ホスティングにデプロイされたアプリケーションで発生する問題を修正する方法を説明する Amplify ホスティングのトラブルシューティング の章を追加しました。 | 2024 年 4 月 2 日 |

| 変更 | 説明 | 日付 |
| --- | --- | --- |
| カスタム SSL/TLS 証明書の新しいサポート | アプリをカスタムドメインに接続する際のカスタム SSL/TLS 証明書の Amplify サポートを説明する SSL/TLS 証明書の使用 トピックを カスタムドメインの接続 の章に追加しました。 | 2024 年 2 月 20 日 |
| 更新されたマネージドポリシーのトピック | Amplify の管理ポリシーの最近の AWS への変更について説明するようにAWS の マネージドポリシー AWS Amplifyトピックを更新しました。 | 2024 年 1 月 2 日 |
| SSR フレームワーク向けの新しいサポート | オープンソースのアダプターを使用した Javascript ベースの SSR フレームワーク向けの Amplify のサポートについて説明する Amplify ホスティングでサーバーサイドレンダリングされたアプリのデプロイ トピックを更新しました。 | 2023 年 11 月 19 日 |
| 画像の最適化機能に関する新たなサポートの提供開始 | サーバーサイドレンダリングされるアプリケーション向けの画像の最適化の組み込みサポートについて説明する SSR アプリケーション向けの画像の最適化 トピックを追加しました。 | 2023 年 11 月 19 日 |

| 変更 | 説明 | 日付 |
| --- | --- | --- |
| 更新されたマネージドポリシーのトピック | Amplify の管理ポリシーの最近の AWS への変更について説明するようにAWS の マネージドポリシー AWS Amplifyトピックを更新しました。 | 2023 年 11 月 17 日 |
| 更新されたマネージドポリシーのトピック | Amplify の管理ポリシーの最近の AWS への変更について説明するようにAWS の マネージドポリシー AWS Amplifyトピックを更新しました。 | 2023 年 11 月 6 日 |
| 新しいワイルドカードサブドメインのトピック | カスタムドメインでのワイルドカードサブドメインのサポートを説明するワイルドカードサブドメインの設定トピックを追加しました。 | 2023 年 11 月 1 日 |
| 新しいマネージドポリシー | AWS の マネージドポリシー AWS Amplify Amplify の新しい AmplifyBackendDeployFullAccess AWS 管理ポリシーを説明するトピックを更新しました。 | 2023 年 10 月 8 日 |
| monorepo フレームワーク機能の提供開始を新たにサポート | Yarn ワークスペース、Nx、Turborepo を使用して作成された monorepo でのアプリのデプロイのサポートについて説明するようにモノレポビルド設定の構成トピックを更新しました。 | 2023 年 6 月 19 日 |

| 変更 | 説明 | 日付 |
| --- | --- | --- |
| 更新されたマネージドポリシーのトピック | Amplify の管理ポリシーの最近の AWS への変更について説明するようにAWS の マネージドポリシー AWS Amplifyトピックを更新しました。 | 2023 年 6 月 1 日 |
| 更新されたマネージドポリシーのトピック | Amplify の管理ポリシーの最近の AWS への変更について説明するようにAWS の マネージドポリシー AWS Amplifyトピックを更新しました。 | 2023 年 2 月 24 日 |
| 更新されたサーバーサイドレンダリングの章 | Next.js バージョン 12 と 13 に対する Amplify のサポートに関する最近の変更点を説明するためにAmplify ホスティングでサーバーサイドレンダリングされたアプリのデプロイこの章を更新しました。 | 2022 年 11 月 17 日 |
| 更新されたマネージドポリシーのトピック | Amplify の管理ポリシーの最近の AWS への変更について説明するようにAWS の マネージドポリシー AWS Amplifyトピックを更新しました。 | 2022 年 8 月 30 日 |
| 更新されたマネージドポリシーのトピック | Amplify Studio アプリケーションのバックエンドの構築 を使用してバックエンドをデプロイする方法を説明するようにトピックを更新しました。 | 2022 年 8 月 23 日 |

| 変更 | 説明 | 日付 |
| --- | --- | --- |
| 更新されたマネージドポリシーのトピック | Amplify の管理ポリシーの最近の AWS への変更について説明するようにAWS の マネージドポリシー AWS Amplifyトピックを更新しました。 | 2022 年 4 月 27 日 |
| 更新されたマネージドポリシーのトピック | Amplify の管理ポリシーの最近の AWS への変更について説明するようにAWS の マネージドポリシー AWS Amplifyトピックを更新しました。 | 2022 年 4 月 17 日 |
| GitHub アプリの新機能のリリース | Amplify による GitHub リポジトリへのアクセスを承認するための新しい GitHub App を説明するGitHub リポジトリへの Amplify アクセスの設定トピックを追加しました。 | 2022 年 4 月 5 日 |
| Amplify Studio の新機能のリリース | バックエンドデータに接続できる UI コンポーネントを作成するためのビジュアルデザイナーを提供する Amplify Studio の更新について説明するAWS Amplify ホスティングへようこそトピックを更新しました。 | 2021 年 12 月 2 日 |
| 更新されたマネージドポリシーのトピック | Amplify Studio をサポートするための Amplify の AWS 管理ポリシーの最近の変更について説明するためにAWS の マネージドポリシー AWS Amplifyトピックを更新しました。 | 2021 年 12 月 2 日 |

| 変更 | 説明 | 日付 |
| --- | --- | --- |
| 更新されたマネージドポリシーのトピック | Amplify の管理ポリシーの最近の AWS への変更について説明するようにAWS の マネージドポリシー AWS Amplifyトピックを更新しました。 | 2021 年 11 月 8 日 |
| 更新されたマネージドポリシーのトピック | Amplify の管理ポリシーの最近の AWS への変更について説明するようにAWS の マネージドポリシー AWS Amplifyトピックを更新しました。 | 2021 年 9 月 27 日 |
| 新しい マネージドポリシーのトピック | Amplify の AWS 管理ポリシーと、それらのポリシーに対する最近の変更を説明するAWS の マネージドポリシー AWS Amplifyトピックを追加しました。 | 2021 年 7 月 28 日 |
| 「サーバーサイドレンダリング」の章を更新しました。 | Next.js バージョン 10.x.x と Next.js バージョン 11 の新しいサポートについて説明する「Amplify ホスティングでサーバーサイドレンダリングされたアプリのデプロイ」の章を更新しました。 | 2021 年 7 月 22 日 |
| 「ビルド設定の構成」の章を更新しました。 | Amplify で monorepo アプリをデプロイする際のビルド設定と新しいAMPLIFY_MONOREPO_APP_ROOT 環境変数の構成方法を説明するモノレポビルド設定の構成トピックを追加しました。 | 2021 年 7 月 20 日 |

| 変更 | 説明 | 日付 |
| --- | --- | --- |
| 「機能ブランチのデプロイ」の章を更新しました。 | ビルド時に`aws-exports.js` ファイルを自動生成する方法を説明するAmplify 設定のビルド時の自動生成 (Gen 1 アプリ専用)トピックを追加しました。条件付きバックエンドビルドを有効化する方法を説明する条件付きバックエンドビルド (Gen 1 アプリ専用)トピックを追加しました。新しいアプリを作成したり、新しいブランチを既存のアプリに接続したり、既存のフロントエンドを更新して別のバックエンド環境を指すようにしたりするときに、既存のバックエンドを再利用する方法を説明するアプリ間で Amplify バックエンドを使用する (Gen 1 アプリ専用)トピックを追加しました。 | 2021 年 6 月 30 日 |
| セキュリティ章が追加されました。 | 責任分担モデルを適用する方法と、Amplify が暗号化を使用して保管中のデータまたは転送中のデータを保護する方法を説明するAmplify のデータ保護トピックを追加しました。 | 2021 年 6 月 3 日 |

| 変更 | 説明 | 日付 |
| --- | --- | --- |
| SSR 機能の提供開始を新たにサポート | サーバーサイドレンダリング (SSR) を使用し、Next.js で作成されたウェブアプリの Amplify サポートについて説明する Amplify ホスティングでサーバーサイドレンダリングされたアプリのデプロイ 章を追加しました。 | 2021 年 5 月 18 日 |
| セキュリティに関する新しい章 | Amplify を使用する際に責任分担モデルを適用する方法と、セキュリティとコンプライアンスの目標を達成するように Amplify を構成する方法を説明する Amplify のセキュリティ 章を追加しました。 | 2021 年 3 月 26 日 |
| カスタムビルドのトピックを更新 | Amazon Elastic Container Registry Public でホストされているカスタムビルドイメージの設定を説明するために、カスタムビルドイメージとライブパッケージの更新 のトピックを更新しました。 | 2021 年 3 月 12 日 |
| モニタリングトピックを更新 | Amazon CloudWatch メトリックスデータにアクセスしてアラームを設定する方法を説明するために、モニタリング のトピックを更新しました。 | 2021 年 2 月 2 日 |

| 変更 | 説明 | 日付 |
| --- | --- | --- |
| CloudTrail のログに関する新トピック | コンソール API AWS CloudTrail AWS Amplify リファレンスと AWS Amplify 管理者 UI API リファレンスのすべての API アクションを が AWS CloudTrail キャプチャしてログに記録する方法について説明します。 | 2021 年 2 月 2 日 |
| 管理 UI の新機能をリリース | フロントエンドのウェブ開発者とモバイル開発者が AWS マネジメントコンソールの外部で、アプリのバックエンドを作成および管理するためのビジュアルインターフェイスを提供する新しい管理 UI について説明するAWS Amplify ホスティングへようこそトピックを更新しました。 | 2020 年 12 月 1 日 |
| パフォーマンスモードの新機能をリリース | 「アプリパフォーマンスの管理」のトピックを更新し、パフォーマンスモードを有効にして、ホスティングパフォーマンスを迅速にし最適化する方法について説明しました。 | 2020 年 11 月 4 日 |
| カスタムヘッダーのトピックを更新 | コンソールを使用して、または YML ファイルを編集して Amplify アプリのカスタムヘッダーを定義する方法を説明するカスタムヘッダーのトピックを更新しました。 | 2020 年 10 月 28 日 |

| 変更 | 説明 | 日付 |
| --- | --- | --- |
| 自動サブドメインの新機能をリリース | Amazon Route 53 カスタムドメインに接続されたアプリにパターンベースの機能ブランチデプロイを使用する方法を説明する「Route 53 カスタムドメインの自動サブドメインの設定」のトピックを追加しました。プルリクエストからのウェブプレビューを、サブドメインでアクセスできるように設定する方法を説明する「サブドメインによるウェブプレビューアクセス」のトピックを追加しました。 | 2020 年 6 月 20 日 |
| 新しい通知に関するトピック | ビルドが成功または失敗したときに利害関係者またはチームメンバーに警告する Amplify アプリのメール通知を設定する方法を説明する通知に関するトピックを追加しました。 | 2023 年 6 月 20 日 |
| カスタムドメインのトピックを更新 | Amazon Route 53、GoDaddy、および Google ドメインにカスタムドメインを追加する手順を改善するためにカスタムドメインの接続トピックを更新しました。この更新には、カスタムドメインの設定に関する新しいトラブルシューティング情報も含まれています。 | 2020 年 5 月 12 日 |
| AWS Amplify リリース | このリリースでは、Amplify が導入されました。 | 2018 年 11 月 26 日 |

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。