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 Hosting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を備えた Astro
- SvelteKit コミュニティアダプターを使用する
- 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
- 角度
- Vue.js
- イオニック
- Ember

静的サイトジェネレーター

- イベント
- Gatsby
- Hugo
- ジキル語

- VuePress

Amplify Hosting の機能

特徴量ブランチ

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。

E nd-to-end テスト

end-to-end テストでアプリケーション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では、デプロイ全体が終了した後にのみウェブアプリケーションが更新されるようにすることで、メンテナンスウィンドウがなくなります。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を開始する

Amplify ホスティングの使用を開始するには、Amplify ホスティングの使用開始「」チュートリアルを参照してください。チュートリアルを完了すると、Git リポジトリ (GitHub、、または AWS CodeCommit) にウェブアプリを接続し BitBucket GitLab、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。

バックエンドを構築する

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のデベロッパー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てアプリに接続する方法については、Amplify ドキュメントの「バックエンドを構築して接続する」を参照してください。

CLI と Amplify Studio を使用して Gen 1 アプリケーションのバックエンドを構築するためのドキュメントをお探しの場合は、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 の一部です AWS 無料利用枠。無料利用枠の制限を超えたら、無料で開始し、従量制料金を支払うことができます。Amplify ホスティング料金の詳細については、AWS Amplify 「 の料金」を参照してください。

Amplify ホスティングの使用開始

Amplify ホスティングの仕組みを理解するために、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。

トピック

前提条件

このチュートリアルを開始する前に、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可能なその他の 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 create-next-appの手順を使用して、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 リポジトリを接続する

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。

Git リポジトリでアプリを接続するには

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する場合は、デフォルトで AWS Amplifyサービスページから開始します。

   ページの上部にある「新しいアプリの作成」を選択します。
3. Amplify で構築を開始するページで、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 アプリ機能を使用して Amplify アクセスを承認します。 GitHub アプリのインストールと承認の詳細については、「」を参照してくださいGitHub リポジトリへの Amplify アクセスの設定。

   > **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] をクリックします。

ステップ 2: ビルド設定を確認する

Amplify は、デプロイす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確認します。

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 に設定されます `.next`。

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。
   - 新しい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] をクリックします。

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、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。

アプリケーションを保存してデプロイするには

1. 確認ページで、リポジトリの詳細とアプリの設定が正しいことを確認します。
2. [保存してデプロイ] を選択します。フロントエンドビルドには通常 1～2 分かかりますが、アプリケーションのサイズによって異なる場合があります。
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: (オプション) リソースをクリーンアップする

チュートリアル用に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、 と入力します**delete**。次に、アプリの削除を選択します。

アプリに機能を追加する

Amplify にアプリがデプロイされたので、ホストされたアプリケーションで利用できる以下の機能の一部を試すことができます。

環境変数

多くの場合、アプリケーションは実行時に設定情報を必要とします。これらの設定は、データベース接続の詳細、API キー、またはパラメータにすることができます。環境変数は、ビルド時にこれらの設定を公開する方法を提供します。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 が 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https://branch-name.d1m7bkiki6tdw1.amplifyapp.com。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、「カスタムドメインのセットアップ」を参照してください。

プルリクエストのプレビュー

ウェブプルリクエストのプレビューでは、コードを本番稼働用ブランチまたは統合ブランチにマージする前に、プルリクエスト (PRs) からの変更をプレビューできます。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。

Amplify ホスティングを使用してサーバー側でレンダリングされたアプリをデプロイします

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 Management Consoleで手動設定を行う必要はありません。また、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。

Amplify が SSR をどのようにサポートするかについては、以下のトピックを確認してください。

トピック

サーバーサイドレンダリングとは

Amplify は、React などのシングルページアプリケーション (SPA) フレームワークを使用して作成された静的ウェブアプリケーションや、Gatsby などの静的サイトジェネレーター (SSG) を使用して作成されたアプリケーションのデプロイとホスティングをサポートしています。静的ウェブアプリケーションは、コンテンツ配信ネットワーク (CDN) に保存されている HTML、CSS、ファイルなどの JavaScript ファイルの組み合わせで構成されます。クライアントブラウザーがウェブサイトにリクエストを送信すると、サーバーは HTTP 応答を含むページをクライアントに返し、クライアントブラウザーはコンテンツを解釈してユーザーに表示します。

また、Amplify は、サーバーサイドレンダリング (SSR) を備えたウェブアプリケーションもサポートしています。クライアントが SSR ページにリクエストを送信すると、リクエストのたびにページの HTML がサーバー上に作成されます。SSR により、開発者はリクエストごとおよびユーザーごとに

ウェブサイトをカスタマイズできます。さらに、SSR はウェブサイトのパフォーマンスと検索エンジン最適化 (SEO) を向上させることができます。

SSR フレームワーク向けの Amplify サポート

Amplify ホスティングは、Amplify が想定するビルド出力に準拠するデプロイバンドルを持つ任意の JavaScript ベースの SSR フレームワークをサポートします。Amplify ホスティングは、SSR フレームワーク向けのアプリケーションのビルド出力のファイルとディレクトリ構造を標準化するデプロイ仕様を提供します。

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。

フレームワークまたはフレームワークアダプター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 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を生成できます。

Amplify ホスティングは、静的アセット、コンピューティング、画像の最適化、ルーティングルールのプリミティブタイプをサポートしています。これらのプリミティブタイプを活用して、より豊富な機能を備えたアプリケーションをデプロイできます。各プリミティブタイプの詳細については、「」を参照してくださいAmplify SSR プリミティブタイプのサポート。

次のシナリオから選択して、SSR アプリケーションの Amplify へのデプロイを開始できます。

Next.js アプリケーションをデプロイする

Amplify は、アダプターやコンソールでの手動設定を必要とすることなく、Next.js を使用して作成されたアプリケーションをサポートしています。詳細については、「Next.js の Amplify サポート」を参照してください。

フレームワークアダプターを使用するアプリケーションをデプロイする

使用可能なオープンソースのフレームワークアダプターを参照して、SSR アプリケーションを Amplify ホスティングにデプロイできます。詳細については、「フレームワークアダプターの使用」を参照してください。

Nuxt フレームワーク用のアダプターが使用可能です。この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。

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と考えている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 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、「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。

ビルド後のスクリプトを設定する

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仕様を使用し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。

SSR アプリケーションを Amplify にデプロイする

このトピックの手順に従って、Amplify が想定するビルド出力に準拠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ケーション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、「フレームワークアダプターの使用」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 Management Console し、Amplify コンソール を開きます。
2. 「すべてのアプリ」ページで、「新しいアプリの作成」を選択します。
3. Amplify で構築を開始するページで、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] をクリックします。
5. アプリ設定ページで、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き、サーバー側のレンダリング (SSR) デプロイセクションで SSR アプリケーション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。

   - 新しい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] をクリックします。
8. [レビュー]ページで、[保存してデプロイ] を選択します。

フレームワークアダプターの使用

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インストールして使用できます。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 アプリケーションを Amplify にデプロイする」を参照してください。

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 のデプロイ仕様を使用して、Amplify ホスティングと統合する SSR フレームワークのビルド出力を設定します。フレームワークの作成者は、デプロイ仕様を参照して、Amplify が想定するビルド出力を構造化する方法を理解できます。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。

Amplify ホスティングのデプロイ仕様

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予想されるディレクトリ構造を生成し、フレームワークが Amplify ホスティングのサービスプリミティブタイプを利用できるようにします。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。

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 プリミティブタイプのサポート

Amplify ホスティングのデプロイ仕様では、以下のプリミティブタイプに密接にマッピングされる契約を定義します。

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。

イメージの最適化

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、静的アセットプリミティブタイプ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タイプ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た際に 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 属性を使用すると、フレームワークは Amplify ホスティングルーティングルールのプリミティブタイプを活用できます。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、「リダイレクトを使用する」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。

キー	タイプ	必須	説明
パス	文字列	あり	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。 最大パス長は 255 文字です。 パスはスラッシュ / で始まる必要があります。 パスには、[A-Z]、[a-z]、[0-9]、[_-.*$/~"' @:+] の文字を使用できます。 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: • * (0 個以上の文字に一致) • /* パターンはキャッチオールパターンと呼ばれ、すべての着信リク

キー	タイプ	必須	説明
			エストに一致します。
target	Target	あり	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。 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。 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。

キー	タイプ	必須	説明
fallback	Target	なし	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。 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。

キー	タイプ	必須	説明
kind	Targetkind	あり	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	文字列	Compute: はい 他のプリミティブ型にはいいえ	プリミティブタイプ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タイプに対してのみ有効で必須です。 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	文字列	なし	応答に適用する Cache-Control ヘッダーの値を指定する文字列。静的型と ImageOptimizationプリミティブ型に対してのみ有効です。

キー	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、「カスタムヘッダー」を参照してください。 **Note** この Cache-Control ヘッダーは、ステータスコードが 200 (OK) に設定されている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

静的アセットプリミティブタイプにリクエストを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タイプにルーティングします。

ImageOptimization

リクエストを画像最適化プリミティブタイプ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 = 'nodejs16.x' | 'nodejs18.x' | 'nodejs20.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。

キー	タイプ	必須	説明
name	文字列	あり	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。 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	あり	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。 有効な値は、nodejs16.x 、nodejs18.x 、nodejs20.x です。
entrypoint	文字列	あり	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

キー	タイプ	必須	説明
			グリソースを表すサ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": "nodejs16.x",
      "entrypoint": "index.js",
    }
  ]
```

imageSettings 属性の使用

imageSettings 属性を使用すると、フレームワークはイメージ最適化プリミティブタイプの動作をカスタマイズでき、実行時にイメージをオンデマンドで最適化できます。

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。

キー	タイプ	必須	説明
sizes	Number[]	あり	サポートされている画像の幅の配列。
domains	String[]	あり	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[]	あり	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[]	あり	許可される出力画像形式の配列。
minimumCacheTTL	数	あり	最適化された画像のキャッシュ期間 (秒)。
dangerouslyAllowSVG	ブール値	あり	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

キー	タイプ	必須	説明
			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?: 'http' | '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。

キー	タイプ	必須	説明
protocol	文字列	なし	許可されるリモートパターンのプロトコル。 有効な値は http または https です。
hostname	文字列	あり	許可されるリモートパターンのホスト名。 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

キー	タイプ	必須	説明
			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	文字列	なし	許可されるリモートパターンのポート。
pathname	文字列	なし	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。

キー	タイプ	必須	説明
name	文字列	あり	フレームワークの名前。
version	文字列	あり	フレームワークのバージョン。 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:

- 静的アセットのプリミティブタイプ – ファイルは .amplify-hosting/static ディレクトリに含まれています。
- コンピューティングプリミティブタイプ — ファイルは .amplify-hosting/compute/default ディレクトリに含まれています。

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。

などの SSR フレームワークの場合Nuxt.js、キャッチオールルールターゲットはコンピューティングプリミティブタイプ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。つまり、/_nuxt/*パスは、リクエストを静的アセットプリミティブタイプ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ターゲットにすることができます。

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ターゲットはコンピューティングプリミティブタイプ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、静的プリミティブタイプから 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タイプにフォールバックす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イプのターゲットにキャッチオールルートNuxt.jsが指定されている などの SSR フレームワークの場合、フレームワークの作成者はpublic、フォルダルーティングの問題を解決するために、キャッチオールルートの静的フォールバックの指定を検討する場合があります。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型にフォールバックし、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": "nodejs18.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": "nodejs18.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": "nodejs18.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。

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しようとしている場合は、「カスタム画像ローダーの使用」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。

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、HTTP または 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	文字列	あり	ソース画像への相対パスまたは絶対 URL。リモート URL の場合、http および https プロトコルがサポートされます。値は URL エンコードされている必要があります。	`?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
width	数	あり	最適化された画像の幅 (ピクセル)。	`?width=800`
height	数	なし	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`	なし	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`	なし	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	数	なし	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	なし	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	なし	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	文字列	なし	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	数	なし	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	数	なし	指定した角度 (度) で画像を回転します。	`?rotate=45`
flip	ブール値	なし	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	なし	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	数	なし	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`
中央値	数	なし	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	数	なし	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	数	なし	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	なし	画像の色を反転します。	`?negate`
normalize	ブール値	なし	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	数	なし	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	文字列	なし	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	なし	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。

キャッシュ

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。

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。

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、「ライブパッケージのアップデート」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 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、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。ここで問題に対する解決策が表示されない場合は、Amplify ホスティング GitHub の問題リポジトリの SSR ウェブコンピューティングの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を使用している
- エッジ API ルートが原因で Next.js ビルドが失敗する
- オンデマンドのインクリメンタル・スタティック・リジェネレーションがアプリでは機能しない
- アプリのビルド出力が最大許容サイズを超えています
- ビルドはメモリ不足エラーで失敗する
- HTTP レスポンスサイズが大きすぎる

フレームワークアダプターを使用している

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 フレームワーク向けの Amplify サポート」を参照してください。

エッジ 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。

オンデマンドのインクリメンタル・スタティック・リジェネレーションがアプリでは機能しない

バージョン 12.2.0 以降、Next.js は増分静的再生 (ISR) をサポートし、特定のページの Next.js キャッシュを手動でクリアします。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 アプリでサポートするビルド出力の最大サイズは 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、それを減らす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小さくするには、アプリのビルドアーティファクトを検査し、更新または削除する大きな依存関係を特定できます。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 サーバーランタイムファイルによって参照`@esbuild`される `@swc`や などのバイナリが含まれている場合があります。これらのバイナリはランタイムで必要ないため、ビルド後に削除できます。

(AWS Command Line Interface CLI) を使用してアプリケーションのビルド出力をダウンロードし、ディレクトリのサイズを検査するには、以下の手順に従います。

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き、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 をユーザー自身の情報に変更します。ジョブ ID には、調査対象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、 stepsstepName: "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。次の出力例では、URL が赤で強調表示されています。

```
"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 ファイルがローカルコンピュータにダウンロードされます。これはビルド出力です。

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。次のコマンド例は、 computeおよび static ディレクトリのサイズを返します。

```
du -csh compute static
```

以下は、 computeおよび static ディレクトリのサイズ情報を含む出力の例です。

```
 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。フォルダのサイズを小さくするために更新または削除できるファイルの依存関係を確認します。
6. アプリケーションにランタイムに必要のないバイナリが含まれている場合は、アプリケーションの 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、ビルド後にバイナリを削除します。

```
- 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
```

以下は、本番ビルドの実行後にこれらのコマンドが追加された amplify.yml ファイルのビルドコマンドセクションの例です。

```
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
```

ビルドはメモリ不足エラーで失敗する

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、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、後続のビルドパフォーマンスを向上させます。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、「環境変数を設定する」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 ウェブサイトの「AWS CodeBuild」を参照してください。

HTTP レスポンスサイズが大きすぎる

現在、ウェブコンピューティングプラットフォームを使用する Next.js 12 以降のアプリで Amplify がサポートする最大レスポンスサイズは 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。

Next.js の Amplify サポート

Amplify は、Next.js を使用して作成されたサーバー側レンダリング (SSR) ウェブアプリケーションのデプロイとホスティングをサポートしています。Next.js は、フルスタックのウェブアプリケーションを構築するための 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 14 で構築されたアプリケーション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、13、および 14 で構築されたアプリケーションのサーバー側のレンダリング (SSR) を完全に管理します。Amplify ホスティングコンピューティングのリリース前に 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クラシック (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
- ミドルウェア
- 環境変数
- イメージの最適化
- 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)
- 国際化 (i18n) 自動ロケール検出
- 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ミドルウェアを実行する

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。

Next.js アプリケーションの料金

Next.js 12 以降の SSR アプリをデプロイする場合、Amplify ホスティングコンピューティングは SSR アプリのデプロイに必要なリソースを自動的に管理します。Amplify ホスティングコンピューティング料金については、AWS Amplify 価格設定をご覧ください。

Amplify を使用した Next.js アプリケーションのデプロイ

デフォルトでは、Amplify は Next.js 12、13、および 14 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 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。

新しい Next.js アプリをデプロイするには、以下の手順に従います。

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 Next.js アプリを Amplify にデプロイするには

1. にサインイン AWS Management Console し、Amplify コンソール を開きます。
2. 「すべてのアプリ」ページで、「新しいアプリの作成」を選択します。
3. Amplify で構築を開始するページで、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 がロールを自動的に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] をクリックします。
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、アプリが SSR か SSG かを検出します。

Next.js SSR アプリのビルドスクリプトの例を示します。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。

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

Next.js SSG アプリのビルドスクリプトの例を示します。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"
},
```

Amplify ビルド設定

package.jsonアプリのファイルを調べて SSG アプリと SSR アプリのどちらをデプロイしているのかを判断した後、Amplify はアプリのビルド設定を確認します。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、「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/**/*
```

Amplify が 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スペックが生成され、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。

以下は、baseDirectoryがout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 ページのみをサポートする 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: out
    files:
      - '**/*'
  cache:
    paths:
      - node_modules/**/*
```

Next.js 11 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 14 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、13、および 14 を完全にサポートして、Amplify ホスティングコンピューティングサービスのリリース前にデプロイされたアカウント内のアプリを検出します。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。

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。アプリケーションには、Classic (Next.js 11 のみ) ブランチと Next.js 12、13、または 14 ブランチの両方を含め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 Management Console 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ケーションを移行する前に、まずアプリケーションの package.json ファイルを更新して Next.js バージョン 12、13、または 14 を使用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、13、または 14 を使用するようにアプリを更新し、SSR 機能を追加します。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によって管理される 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 に設定されますWEB_COMPUTE。

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 Management Console 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、「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 ファイルを更新します。

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。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 DB_HOST -e DB_USER -e DB_PASS >> .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 DB_HOST -e DB_USER -e DB_PASS >> .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。

```
DB_HOST=localhost
DB_USER=myuser
DB_PASS=mypassword
```

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_SEARCH_KEY=asdfiojslf
NEXT_PUBLIC_SEARCH_ENDPOINT=https://search-url
```

monorepos の SSR 環境変数

モノレポに SSR アプリケーションをデプロイしていて、特定の環境変数を Next.js にアクセスできるようにする場合は、 .env.production ファイルにアプリケーションルートのプレフィックスを付ける必要があります。次の Nx monorepo 内の Next.js アプリケーションのビルド仕様の例は、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 DB_HOST -e DB_USER -e DB_PASS >> 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、アプリケーションルート を持つモノレポ内のアプリケーションの `.env.production` ファイルに追加する方法を示していますapps/app。

```
- env | grep -e DB_HOST -e DB_USER -e DB_PASS >> 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。pnpm および Turborepo アプリには追加の設定が必要であることに注意してください。詳細については、「モノレポのビルド設定」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。

Amazon CloudWatch Logs for SSR アプリ

Amplify は Next.js ランタイムに関する情報を の Amazon CloudWatch Logs に送信します AWS アカウント。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 が IAM ロールを作成することを許可することを選択した場合、そのロールには CloudWatch ログを作成するアクセス許可が既に付与されています。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許可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、「サービスロールの追加」を参照してください。

Amplify Next.js 11 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 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 アプリケーションの料金

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 料金の詳細については、「Amazon の CloudFront 料金」を参照してください。
- が CloudFront 配信するコンテンツをカスタマイズするための 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。必要な最小限の権限がないと、SSR アプリをデプロイしようとしたときに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、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 管理ポリシーは、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: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
cloudfront:DeleteDistribution
iam:AttachRolePolicy
iam:CreateRole
```

```
iam:CreateServiceLinkedRole
iam:GetRole
iam:PutRolePolicy
iam:PassRole
iam:UpdateAssumeRolePolicy
iam:DeleteRolePolicy
lambda:CreateFunction
lambda:EnableReplication
lambda:DeleteFunction
lambda:GetFunction
lambda:GetFunctionConfiguration
lambda:PublishVersion
lambda:UpdateFunctionCode
lambda:UpdateFunctionConfiguration
lambda:ListTags
lambda:TagResource
lambda:UntagResource
lambda:ListEventSourceMappings
lambda:CreateEventSourceMapping
route53:ChangeResourceRecordSets
route53:ListHostedZonesByName
route53:ListResourceRecordSets
s3:CreateBucket
s3:GetAccelerateConfiguration
s3:GetObject
s3:ListBucket
s3:PutAccelerateConfiguration
s3:PutBucketPolicy
s3:PutObject
s3:PutBucketTagging
s3:GetBucketTagging
sqs:CreateQueue
sqs:DeleteQueue
sqs:GetQueueAttributes
sqs:SetQueueAttributes
amplify:GetApp
amplify:GetBranch
amplify:UpdateApp
amplify:UpdateBranch
```

Next.js 11 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

出力ディレクトリは上書きされる

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、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/**/*
```

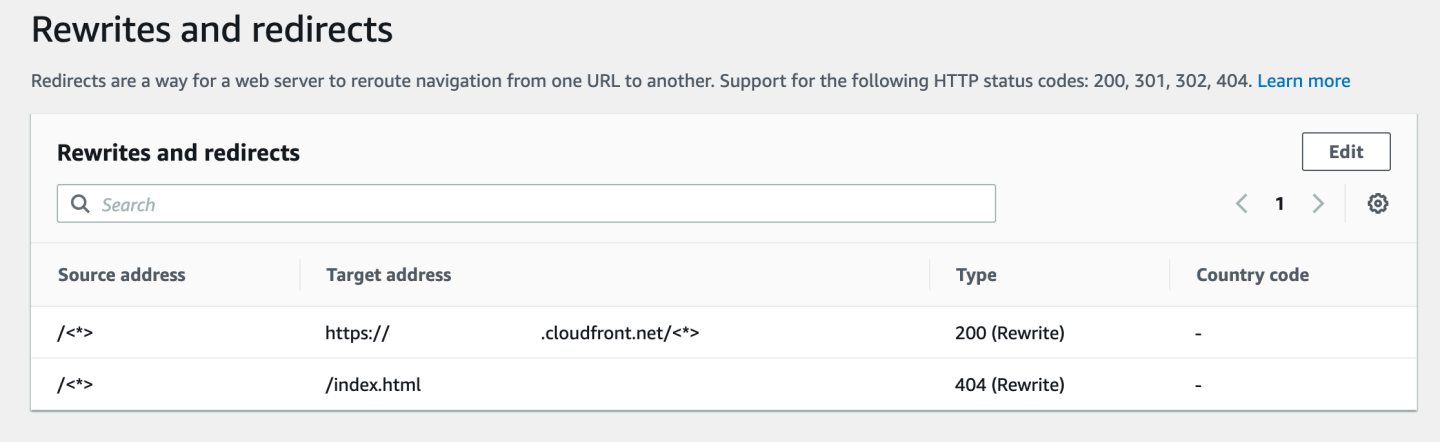
SSR サイトをデプロイすると 404 エラーが発生します。

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、前のトピックの出力ディレクトリは上書きされるの手順に従ってください。

アプリケーションに CloudFront SSR ディストリビューションの書き換えルールがない

SSR アプリケーションをデプロイすると、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書き換えルールが存在することを確認します CloudFront。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、書き CloudFront 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
      },
}
```

ビルドはメモリ不足エラーで失敗する

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、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、後続のビルドパフォーマンスを向上させます。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、「環境変数を設定する」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 Next.js キャッシュ設定の詳細については、Next.js 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は 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。

アプリは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。

アプリが制限に達しました CloudFront

CloudFront サービスクォータは、Lambda@Edge 関数がアタッチされた 25 個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た場合は、アカウントから未使用の CloudFront ディストリビューションを削除するか、クォータの引き上げ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。

環境変数は Lambda 関数に引き継がれません。

SSR アプリの Amplify コンソールで指定した環境変数は、アプリの AWS Lambda 関数に引き継がれません。Lambda 関数から参照できる環境変数を追加する方法の詳細については、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。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 Lambda@Edge 関数を作成して、 が CloudFront 配信するコンテンツをカスタマイズします。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 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。 CloudFront

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 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 アプリの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。

カスタムドメインのセットアップ

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ケーションをデプロイすると、Amplify は などの URL を使用してデフォルトのamplifyapp.comドメインでホストしますhttps://branch-name.d1m7bkiki6tdw1.amplifyapp.com。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。

カスタムドメインは、Amazon Route 53 や などの認定ドメインレジストラを通じて購入できます GoDaddy。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リストについては、ICANN 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、「」を参照してくださいSSL/TLS 証明書の使用。

カスタムドメインの設定に進む前に、次の前提条件を満たしていることを確認します。

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

  このステップを完了する方法の詳細については、「」を参照してくださいAmplify ホスティングの使用開始。
- ドメインと DNS の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。

トピック

- DNS の用語と概念を理解する
- SSL/TLS 証明書の使用
- Amazon Route 53 で管理されているカスタムドメインを追加する
- 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の追加

- によって管理されるドメインの DNS レコードを更新する GoDaddy
- Google ドメインによって管理されるドメインの DNS レコードを更新する
- ドメインの SSL/TLS 証明書を更新する
- サブドメインの管理
- ワイルドカードサブドメイン
- Amazon Route 53 カスタムドメインに自動サブドメインを設定します。
- カスタムドメインのトラブルシューティング

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、 などのサードパーティープロバイダーでドメインをホストする場合は GoDaddy、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。

Amplify ホスティングを利用したカスタムドメインのアクティベーションプロセス

Amplify ホスティングでカスタムドメインを追加する場合、カスタムドメインを使用してアプリを表示できるようになるまでには、いくつかの手順を完了する必要があります。次のリストは、ドメインセットアッププロセスの各ステップを示しています。

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、ユーザー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。Route 53 の外部で管理されるドメインの場合、Amplify コンソールで提供される DNS 検証レコードを、サードパーティーの DNS プロバイダーを使用して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。Route 53 の外部で管理されるドメインの場合、Amplify コンソールで提供される CNAME レコードを、サードパーティーの DNS プロバイダーを使用して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、ウェブブラウザが安全な SSL/TLS プロトコルを使用して、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。

管理証明書を使用すると、Amplify はアプリに接続されているすべてのドメインの SSL/TLS 証明書を発行し、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。

カスタム証明書を使用するには、選択したサードパーティー認証機関から証明書を取得する必要があります。次に、証明書を にインポートします AWS Certificate Manager。ACM は、パブリックおよびプライベートの SSL/TLS 証明書を簡単にプロビジョニング、管理、デプロイして、 AWS のサービス および内部に接続されたリソースで使用できるようにするサービスです。米国東部 (バージニア北部) (us-east-1) リージョンで証明書をリクエストまたはインポートしてください。

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ワイルドカードを使用して、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 の場合`example.com`、ワイルドカードドメイン を含めることができます`*.example.com`。これは、 `product.example.com`や などのサブドメインを対象としています`api.example.com`。

カスタム証明書が ACM で利用可能になると、ドメインのセットアッププロセス中に選択できるようになります。に証明書をインポートする手順については AWS Certificate Manager、 AWS Certificate

Manager ユーザーガイドの「 への証明書 AWS Certificate Managerのインポート」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て再度インポートする責任はお客様にあります。

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か、カスタム証明書から マネージド証明書に変更することができます。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 で管理されているカスタムドメインを追加する

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 Management Console 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. 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 に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 プロバイダーによって管理され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 または Google ドメインを使用している場合は、これらのプロバイダーに固有の手順the section called “Google ドメインによって管理されるドメインの DNS レコードを更新する”についてthe section called “によって管理されるドメインの DNS レコードを更新する GoDaddy”または 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 Management Console 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。

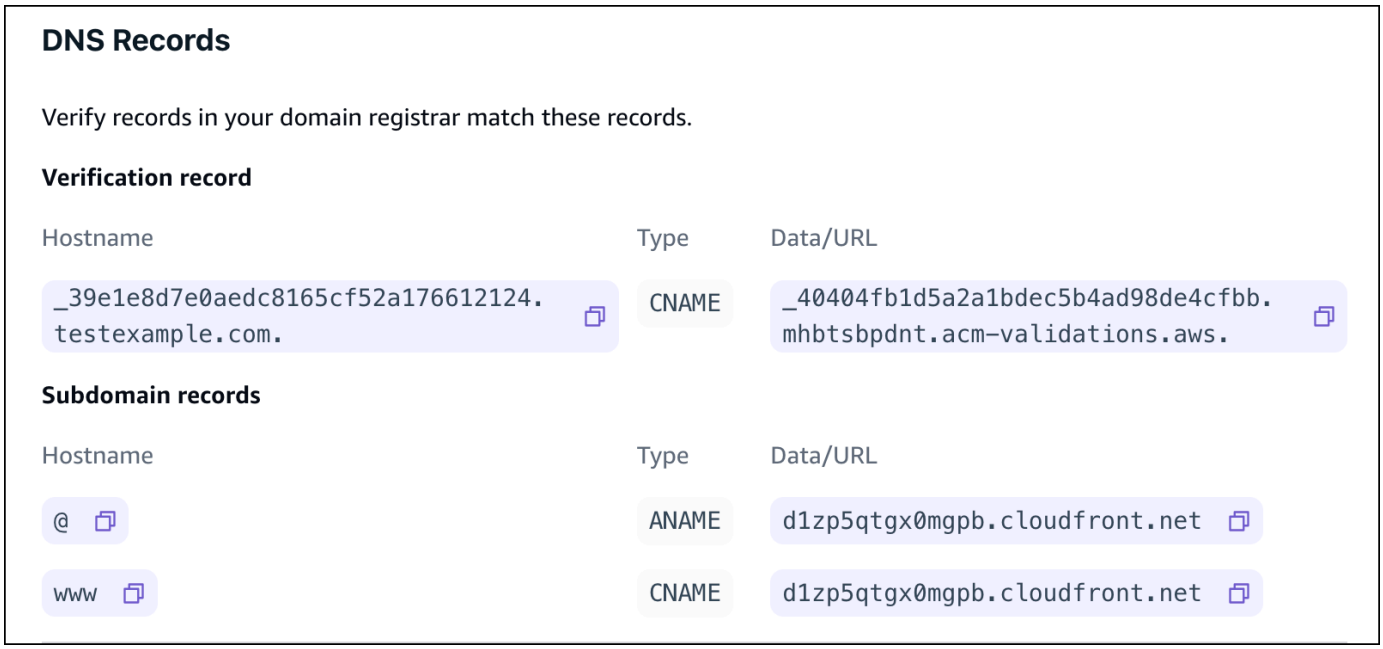
5. ルートドメインの名前を入力します。たとえば、ドメインの名前が https://example.com の場合は、**`example.com`** と入力します。
6. Amplify は、Route 53 ドメインを使用していないことを検出し、Route 53 でホストゾーンを作成するオプションを提供します。
   - Route 53 でホストゾーンを作成するには
     a. Route 53 でホストゾーンの作成を選択します。
     b. ドメイン の設定 を選択します。
     c. ホストゾーンネームサーバーがコンソールに表示されます。DNS プロバイダーのウェブサイトに移動し、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 に追加しました」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 に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 でホストゾーンの作成を選択した場合は、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。

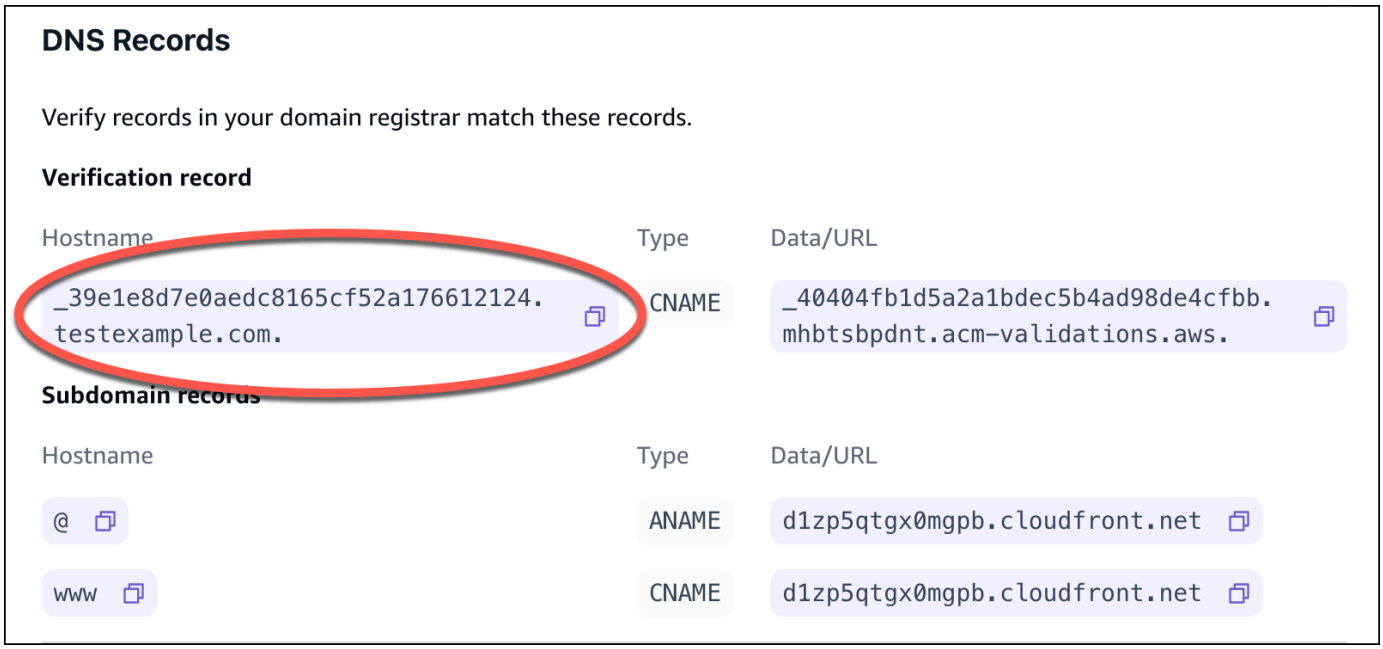


11. 次のいずれかを行います。

    - を使用している場合は GoDaddy、「」を参照してくださいによって管理されるドメインの DNS レコードを更新する GoDaddy。
    - Google Domains を使用している場合は、Google ドメインによって管理されるドメインの DNS レコードを更新する 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 レコードを設定します。
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。

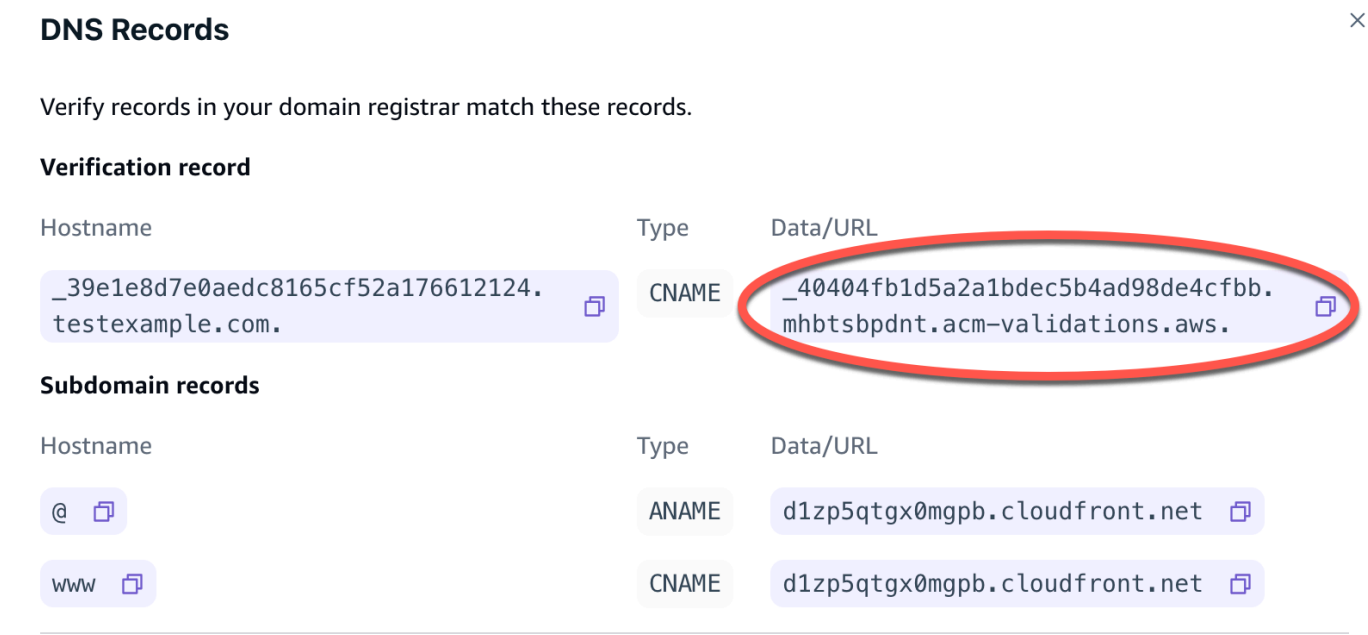
    Amplify コンソールに _c3e2d7eaf1e656b73f46cd6980fdc0e.example.com などのサブドメインの所有権を検証するための DNS レコードが表示される場合は、CNAME レコードのサブドメイン名 **`_c3e2d7eaf1e656b73f46cd6980fdc0e`**にのみ を入力します。

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る場合は、CNAME レコード値**_cjhwou20vhu2exampleuw20vuyb2ovb9.j9s73ucn9vy.acm-validations.aws**に 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。

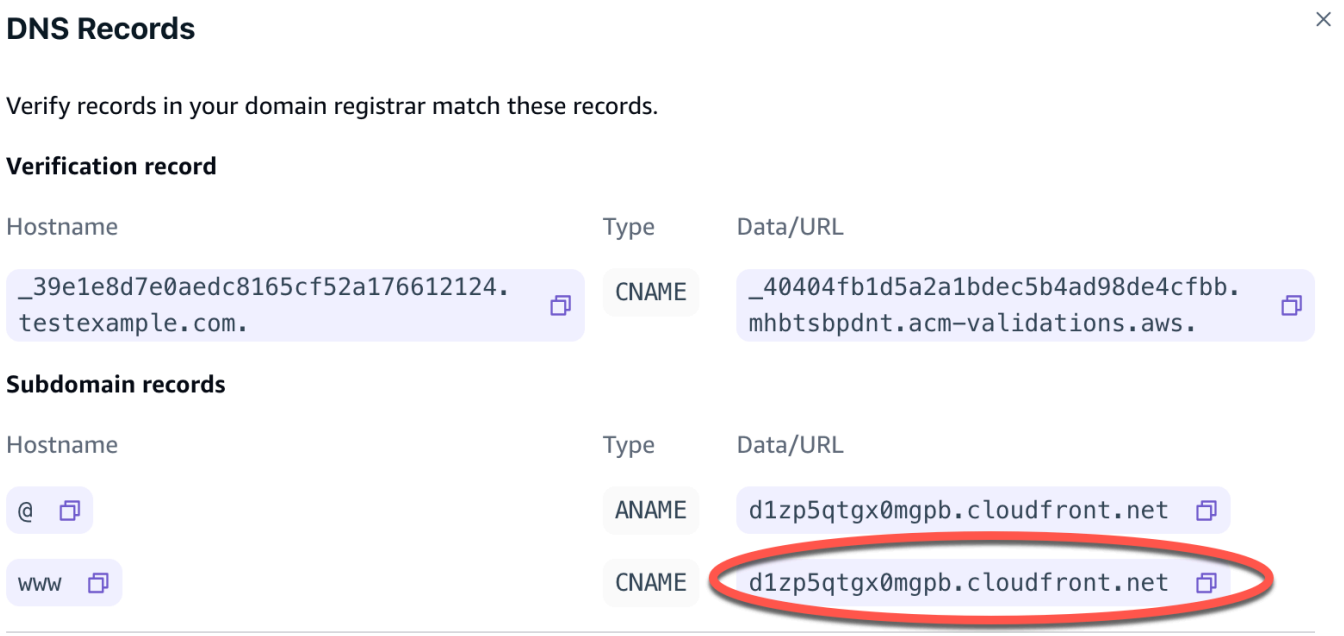
> **Important**
>
> Amplify コンソールでカスタムドメインを追加した直後に、このステップを実行することが重要です。 AWS Certificate Manager (ACM) は、所有権の検証をすぐに開始します。時間が経つにつれて、チェックの頻度は少なくなります。アプリを作成してから数時間後に CNAME レコードを追加または更新すると、アプリが検証保留中の状態で停止する可能性があります。

14. サブドメインを Amplify ドメインを指すように 2 番目の CNAME レコードを設定します。例えば、サブドメインが www.example.com の場合は、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ケーションのドメインが d111111abcdef8.cloudfront.net と表示される場合は、Amplify ドメイン**`d111111abcdef8.cloudfront.net`**に 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. アプリケーションのルートドメイン (https://example.com など) を指すように ANAME/ALIAS レコードを設定します。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。

によって管理されるドメインの DNS レコードを更新する GoDaddy

によって管理されるカスタムドメインを追加するには GoDaddy

1. で DNS レコードを更新する前に GoDaddy、手順 のステップ 1～9 を完了しますthe section called “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の追加”。
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検索し、DNS の管理 を選択します。
4. DNS ページに、DNS レコードセクションにドメインのレコードのリスト GoDaddy 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。

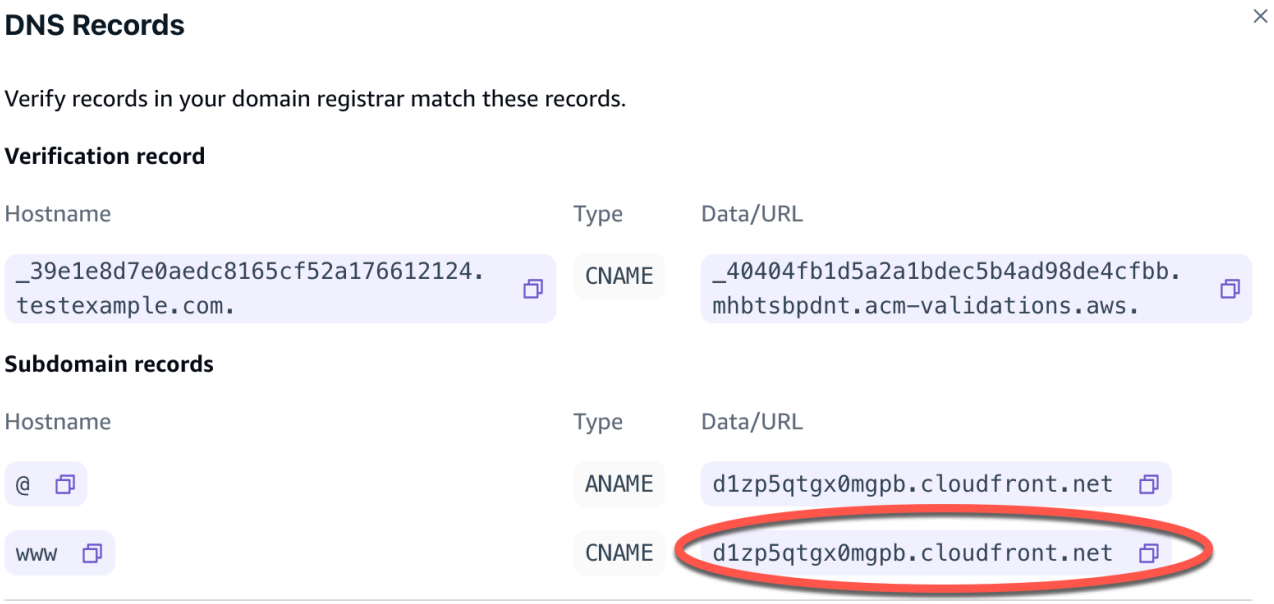
   a. 「DNS レコード」セクションで、「新しいレコードを追加」を選択します。

   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ケーションのドメインが d111111abcdef8.cloudfront.net と表示される場合は、値 **`d111111abcdef8.cloudfront.net`**に 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。

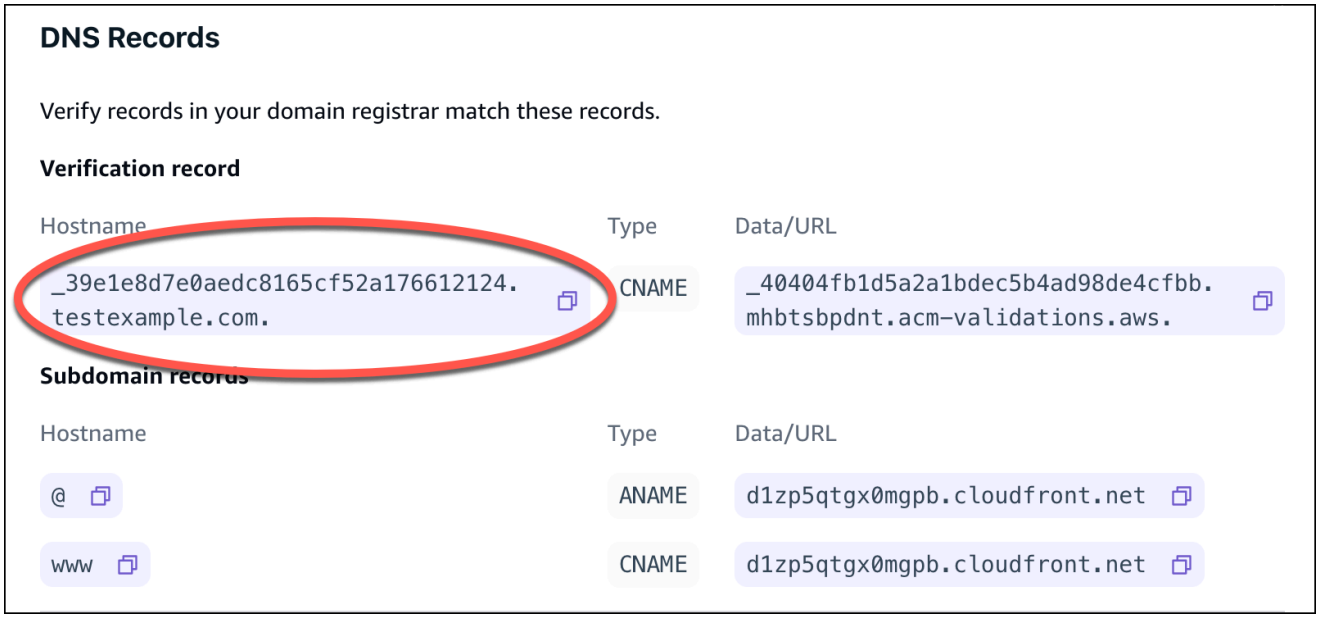
6. AWS Certificate Manager (ACM) 検証サーバーを指す 2 番目の CNAME レコードを作成します。検証済みの 1 つの ACM はドメインの SSL/TLS 証明書を生成します。

   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。

例えば、サブドメインの所有権を検証するための Amplify コンソールの DNS レコードが _c3e2d7eaf1e656b73f46cd6980fdc0e.example.com である場合は、名前 **_c3e2d7eaf1e656b73f46cd6980fdc0e**にのみ と入力します。

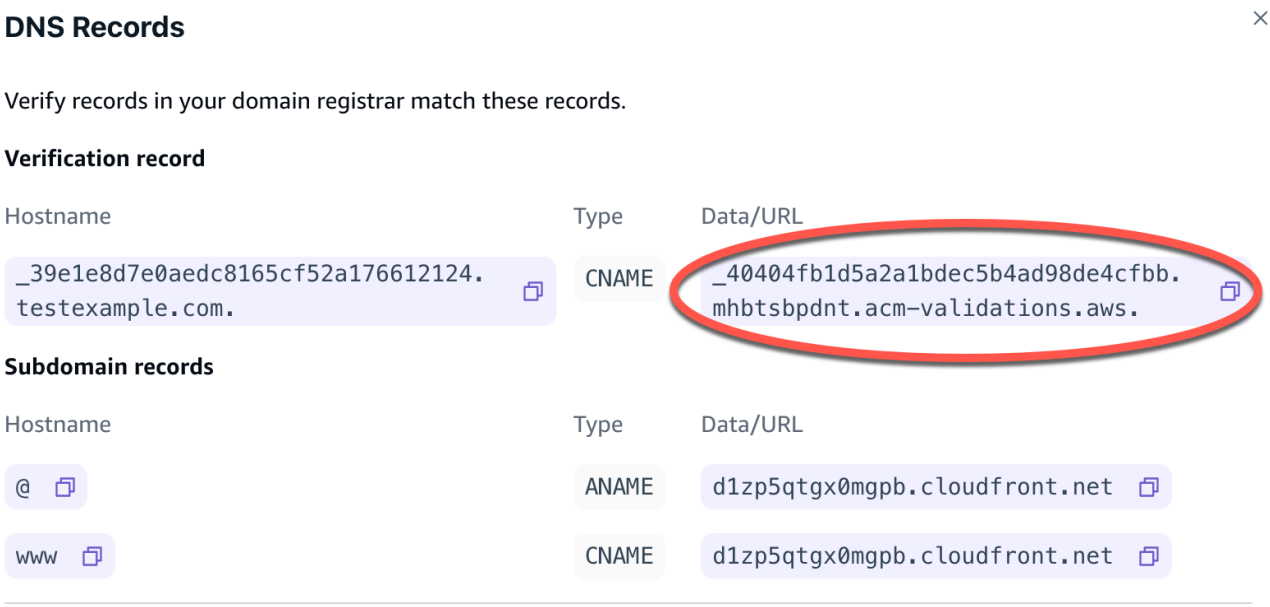
次のスクリーンショットは、使用する検証レコードの場所を示しています。



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、subdomains. GoDaddy does が ANAME/ALIAS レコードをサポートしていない場合には必要ありません。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。

   をプロバイダー GoDaddy 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、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、送信先 URL の転送先のサブドメインの名前 (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。

Google ドメインによって管理されるドメインの DNS レコードを更新する

Google Domains が管理するカスタムドメインを追加するには

1. Google ドメインで DNS レコードを更新する前に、「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」の手順のステップ 1～9 を完了します。
2. https://domains.google.com でアカウントにログインし、左側のナビゲーションペインで [マイドメイン] を選択します。
3. ドメインのリストで、追加するドメインを見つけて [管理] を選択します。
4. 左側のナビゲーションペインで、[DNS] を選択します。Google にドメインのリソースレコードが表示されます。2 つの新しい CNAME レコードを追加する必要があります。
5. 次のように、すべてのサブドメインが Amplify ドメインを指す最初の CNAME レコードを作成します。

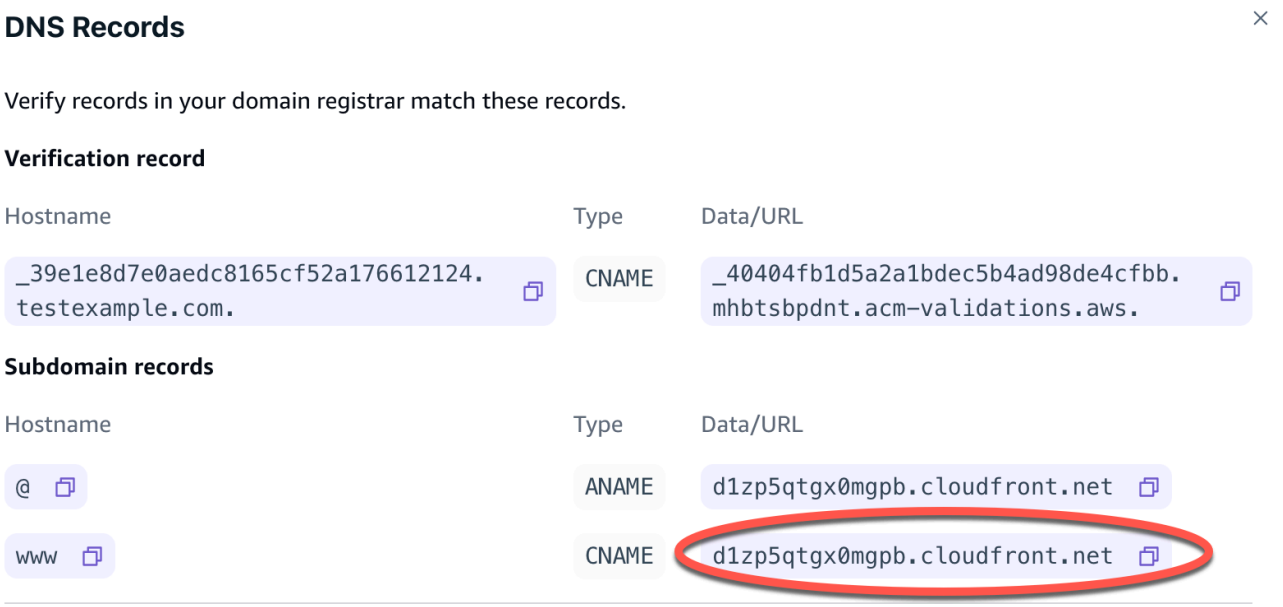
   a. [ホスト名] には、サブドメイン名のみを入力します。たとえば、サブドメインが www.example.com の場合、[ホスト名] に www と入力します。

   b. [タイプ]には [CNAME] を選択します。

   c. [データ] には、Amplify コンソールで使用できる値を入力します。

      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、[データ] 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。

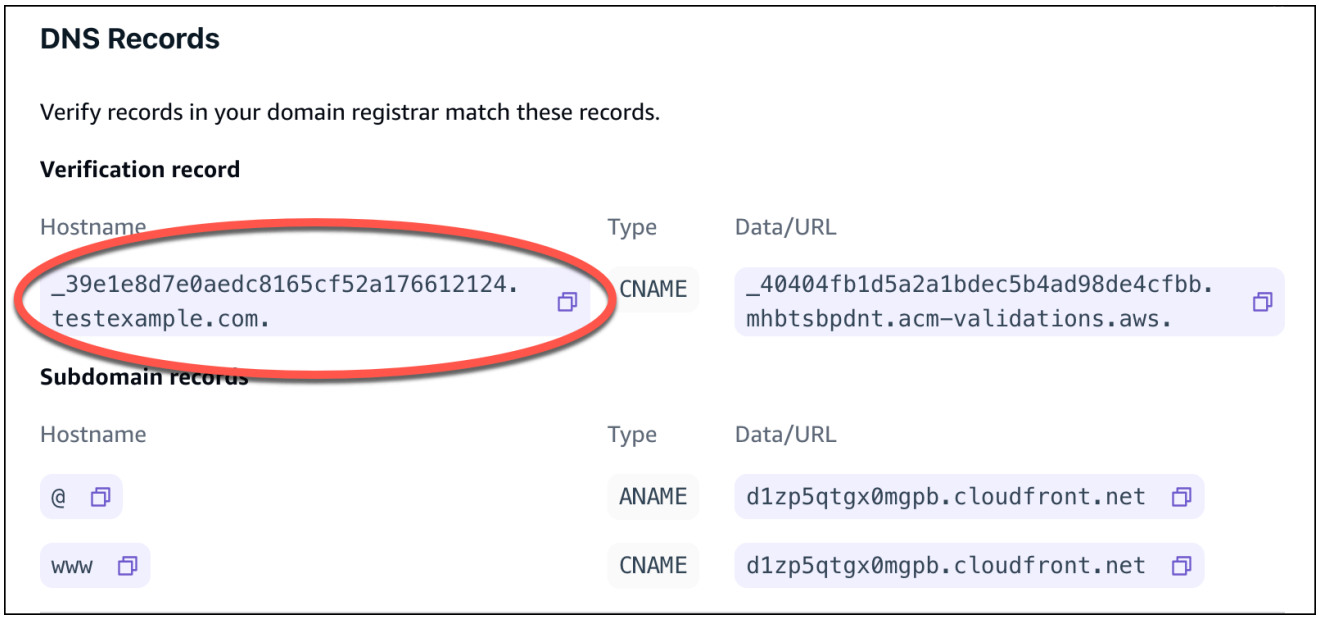


6. AWS Certificate Manager (ACM) 検証サーバーを指す 2 番目の CNAME レコードを作成します。検証済みの 1 つの ACM はドメインの SSL/TLS 証明書を生成します。

   a. [ホスト名] 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、[ホスト名] には _c3e2d7eaf1e656b73f46cd6980fdc0e のみを入力してください。

      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。

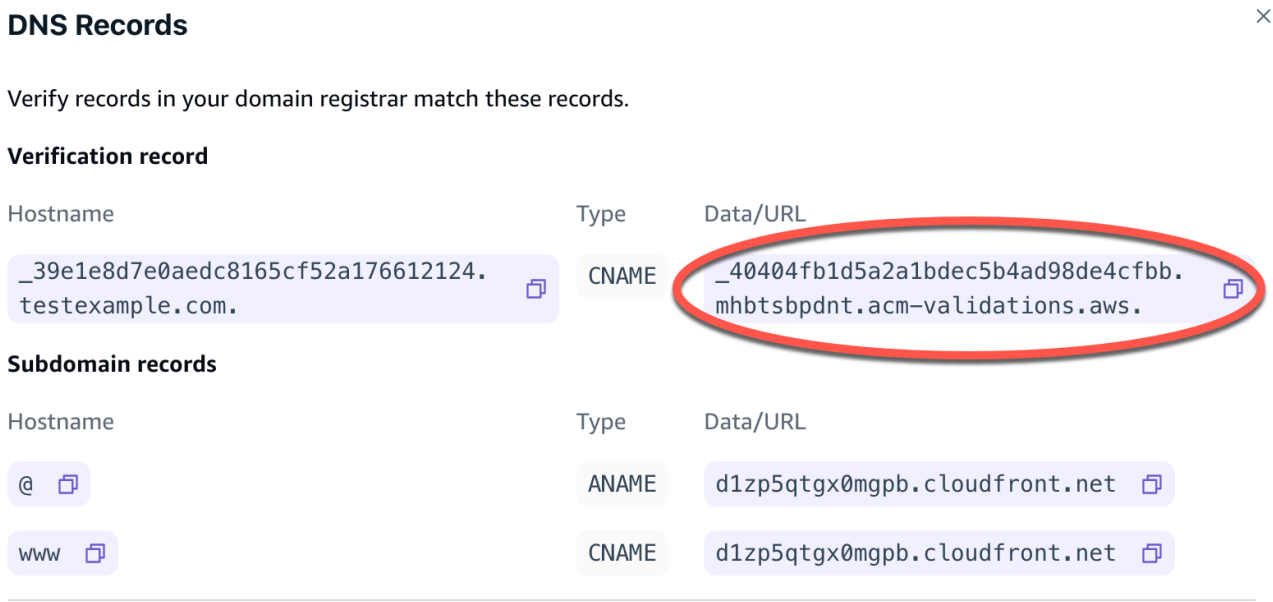


   b. [タイプ]には [CNAME] を選択します。

c. [データ] 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 _cf1z2npwt9vzexample93c1j4xzc92wl.2te3iym6kzr.acm-validations.aws の場合、[データ] には _cf1z2npwt9vzexample93c1j4xzc92wl.2te3iym6kzr.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。



7. [保存] を選択します。

> **Note**
>
> デフォルトの Amplify。 AWS Certificate Manager (ACM) によって生成された証明書は 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。

8. Google Domains の ANAME/ALIAS レコードのサポートはプレビュー段階です。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。Google Domains 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、サブドメイン転送を設定します。お使いの Google Domains のウェブサイトページを見つけます。次に、[ドメインを転送] を選択し、「ウェブ転送」ページで転送を設定します。

> **Note**
>
> Google ドメインの DNS 設定の更新が有効になるには、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」をご覧ください。

ドメインの SSL/TLS 証明書を更新する

ドメインに使用されている SSL/TLS 証明書はいつでも変更できます。例えば、マネージド証明書の使用からカスタム証明書の使用に変更できます。ドメインで使用されているカスタム証明書を変更することもできます。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。

ドメインで使用されている証明書またはカスタム証明書のタイプを更新するには、次の手順に従います。

ドメインの証明書を更新するには

1. にサインイン AWS Management Console 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、変更の種類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 では、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の詳細は、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を示します。

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 Management Console 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。例えば、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 Management Console 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. サブドメインを持つ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。

   例えば、alpha.example.com というドメインがあり、マルチレベルサブドメイン beta.alpha.example.com を作成する場合は、サブドメイン値としてベータを入力します。
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 Management Console 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. (オプション) 新しいサブドメインを追加するには、新しい を追加 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

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 Management Console 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 カスタムドメインに自動サブドメインを設定します。

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 Management Console 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

上記の手順を使用して自動サブドメイン作成を有効にすると、アプリのプルリクエストウェブプレビューにも自動的に作成された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、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 Web 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ケーション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、このセクションの以下のトピックを参照してトラブルシューティングのヘルプを参照してください。

ここで問題の解決策が見つからない場合は、 AWS Supportにお問い合わせください。詳細については、「AWS Support 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 が解決されたことを確認する方法
- 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
- CNAME AlreadyExistsException エラーが表示される
- 「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 エラーが表示される
- 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。

CNAME が解決されたことを確認する方法

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。

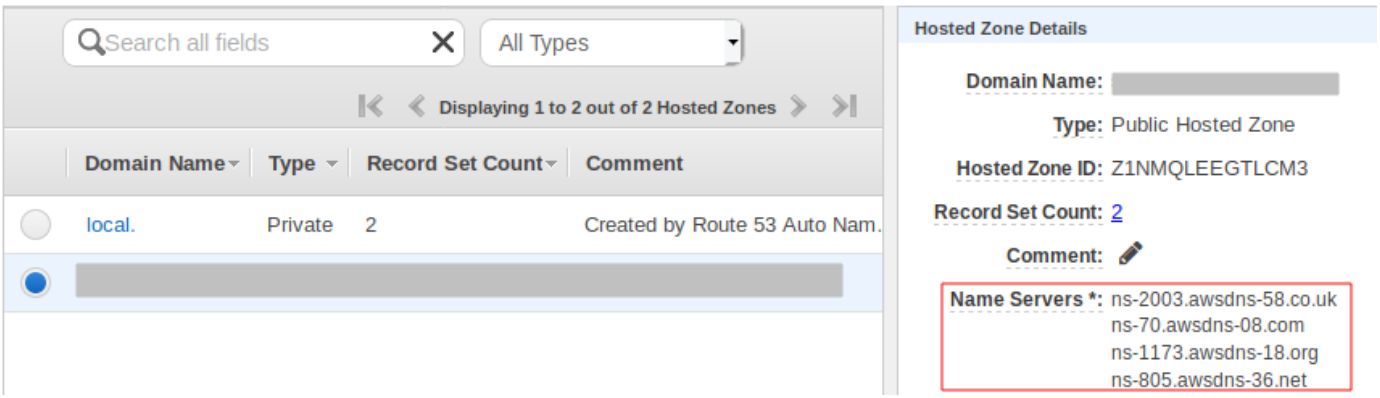
> **Important**
>
> カスタムドメインを作成したらすぐに CNAME レコードを更新することが重要です。アプリが Amplify コンソールで作成されると、CNAME レコードが数分ごとにチェックされ、解決したかどうかを判別します。1 時間経っても解決しない場合は、数時間ごとにチェックが行われるため、ドメインを使用する準備ができるまでに時間がかかる可能性があります。アプリを作成してから数時間後に CNAME レコードを追加または更新した場合、これがアプリが「検証保留中」の状態で停止する最も可能性の高い原因です。

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。

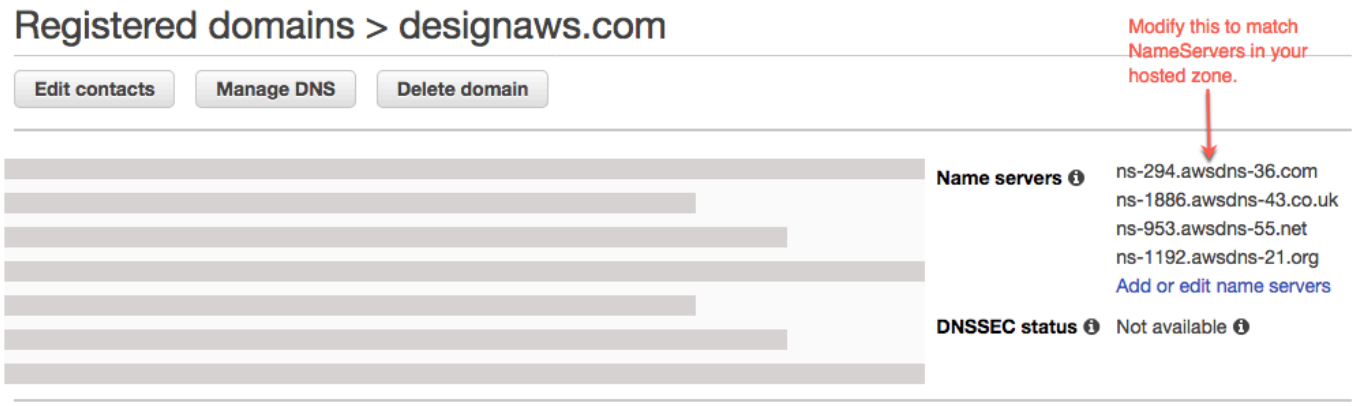
Amazon Route 53 でホストされているドメインが [Pending Verification] (検証待ち) 状態のままになっている

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、 AWS Supportにお問い合わせください。詳細については、「AWS Support 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。

CNAME AlreadyExistsException エラーが表示される

CNAMEAlreadyExistsException エラーが発生した場合、接続しようとしたホスト名の 1 つ (サブドメインまたは apex ドメイン) が既に別の Amazon CloudFront ディストリビューションにデプロイされていることを意味します。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 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。一度に 1 つのCNAMEレコードを 1 つの CloudFront ディストリビューションにアタッチできます。

2. 以前にドメインを CloudFront ディストリビューションにデプロイした場合は、ドメインを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. 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、他のサブドメインも使用できます。

4. 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引き続き表示される場合は、にお問い合わせください AWS Support。詳細については、「AWS Support 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。

「追加の検証が必要です」というエラーが表示されます。

追加の検証が必要な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 AWS Support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 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 エラーが表示される

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と、Amplify コンソールにアプリの

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証 (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 Management Console 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 で管理されているカスタムドメインを追加する.
- 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の追加.
- によって管理されるドメインの DNS レコードを更新する GoDaddy.
- Google ドメインによって管理されるドメインの DNS レコードを更新する.

- SSL 証明書を再発行 AWS Support するには、 にお問い合わせください。

ビルド設定の構成

Amplify ホスティングを使用してアプリをデプロイすると、リポジトリ内の 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が自動的に検出されます。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 コンソールから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。

Amplify コンソールでアプリのビルド設定を編集するには、「ホスティング」を選択し、ナビゲーションペインでビルド設定を選択します。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、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。

> **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。このタイプのデプロイの手順については、「 の開始方法」を参照してください。

ビルド仕様のコマンドと設定

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。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。

version

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

buildspec の cache フィールドは、node_modules フォルダーなどのビルド時の依存関係をキャッシュするために使用され、顧客のアプリに組み込まれているパッケージマネージャーとフレームワークに基づいて自動的に提案されます。最初のビルドでは、ここにあるすべてのパスがキャッシュされ、その後のビルドではキャッシュを再度増やし、ビルド時間を短縮するためにキャッシュされた依存関係を可能な限り使用します。

以下のビルド仕様例は、基本的な YAML 構文を示しています。

ビルド仕様 YAML 構文

```
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
        - path
test:
  phases:
    preTest:
      commands:
        - *enter command*
    test:
      commands:
        - *enter command*
    postTest:
      commands:
        - *enter command*
  artifacts:
    files:
        - location
        - location
    configFilePath: *location*
    baseDirectory: *location*
```

ブランチ固有のビルド設定

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。例えば、次のスクリプトでは、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 1 つの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
```

サブフォルダに移動する

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 アプリケーションのフロントエンドを使用した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適用されます。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、 という名前の非特権ユーザーでコードを実行しますamplify。Amplify は、Linux コマンドを使用して OS sudo 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 OS パッケージをインストールする場合は、 yumや などのコマンドを rpmで使用できますsudo。

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>
```

ビルドごとのキーと値のストレージ

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>
```

コミットのビルドをスキップする

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。

自動ビルドを無効にする

コードのコミットごとに自動ビルドを無効にするように Amplify を設定できます。セットアップするには、アプリ設定 、ブランチ設定 を選択し、接続されたブランチを一覧表示するブランチセクショ

ンを見つけます。ブランチを選択し、アクション 、自動ビルド を無効にする を選択します。そのブランチへの新しいコミットは、新しいビルドを開始しません。

差分ベースのフロントエンドビルドとデプロイを有効または無効にする

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 Management Console 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差分ベースのバックエンドビルドを有効または無効にする

> **Note**
>
> このセクションは、Amplify Gen 1 アプリケーションにのみ適用されます。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 Management Console 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。

モノレポのビルド設定

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

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。

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*
```

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 Management Console し、Amplify コンソール を開きます。
2. 右上隅にある新しいアプリの作成を選択します。
3. Amplify で構築を開始するページで、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. Select アプリがモノレポ
   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] をクリックします。
5. アプリ設定ページで、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AMPLIFY_MONOREPO_APP_ROOTに を設定します。
6. [次へ] をクリックします。
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

以下の手順に従って、Amplify に既にデプロイされている、または を使用して作成されたアプリケーションのAMPLIFY_MONOREPO_APP_ROOT環境変数を手動で設定します CloudFormation。

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 Management Console 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 を選択し、環境変数 を選択します。
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
```

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したら、追加の機能ブランチを作成します。

アプリケーションにブランチを追加するには

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、アプリは や https://main.appid.amplifyapp.com など、Amplify のデフォルトドメインで 2 つのデプロイを使用できますhttps://dev.appid.amplifyapp.com。これは とは異なる場合がありますが team-to-team、通常、メインブランチはリリースコードを追跡し、本番ブランチです。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 config の自動ビルド時間生成 (Gen 1 アプリケーションのみ)
- 条件付きバックエンドビルド (Gen 1 アプリケーションのみ)
- アプリ間で Amplify バックエンドを使用する (Gen 1 アプリのみ)

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のデベロッパー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。

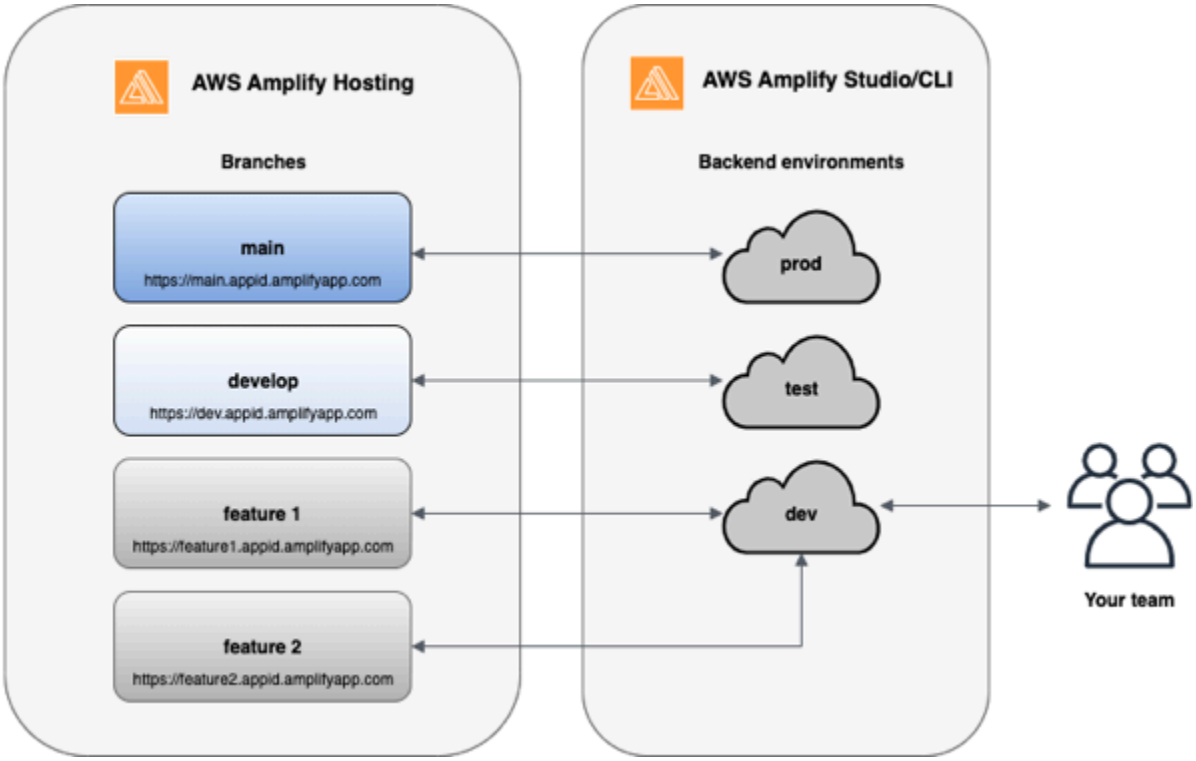
フルスタックの Amplify Gen 1 アプリを使用したチームワークフロー

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、「」を参照してくださいアプリケーションのバックエンドの構築。

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。

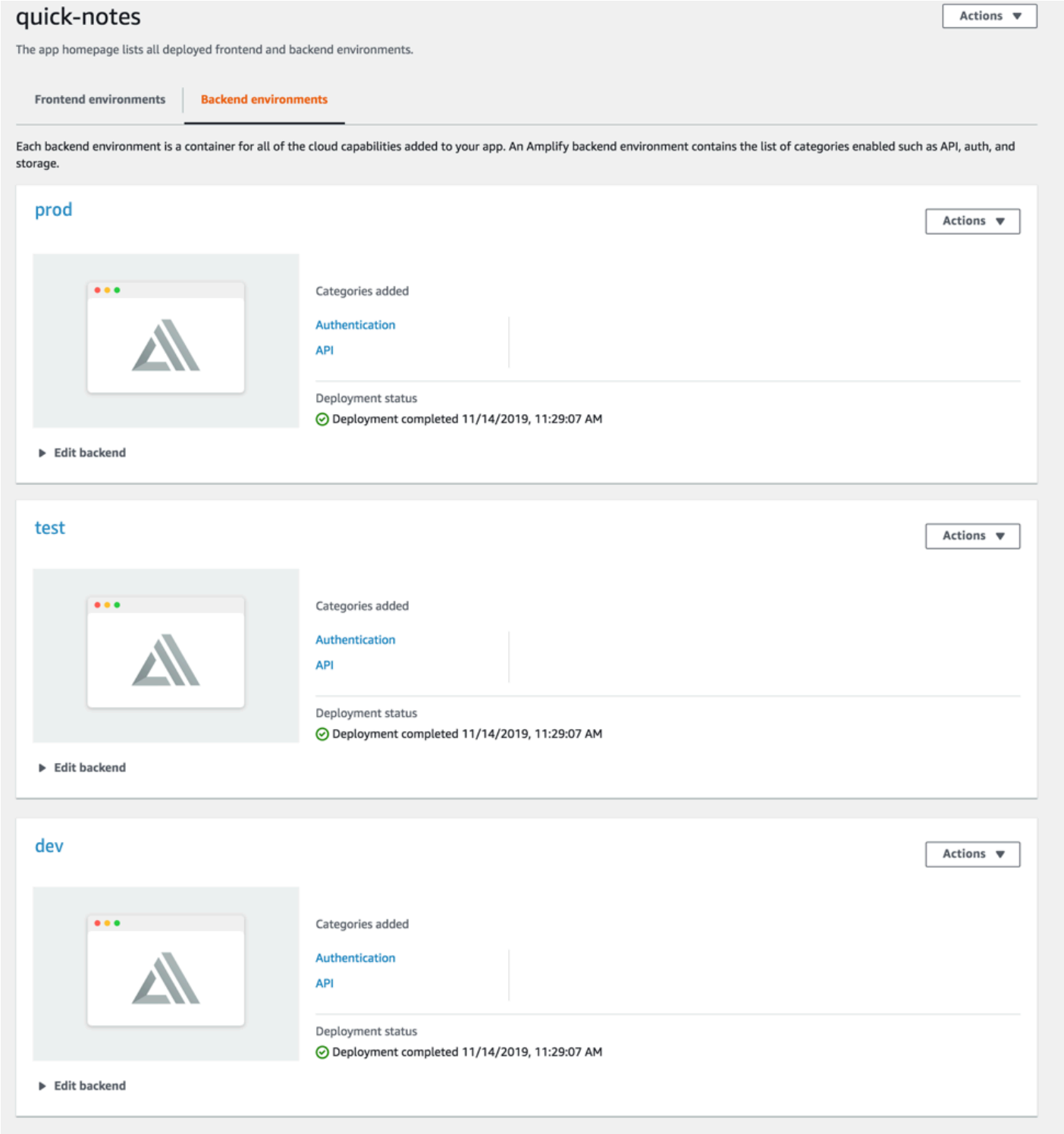
```
amplify env add
 ? Do you want to use an existing environment? (Y/n): n
 ? Enter a name for the environment: test
...
amplify push
```

```
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)。

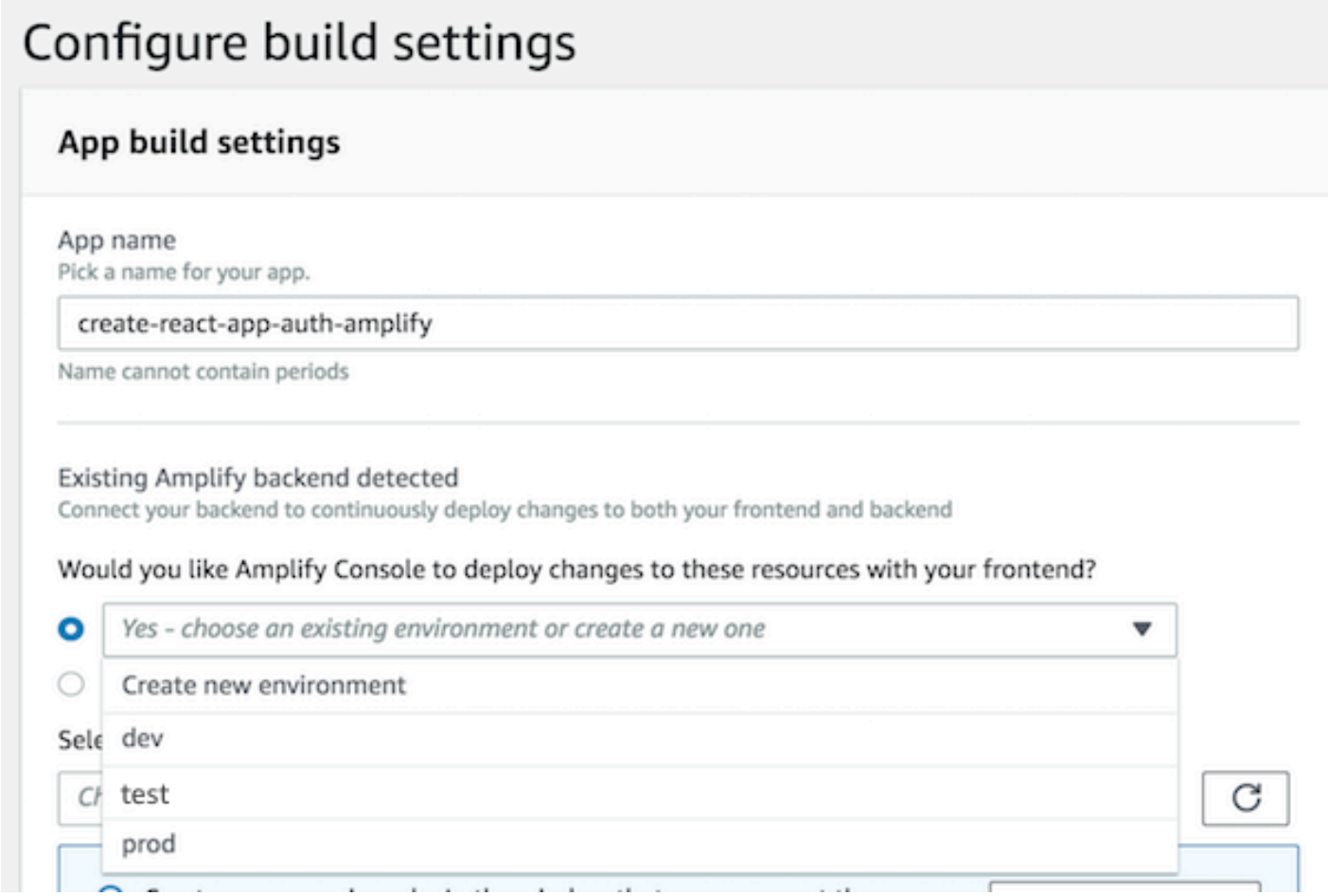
```
git commit -am 'Added dev, test, and prod environments'
git push origin main
```

5. の Amplify AWS Management Console 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。

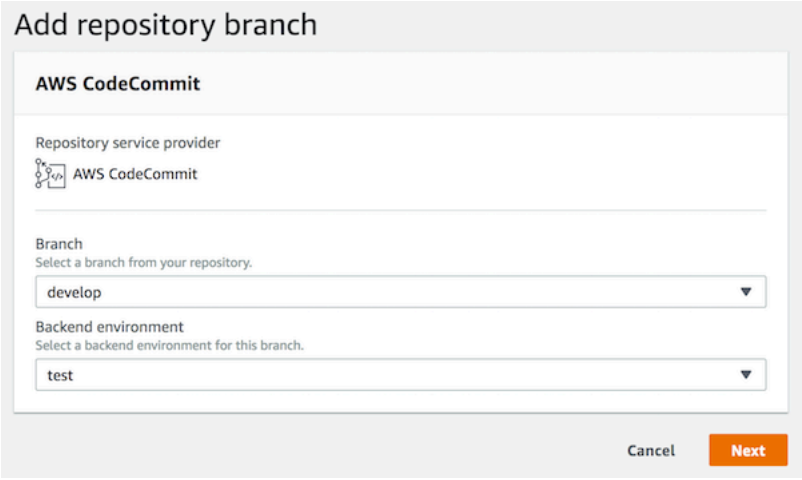


6. [フロントエンド環境] タブに切り替え、リポジトリプロバイダーと mainブランチを接続します。
7. ビルド設定画面で、既存のバックエンド環境を選択し、main ブランチとの継続的デプロイを設定します。ドロップダウンから prod を選択し、サービスロールを Amplify に付与します。[保存して

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)。

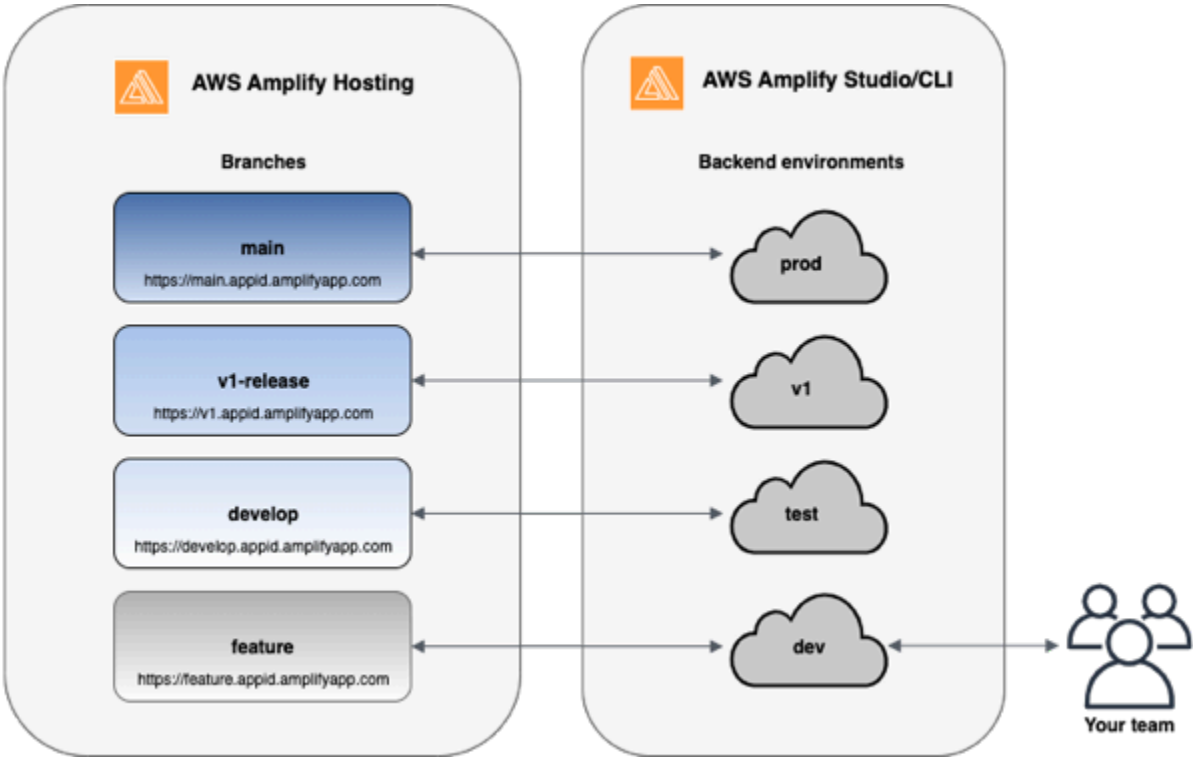
```
git push origin --delete newinternet
aws amplify delete-branch --app-id <appid> --branch-name <branchname>
```

```
amplify env remove dev
```

GitFlow ワークフロー

GitFlow は 2 つのブランチを使用してプロジェクトの履歴を記録します。メインブランチはリリースコードのみを追跡し、開発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、新しい開発を完了作業から分離することで、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)。

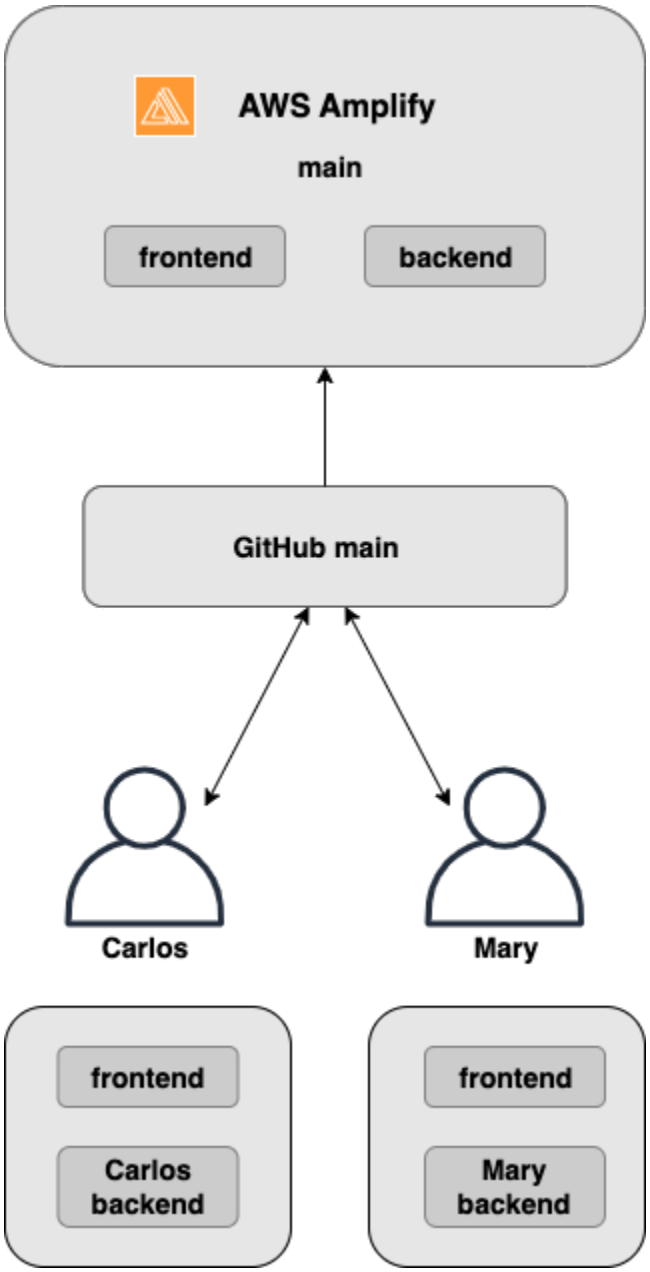
以下の図は、 で推奨されるセットアップを示しています GitFlow。上記の機能ブランチのワークフローのセクションで説明したプロセスと同じプロセスに従って行うことができます。



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 で開発ブランチを接続し (この時点で開発ブランチとメインブランチが同じであると仮定) 、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。リリースに特徴量ブランチまたは GitFlow ワークフローを使用する製品チームは、「release**」などの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。このブログ記事では、さまざまなチームワークフローでこの機能を使用する方法について説明します。

1. アプリ設定 > ブランチ設定 > 編集 を選択します。
2. 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ブランチを Amplify に自動的に接続します。
3. ブランチの自動検出 - 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*`です。

4. 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の自動パスワード保護を設定します。
5. Amplify バックエンドで構築された Gen 1 アプリケーションの場合、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。
6. [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 カスタムドメインに自動サブドメインを設定します。」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 で管理されているカスタムドメインを追加する 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。

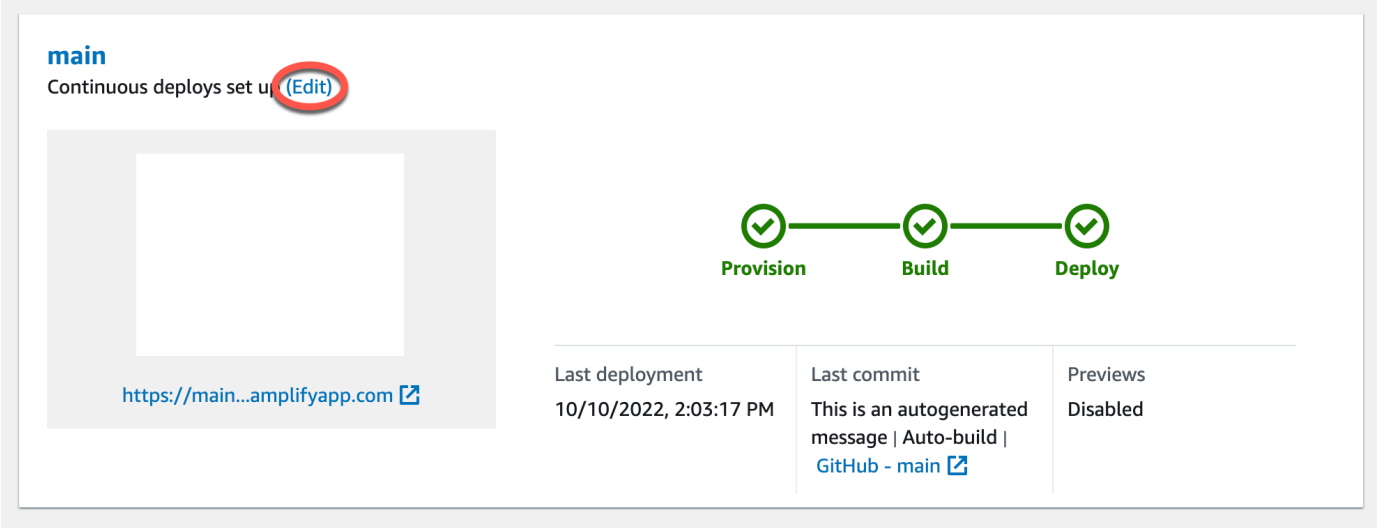
Amplify config の自動ビルド時間生成 (Gen 1 アプリケーションのみ)

> **Note**
>
> このセクションの情報は、Gen 1 アプリケーションのみを対象としています。Gen 2 アプリケーション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。

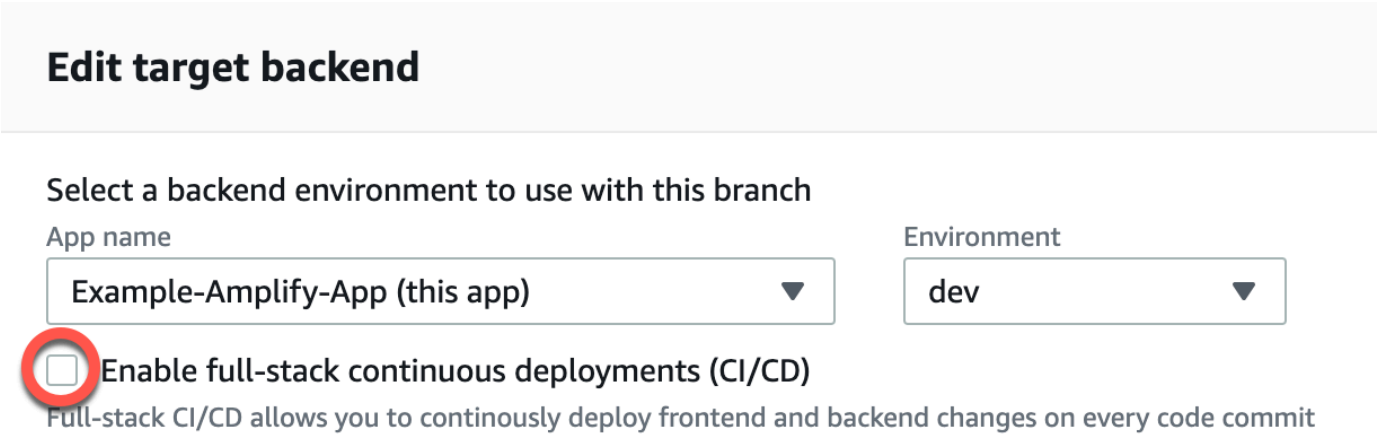
Amplify は、Gen 1 アプリケーション用の Amplify 設定aws-exports.jsファイルのビルド時自動生成をサポートしています。フルスタック CI/CD デプロイをオフにすることで、アプリによる aws-exports.js ファイルの自動生成が可能になり、ビルド時にバックエンドが更新されないようになります。

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 Management Console 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、「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 アプリケーションのみ)

> **Note**
>
> このセクションの情報は、Gen 1 アプリケーションのみを対象としています。Amplify Gen 2 では、 TypeScriptベースのコードファーストのデベロッパー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 アプリのすべてのブランチで条件付きバックエンドビルドをサポートしています。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差分ベースのバックエンドビルドを有効または無効にする 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、「ビルド仕様のコマンドと設定」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 アプリのみ)

> **Note**
>
> このセクションの情報は、Gen 1 アプリケーションのみを対象としています。Gen 2 アプリケーション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 を使用すると、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 Management Console 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、「サービスロールの追加」を参照してください。

e. [次へ] をクリックします。

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。

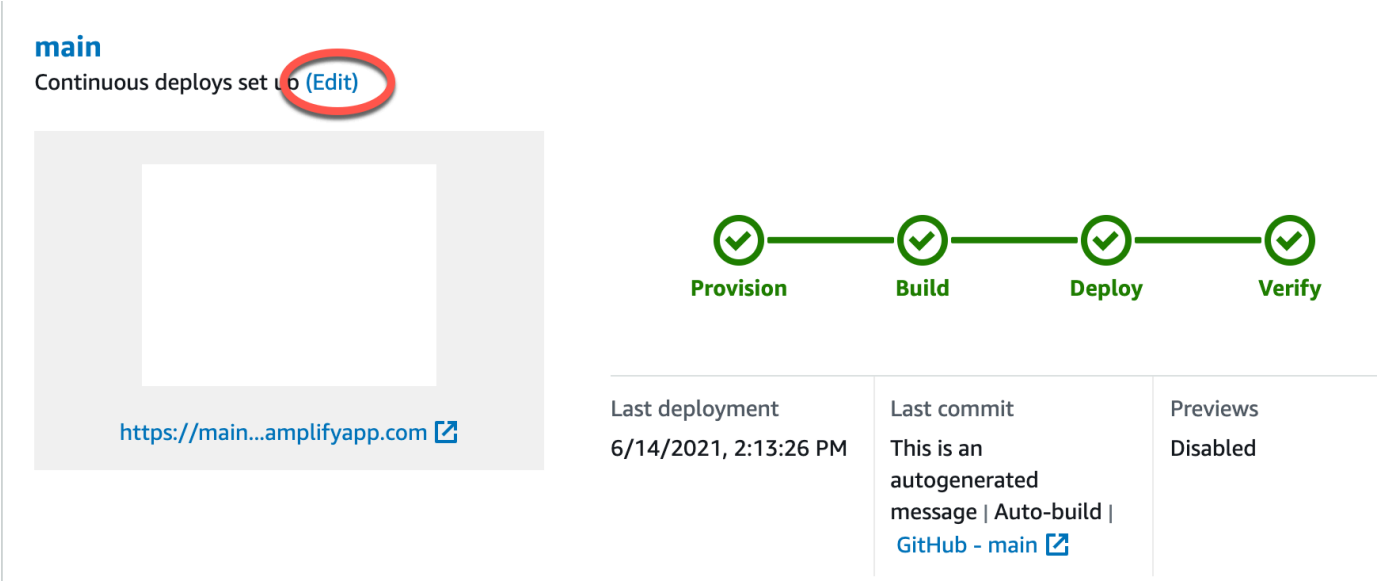
ブランチを既存の Amplify アプリに接続するときにバックエンドを再利用するには

1. にサインイン AWS Management Console 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、「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] をクリックします。
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 Management Console 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

を使用すると、 にデプロイされたデータ、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 AWS Amplify 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のデベロッパー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て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をお探しの場合は、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 アプリケーション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「開始方法」を参照してください。

Gen 1 アプリケーション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開始する前に、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他の AWS の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。

この例で使用する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 Management Console 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、または AWS CodeCommit リポジトリプロバイダーを選択し GitLab、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 アプリ機能を使用して Amplify アクセスを承認するようになりました。

   GitHub アプリのインストールと承認の詳細については、「」を参照してくださいGitHub リポジトリへの Amplify アクセスの設定。
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] をクリックします。
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 Management Console し、Amplify コンソール を開きます。
2. 「All apps」(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。

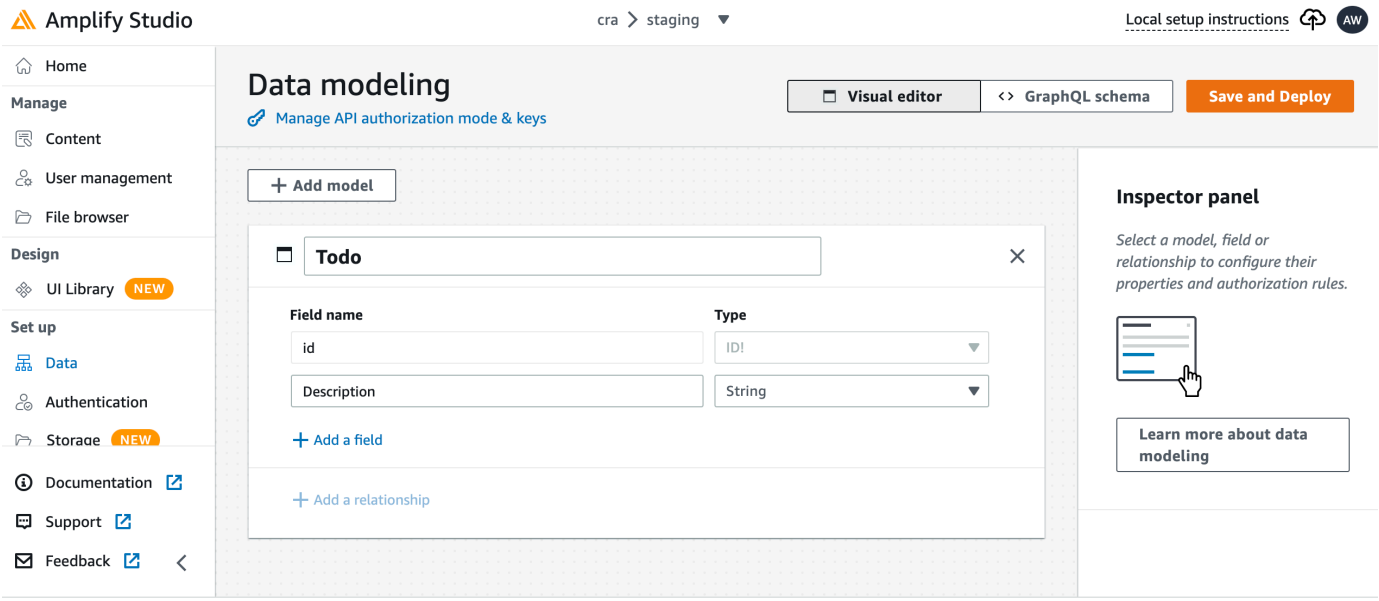
次のステップに従って、Amplify Studio ビジュアルバックエンドビルダーインターフェイスを使用して簡単なデータベースを作成します。

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、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 テーブルなど、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。Amplify は AWS 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 として保存できます infrastructure-as-code。

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。

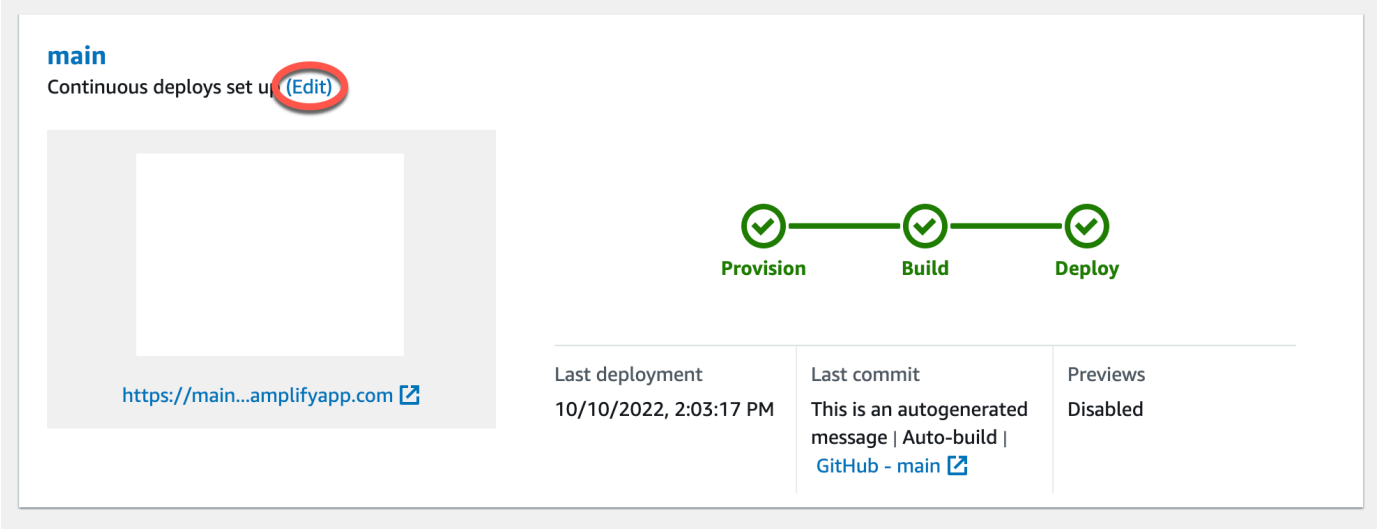
```
amplify pull --appId abcd1234 --envName staging
```

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、「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。

次のステップ

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、一連の UI コンポーネントを使用してフロントエンド ready-to-use UI を構築し、アプリのバックエンドに接続します。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。

手動デプロイ

手動デプロイを使用すると、Git プロバイダーに接続せずに、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。

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 ホスティングを使用してサーバー側でレンダリングされたアプリをデプロイします」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 Management Console し、Amplify コンソール を開きます。
2. 右上隅で、新しいアプリの作成 を選択します。
3. Amplify で構築を開始するページで、Git を使用せずにデプロイを選択します。[次へ] を選択します。
4. 「手動デプロイを開始する」セクションの [アプリ名] に、アプリの名前を入力します。
5. ブランチ名 には、 **`development`**や などのわかりやすい名前を入力します**`production`**。
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をドラッグアンドドロップするか、.zip フォルダを選択してコンピュータ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 zip 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

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 Management Console し、Amplify コンソール を開きます。
2. 右上隅で、新しいアプリの作成 を選択します。
3. Amplify で構築を開始するページで、Git を使用せずにデプロイを選択します。[次へ] を選択します。
4. 「手動デプロイを開始する」セクションの [アプリ名] に、アプリの名前を入力します。
5. ブランチ名 には、 **`development`**や などのわかりやすい名前を入力します**`production`**。
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. Amazon S3 バケット では、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、「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. Zip ファイルの場合は、デプロイする Zip ファイルの名前を選択します。
   - 任意の URL
     - [リソース URL] には、デプロイする ZIP ファイルの URL を入力します。
8. [保存してデプロイ] を選択します。

> **Note**
>
> ZIP 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。

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 のボタンをデプロイする

Deploy to Amplify Hosting ボタンを使用すると、 GitHub プロジェクトをパブリックに、またはチーム内で共有できます。以下は、ボタンの画像です。

リポジトリまたはブログに [Amplify ホスティングにデプロイ]ボタンを追加します

GitHub README.md ファイル、ブログ投稿、または HTML をレンダリングするその他のマークアップページにボタンを追加します。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 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、README.md などの GitHub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>
```

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
- AWS CloudFormation、CLI、および SDK デプロイメントのための Amplify GitHub アプリの設定
- 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証します。

Amplify Github アプリをインストールして認証するには

1. AWS Management Console にサインインし、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 の認証許可を求められます。[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証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. 次のいずれかを実行します。
   - 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. [Next] (次へ) をクリックします。
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証したら、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. AWS Management Console にサインインし、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認証」ページで、[GitHub アプリの設定] を選択します。
5. 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 認証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. 次のいずれかを実行します。
   - 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ケーションの「GitHub アプリのインストールと認証」ページにリダイレクトされます。GitHub の認証が成功すると、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。

AWS CloudFormation、CLI、および SDK デプロイメントのための Amplify GitHub アプリの設定

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)、AWS CloudFormation、または SDK を使用してデプロイしたアプリが含まれます。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 Management Console 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。

AWS CloudFormation、Amplify CLI および SDK を使用して、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 App 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. 次のいずれかを実行します。
   - 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. [Install] (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、AWS CloudFormation、または SDK を使用して Amplify アプリをデプロイするときにこれを提供する必要があります。

Amplify Github App を GitHub アカウントにインストールし、個人アクセストークンを生成したら、Amplify CLI、AWS CloudFormation、または SDK 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、Web プレビューを有効にする を参照してください。

プルリクエストの Web プレビュー

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 を使用してプロビジョニングされたバックエンド環境を持つアプリの場合、プルリクエスト (プライベート Git リポジトリのみ) ごとに一時的なバックエンドが作成され、PR が閉じられたときに削除されます。

アプリでウェブプレビューを有効にすると、各 PR はアプリごとに 50 ブランチの 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するには、PRsを必ず閉じてください。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、 をリポジトリプロバイダー AWS CodeCommit として使用する場合、`AWS_PULL_REQUEST_ID`環境変数は使用できません。

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、プレビューではリポジトリアクセスに Amplify GitHub アプリを使用します。アクセスのために OAuth を使用して GitHub リポジトリから以前にデプロイした既存の Amplify アプリでウェブプレビューを有効にする場合は、まず Amplify GitHub アプリ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。

> **Important**
>
> 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、Web 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。

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 を選択し、 をプレビューします。

> **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、次の操作を実行して、アカウントに Amplify GitHub アプリをインストールして承認します。

   a. GitHub アプリ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 アプリのインストール を選択します。

   b. Amplify GitHub アプリを設定する 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、設定の編集を選択します。

4. プレビュー設定の管理ページで、プルリクエストプレビュー をオンにします。[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、GitHub アカウントのプルリクエストから直接 URL のプレビューにアクセスできます。

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 によって管理されるカスタムドメインに接続されている Amplify アプリケーションのサブドメインからアクセスできます。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 Web 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 カスタムドメインに自動サブドメインを設定します。」を参照してください。

Amplify アプリに end-to-end Cypress テストを追加する

end-to-end (E2E) テストを Amplify アプリケーションのテストフェーズで実行して、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キャッチできます。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。

チュートリアル: Cypress でテストを設定する end-to-end

Cypress は、ブラウザで E2E テストを実行できるようにする JavaScriptベースのテストフレームワークです。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 を使用したフルスタック CI/CD デプロイの end-to-end Cypress テストの実行」を参照してください。

既存の Amplify アプリにテストを追加

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 のレポートは出力 JSON 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。

アーティファクト > 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。

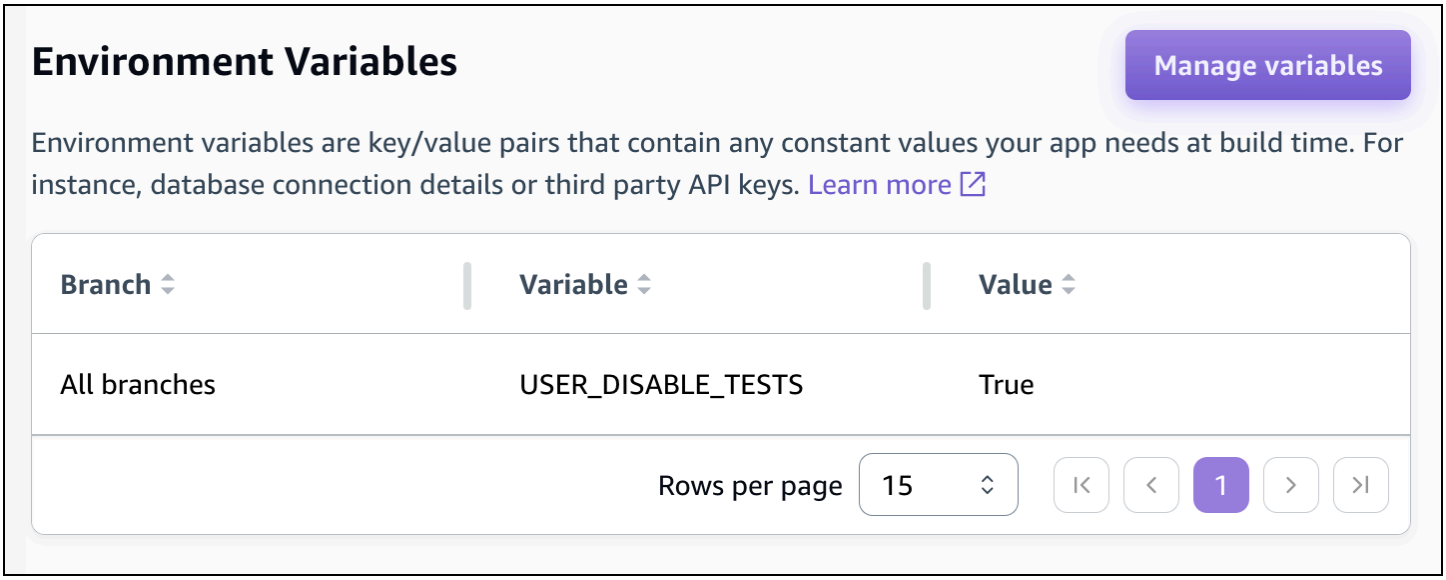
以下のビルド仕様 amplify.yml ファイルからの抜粋例は、Cypress によるテストをアプリに追加する方法を示しています。

```
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'
```

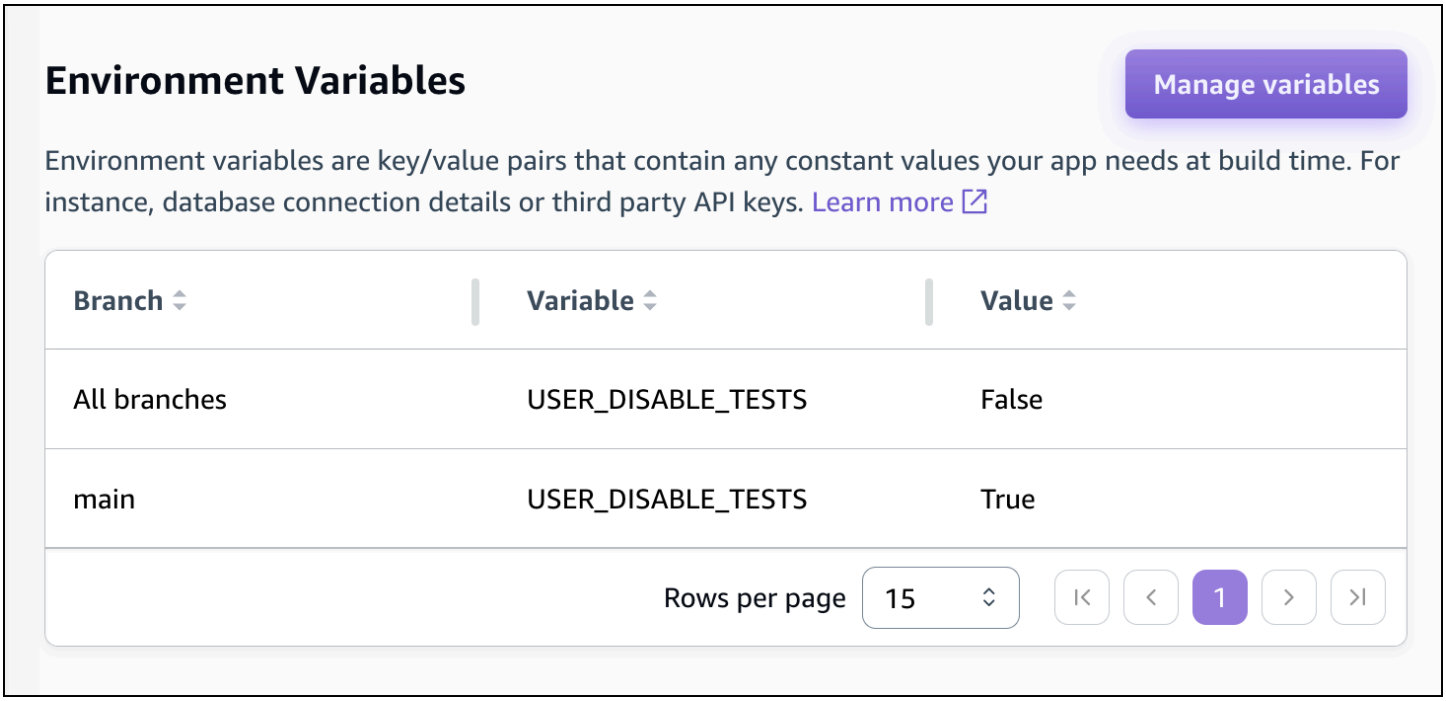
テストを無効にする

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。

リダイレクトを使用する

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。

リダイレクトの種類

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 Management Console 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、書き込みとリダイレクト 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. リダイレクトを追加する手順は、ルールを個別に追加するか、一括編集するかによって異なります。
   - 個々のリダイレクトを作成するには、「書き換えを追加」を選択します。
     a. 元のアドレスには、ユーザーがリクエストした元のアドレスを入力します。
     b. [ターゲットアドレス] 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入力します。
     c. [タイプ] には、一覧からリダイレクトの種類を選択します。
     d. (オプション) 国コードには、2 文字の国コード条件を入力します。
   - リダイレクトを一括編集するには、[テキストエディタを開く] を選択します。
     - Rewrites and redirects JSON エディタでリダイレクトを手動で追加または更新します。
6. [保存] を選択します。

リダイレクトの順序

リダイレクトはリストの上部から下に適用されます。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
```

クエリパラメータ

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。

シンプルなリダイレクトと書き換え

このセクションには、一般的なリダイレクトシナリオのサンプルコードが含まれています。

> **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。

次のサンプルコードを使用して、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 [{"source": "/original.html", "status": "301", "target": "/destination.html", "condition": null}]

次のサンプルコードを使用して、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 [{"source": "/docs/<*>", "status": "301", "target": "/documents/<*>", "condition": null}]

次のサンプルコードを使用して、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 [{"source": "/<*>", "status": "200", "target": "/index.html", "condition": null}]

次のサンプルコードを使用して、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 [{"source": "https://mydomain.com", "status": "200", "target": "https://www.mydomain.com", "condition": null}]

次のコード例を使用して、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 [{"source": "https://mydomain.com", "status": "302", "target": "https://www.mydomain.com/documents/", "condition": null}]

次のサンプルコードを使用して、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 [{"source": "/<*>", "status": "404", "target": "/404.html", "condition": null}]

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

ほとんどの SPA フレームワークは、サーバーリクエストを開始せずにブラウザの場所を変更するための HTML5 history.pushState () をサポートしています。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|	/index.html	200	

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
`json|webp)$)([^.]+$)/>`			

JSON [{"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。

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 [{"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`

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/index.html

プレースホルダー

次のサンプルコードを使用して、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 [{"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

次のサンプルコードを使用して、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 [{"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。

次のサンプルコードを使用して、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 [{"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

次のサンプルコードを使用して、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 [{"source": "/documents", "status": "302", "target": "/documents/us/", "condition": "<US>"}]

リダイレクトと書き換えのワイルドカード式

ワイルドカード式 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。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、別の場所で使用します。

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 つの例は、ワイルドカード式を使用した無効なリダイレクトを示しています。

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/<*>/content	/documents/<*>/content	permanent redirect (301)	
/docs/<*>/content/<*>	/documents/<*>/content/<*>	permanent redirect (301)	

ネットワークへのアクセスを制限する

リリースされていない機能を使用している場合は、機能ブランチをパスワードで保護して、特定のユーザーへのアクセスを制限できます。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されるパスワードを設定することも、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。

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 Management Console 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、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の管理ページで、次のいずれかを実行します。
   - 接続され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 のアクセス管理を有効にします。例えば、メインブランチ、開発ブランチ、および機能ブランチ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とパスワードを適用できます。
   - ユーザー名とパスワードを個々のブランチに適用するには
     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見つけます。アクセス設定 で、必須のパスワードの制限 を選択します。
     c. ユーザー名 に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。

環境変数

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プレフィックス付きの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 の内部使用のみを目的としています。
- 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 アプリケーションの場合、AWS Systems Manager Parameter Store 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

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)。	`30`
_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"}]`

変数名	説明	値の例
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。 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、「テストを無効にする」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`
AWS_COMMIT_ID	現在のビルドのコミット ID 再ビルドの「HEAD」	`abcd1234`
AWS_JOB_ID	現在のビルドのジョブ ID。 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`

変数名	説明	値の例
AWS_PULL_REQUEST_ID	プルリクエストウェブプレビュービルドのプルリクエスト ID。 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 コンソールで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の送信先の Amplify コンソール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`
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`

変数名	説明	値の例
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を有効または無効にする」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 アプリケーション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差分ベースのバックエンドビルドを有効または無効にする」を参照してください。	`true`
AMPLIFY_BACKEND_PULL_ONLY	Amplify は、この環境変数を管理します。これは Gen 1 アプリケーション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`

変数名	説明	値の例
AMPLIFY_BACKEND_APP_ID	Amplify は、この環境変数を管理します。これは Gen 1 アプリケーション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 アプリケーションにのみ適用されます。	`true`
AMPLIFY_ENABLE_DEBUG_OUTPUT	この変数を `true`に設定すると、スタックトレースがログに出力されます。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`
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`

変数名	説明	値の例
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。 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。 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。詳細については、「Amplify生成ロールの IAM アクセス許可の境界」を参照してください。	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 アクセスキーおよびシークレットキーではありません。

環境変数を設定する

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には、以下の手順に従います。

> **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 Management Console 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 を選択し、環境変数 を選択します。
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。

ビルド時に環境変数にアクセスする

ビルド中に環境変数にアクセスするには、ビルドコマンドに環境変数が含まれるようにビルド設定を編集します。

ビルド設定内の各コマンドは Bash シェル内で実行されます。Bash での環境変数の操作について詳しくは、GNU Bash マニュアルの「シェル拡張」を参照してください。

環境変数を含むようにビルド設定を編集するには:

1. にサインイン AWS Management Console 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 を選択し、ビルド設定 を選択します。
3. 「アプリビルド仕様」セクションで、「編集」を選択します。
4. 環境変数をビルドコマンドに追加します。これで、次回のビルド時に自分の環境変数にアクセスできるようになります。この例では npm の動作 (BUILD_ENV) を変更し、外部サービスの API トークン (TWITCH_CLIENT_ID) を後で使用できるように環境ファイルに追加します。

```
build:
  commands:
    - npm run build:$BUILD_ENV
    - echo "TWITCH_CLIENT_ID=$TWITCH_CLIENT_ID" >> backend/.env
```

5. [保存] を選択します。

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。

Next.js サーバーコンポーネントは、デフォルトではアプリ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。

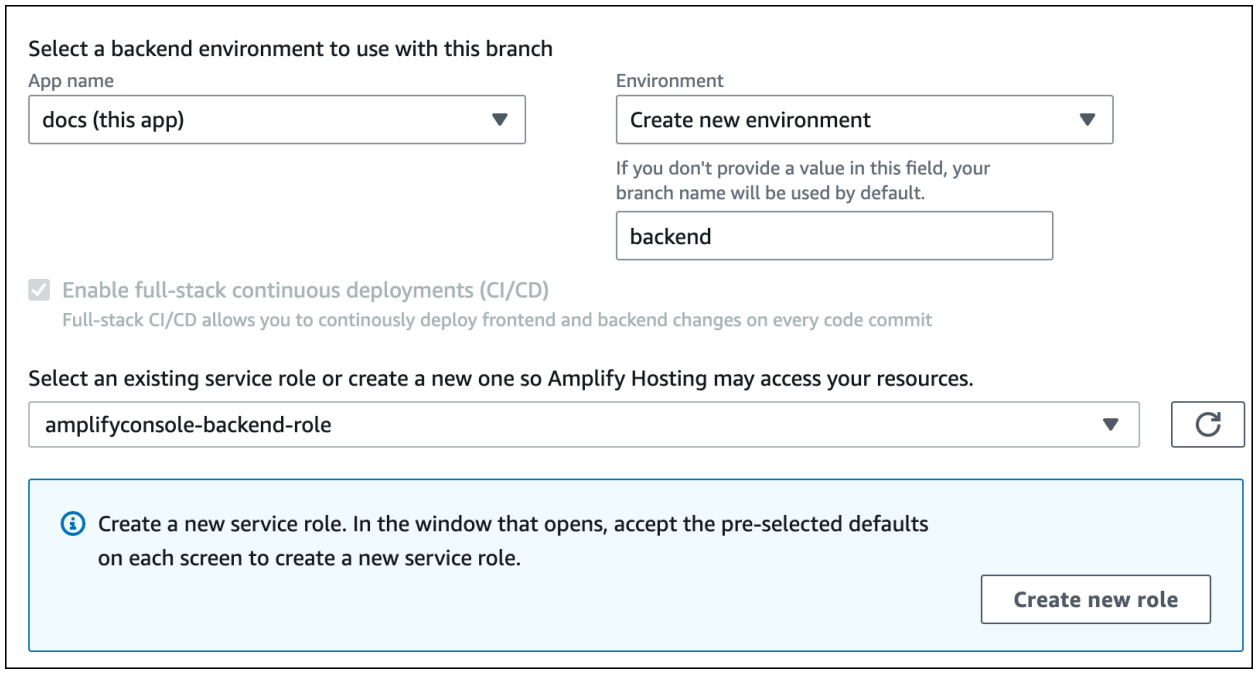
特定の環境変数を Next.js にアクセスできるようにするには、Amplify ビルド仕様ファイルを変更して Next.js が認識する環境ファイル内の環境変数を設定する必要があります。これにより、Amplify はアプリケーションをビルドする前に環境変数をロードできます。ビルド仕様の変更について詳しくは、ビルドコマンドセクションの環境変数の追加方法の例を参照してください。

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 Management Console 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。

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。この目的で Amplify のビルド時環境変数を使用する例については、ビルド時に環境変数にアクセスするを参照してください。

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合理化され、Amplify コンソールで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 アプリケーションの環境シークレットは環境変数に似ていますが、暗号化できる AWS Systems Manager Parameter Store 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。

Gen 1 アプリケーションの環境シークレットを設定してアクセスする

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、以下の手順に従います。

環境シークレットを設定するには

1. にサインイン AWS Management Console 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. [Type] (タイプ) で、SecureString 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 が値を復号 AWS KMS key 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。

環境シークレットにアクセスする

ビルド中に Gen 1 アプリケーションの環境シークレットにアクセスすることは、環境シークレットが JSON 文字列として に保存されている点を除いて、環境変数 にアクセスするのと似ています。`process.env.secrets`

Amplify 環境のシークレット

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----- ****..... -----プライベートキーを終了-----

カスタムヘッダー

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。Amplify コンソールでヘッダーを指定するか、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ヘッダーを指定できます。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、 でビルド仕様 (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 AWS Management Console することで、アプリケーションにカスタム HTTP ヘッダーが指定されていました。この方法で指定されたカスタムヘッダーは、buildspec と amplify.yml ファイルの外に移行する必要があります。手順については、「カスタムヘッダーの移行」を参照してください。

カスタムヘッダー YAML 形式

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

カスタムヘッダーは、パターンに一致するすべての URL ファイルパスに適用されます。

headers

ファイルパターンと一致するヘッダーを定義します。

キー

カスタムヘッダーの名前。

value

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。Amplify コンソールでヘッダーを指定するか、アプリケーションの `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ヘッダーを指定できます。

アプリケーション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 Management Console 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、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 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 Management Console 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` ファイルで設定され、アプリケーション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を上書きします。

カスタムヘッダーの移行

以前は、Amplify コンソールで 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、アプリケーションにカスタム HTTP ヘッダーが指定されていました。カスタムヘッダーを buildspec と `amplify.yml` ファイルの外に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、`customHttp.yml` ファイルをダウンロードして編集します。

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 Management Console 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 ウィンドウで、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 Management Console 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、「 のダウンロード」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、アプリケーション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を上書きします。

モノレポカスタムヘッダー

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 YAML 形式 を参照してください。

- 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、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を上書きします。

セキュリティヘッダー例

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'"
```

カスタムキャッシュコントロールヘッダー

Amplify でホストされるアプリケーションは、オリジンによって送信される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ステータスコードを含む正常なレスポンスにのみ Cache-Control カスタムヘッダーを適用します。これにより、エラーレスポンスがキャッシュされ、同じリクエストを行う他のユーザーに提供されるのを防ぐことができます。

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、「ヘッダーを使用したキャッシュ保持期間の制御」を参照してください。

着信ウェブフック

Amplify コンソールで受信ウェブフックを設定し、コードを Git リポジトリにコミット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）でウェブフックリガーを使用すると、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 Management Console 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 を選択し、次にビルド設定 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 ツールや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。

モニタリング

AWS Amplify は Amazon を通じてメトリクスを出力 CloudWatch し、アプリケーション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。このセクションのトピックを利用して、これらのメトリクスとログによってアプリを監視する方法を学びます。

トピック

- によるモニタリング CloudWatch
- アクセスログ

によるモニタリング CloudWatch

AWS Amplify は Amazon と統合されているため CloudWatch、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できます。メトリック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 サービスの仕組みの詳細については、「Amazon CloudWatch ユーザーガイド」を参照してください。

メトリクス

Amplify は、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するために、 `AWS/AmplifyHosting`名前空間で 6 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 サービスクォータ にはカウントされません。

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。次の表では、最も関連性の高い統計を各メトリクスの説明にまとめています。

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。 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

メトリクス	説明
	タGET、HEAD、OPTIONSの総量 (バイト単位) 。 最も関連性の高い統計はSumです。
BytesUploaded	アプリに転送 (アップロード) されたデータの総量 (POSTとPUTの使用量とリクエスト数) をバイト単位で表したものです。 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。 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。 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。 最も関連性の高い統計はAverage、Maximum、Minimum、p10、p50、p90p9す。 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。

Amplify は、次の CloudWatch メトリクスディメンションを提供します。

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。
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 メトリクスには、https://console.aws.amazon.com/cloudwatch/ AWS Management Console でアクセスできます。または、次の手順に従って、Amplify コンソールのメトリクスにアクセスできます。

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 Management Console 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、[アプリの設定]、[モニタリング]の順に選択します。
4. [概要]ページで、[メトリクス]を選択します。

アラーム

特定の基準が満たされたときに通知を送信する CloudWatch アラームを Amplify コンソールで作成できます。アラームは 1 つの CloudWatch メトリクスを監視し、メトリクスが指定された評価期間数のしきい値を超えたときに Amazon Simple Notification Service 通知を送信します。

コンソールまたは CloudWatch APIs を使用して、メトリクスの数式 CloudWatchを使用する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 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 Management Console 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、[組織の設定]、[モニタリング]の順に選択します。
4. [モニタリング]ページで、[アラーム]を選択します。
5. [アラームを作成] を選択します。
6. 「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. アラームを呼び出すためにアラーム状態にある必要がある連続す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. [アラームを作成] 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。

Amazon CloudWatch Logs for SSR アプリ

Amplify は Next.js ランタイムに関する情報を の Amazon CloudWatch Logs に送信します AWS アカウント。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 が IAM ロールを作成することを許可することを選択した場合、そのロールには CloudWatch ログを作成するアクセス許可が既に付与されています。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許可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、「サービスロールの追加」を参照してください。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 ホスティングを使用してサーバー側でレンダリングされたアプリをデプロイしますを参照してください。

アクセスログ

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。

Amplify は、顧客間で 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 はディス CloudFront トリビューションを事前に作成するため、 CloudFront新しいアプリケーションをデプロイするときに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状況と一致しません。

アプリケーションのアクセスログを取得するには、次の手順に従います。

アクセスログを表示するには

1. にサインイン AWS Management Console 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、ホスティング を選択し、モニタリング を選択します。
4. [監視] ページで [アクセスログ] を選択します。
5. [時間範囲の編集] を選択します。
6. 時間範囲の編集ウィンドウで、次の操作を行います。
   a. 開始日 には、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。
7. 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。step-by-step こちら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;
```

ビルドの E メール通知

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 メール通知を設定する

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 Management Console 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、通知の構築 を選択します。「通知のビルド」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、新しい を追加 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] には [すべてのブランチ] を選択します。
6. [保存] を選択します。

カスタムビルドイメージとライブパッケージのアップデート

トピック

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。

> **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、「既存のコードを使い始める」を参照してください。

カスタムビルドイメージの要件

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。

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。

カスタムビルドイメージの設定

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 Management Console 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」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。

ライブパッケージのアップデート

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のバージョンと依存関係を指定できます。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の設定

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 Management Console 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更新」リストで、「新しい を追加」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。

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、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。このガイドでは、アカウント管理アクセス許可を持ち、Amplify アプリケーションがバックエンドのデプロイ、作成、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する Amplify サービスロールを作成する方法について説明します。

サービスロールの作成

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 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] を選択します。
4. 名前、表示、作成 ページで、ロール名に などのわかりやすい名前を入力しますす**AmplifyConsoleServiceRole-AmplifyRole**。
5. すべてのデフォルトを受け入れ、ロールの作成 を選択します。
6. Amplify コンソールに戻り、アプリにロールをアタッチします。
   - 新しいアプリケーションをデプロイ中である場合
     a. サービスロールのリストを更新します。
     b. 作成したロールを選択します。この例では、 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ケーションのデプロイを完了します。
   - 既存のアプリがある場合
     a. ナビゲーションペインで、アプリ設定 、一般設定 を選択します。
     b. 全般設定ページで、編集 を選択します。
     c. 一般設定の編集ページで、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. Amplify コンソールに、アプリケーション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付与されました。

混乱した代理の防止

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。例のリージョンとアプリケーション ID をユーザー自身の情報に置き換えます。

```
"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 Management Console、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。

アプリパフォーマンスの管理

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。

アプリケーション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) エッジにコンテンツをキャッシュする間隔を長くすることで、HTTP Cache-Controlヘッダーを手動で設定してホスティングパフォーマンスを最適化できます。

ヘッダーを使用したキャッシュ保持期間の制御

HTTP Cache-Controlヘッダーの max-ageおよび s-maxageディレクティブは、アプリケーション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 でホストされているアプリは、オリジンによって送信される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Cache-Controlカスタムヘッダーを適用します。これにより、エラーレスポンスがキャッシュされ、同じリクエストを行う他のユーザーに提供されるのを防ぐことができます。

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。

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、「カスタムキャッシュコントロールヘッダー」を参照してください。

Cache-Control ヘッダーを設定してアプリケーションのパフォーマンスを向上させる

次の手順を使用して、コンテンツを CDN エッジに 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 Management Console 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、次のようにカスタムヘッダーの情報を入力します。

   a. にはpattern、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. アプリケーションを再デプロイして、新しいカスタムヘッダーを適用します。

AWS CloudTrailを使用した Amplify API コールのログ記録

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。 は、Amplify のすべての API コールをイベントとして CloudTrail 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。証跡を作成する場合は、Amplify の CloudTrailイベントなど、Amazon S3 バケットへのイベントの継続的な配信を有効にすることができます。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。が CloudTrail 収集する情報を使用して、Amplify に対して行われたリクエスト、リクエスト元の IP アドレス、リクエスト者、リクエスト日時などの詳細を確認できます。

の詳細については CloudTrail、「 AWS CloudTrail ユーザーガイド」を参照してください。

の Amplify 情報 CloudTrail

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 CloudTrail イベント履歴 の他の AWS サービスイベントとともにイベントに記録されます。 AWS アカウントで最近のイベントを表示、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「イベント履歴を含む CloudTrail 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 CloudTrail 、 はログファイルを 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 ログで収集されたイベントデータをさらに分析し、それに基づいて行動するように他の AWS サービスを設定できます。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 がサポートするサービスと統合
- の Amazon SNS 通知の設定 CloudTrail
- 複数のリージョンからの CloudTrail ログファイルの受信と複数のアカウントからの CloudTrail ログファイルの受信

すべての Amplify オペレーションは によって ログに記録 CloudTrail され、AWS Amplify コンソール API リファレンス 、AWS Amplify Admin UI API リファレンス 、および Amplify UI Builder API リファレンス に記載されています。例えば、、、 `DeleteBackendEnvironment`オペレーションを呼び出す`DeleteApp`と`CreateApp`、 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、ルートまたは 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。
- 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 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

証跡は、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できるようにする設定です。 CloudTrail ログファイルには 1 つ以上のログエントリが含まれます。イベントは任意のソース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 コール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 のセキュリティ

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 とユーザーの間で共有される責任です。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 — 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責任 AWS は AWS にあります。 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。コンプライアンスAWS プログラムコンプライアンスプログラム の一環として、サードパーティーの監査者は定期的にセキュリティの有効性をテストおよび検証。に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、お客様は、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 の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

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 AWS のサービス 使用できる です。

トピック

対象者

AWS Identity and Access Management (IAM) の使用方法は、Amplify で行う作業によって異なります。

サービスユーザー – 業務を行うために Amplify サービスを使用する場合は、管理者から必要な認証情報と許可が提供されます。業務のために使用する Amplify 機能が増えるにつれて、追加の許可が必要になる可能性があります。アクセスの管理方法を理解しておくと、管理者に適切な許可をリクエストするうえで役立ちます。Amplify の機能にアクセスできない場合は、「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」を参照してください。

サービス管理者 - 社内の Amplify リソースを担当している場合は、通常、Amplify へのフルアクセスがあります。サービスのユーザーがどの Amplify 機能やリソースにアクセスするかを決めるのは、管理者の仕事です。その後、IAM 管理者にリクエストを送信して、サービスユーザーの権限を変更する必要があります。このページの情報を点検して、IAM の基本概念を理解してください。会社で Amplify と IAM を併用する方法の詳細については、「Amplify が IAM で機能する仕組み」を参照してください。

IAM 管理者 – IAM 管理者には、Amplify へのアクセスを管理するポリシーの作成方法の詳細を理解することが推奨されます。IAM で使用できる 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。

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。として、IAM ユーザーとして AWS アカウントのルートユーザー、または IAM ロールを引き受けて認証（ にサインイン AWS) される必要があります。

ID ソースを介して提供された認証情報を使用して、フェデレーティッド ID AWS として にサインインできます。 AWS IAM Identity Center（ IAM Identity Center) ユーザー、会社のシングルサインオン認証、Google または Facebook の認証情報は、フェデレーション ID の例です。フェデレーティッド ID としてサインインする場合、IAM ロールを使用して、前もって管理者により ID フェデレーションが設定されています。フェデレーション AWS を使用して にアクセスすると、間接的にロールを引き受けることになります。

ユーザーのタイプに応じて、 AWS Management Console または AWS アクセスポータルにサインインできます。へのサインインの詳細については AWS、「 ユーザーガイド」の「 にサインインする方法 AWS アカウントAWS サインイン 」を参照してください。

AWS プログラムで にアクセスする場合、 は Software Development Kit (SDK) とコマンドラインインターフェイス (CLI) AWS を提供し、認証情報を使用してリクエストに暗号で署名します。 AWS ツールを使用しない場合は、リクエストに自分で署名する必要があります。推奨される方法を使用してリクエストを自分で署名する方法の詳細については、IAM ユーザーガイドの API AWS リクエストの署名を参照してください。

使用する認証方法を問わず、追加セキュリティ情報の提供をリクエストされる場合もあります。例えば、 AWS では、多要素認証 (MFA) を使用してアカウントのセキュリティを向上させることをお勧めします。詳細については、「AWS IAM Identity Center ユーザーガイド」の「Multi-factor authentication」(多要素認証) および「IAM ユーザーガイド」の「AWSでの多要素認証 (MFA) の使用」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、アカウント内のすべての およびリソースへの AWS のサービス 完全なアクセス権を持つ 1 つのサインインアイデンティティから始めます。この ID は AWS アカウント ルートユーザーと呼ばれ、アカウントの作成に使用した E メールアドレスとパスワードでサインインすることでアクセスできます。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。ルートユーザーの認証情報は保護し、ルートユーザーでしか実行できないタスクを実行するときに使用します。ルートユーザーとしてサインインする必要があるタスクの完全なリストについては、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。

フェデレーティッドアイデンティティ

ベストプラクティスとして、管理者アクセスを必要とするユーザーを含む人間のユーザーに、一時的な認証情報を使用して にアクセスするための ID プロバイダーとのフェデレーションの使用を要求 AWS のサービス します。

フェデレーティッド ID は、エンタープライズユーザーディレクトリ、ウェブ ID プロバイダー、AWS Directory Service、アイデンティティセンターディレクトリのユーザー、または ID ソースを通じて提供された認証情報 AWS のサービス を使用して にアクセスするユーザーです。フェデレーティッド ID が にアクセスすると AWS アカウント、ロールを引き受け、ロールは一時的な認証情報を提供します。

アクセスを一元管理する場合は、 AWS IAM Identity Centerを使用することをお勧めします。IAM Identity Center でユーザーとグループを作成することも、独自の ID ソース内のユーザーとグループのセットに接続して同期して、すべての AWS アカウント とアプリケーションで使用することもできます。IAM Identity Center の詳細については、「AWS IAM Identity Center ユーザーガイド」の「What is IAM Identity Center?」(IAM Identity Center 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、単一のユーザーまたはアプリケーションに対して特定のアクセス許可 AWS アカウント を持つ 内のアイデンティティです。可能であれば、パスワードやアクセスキーなどの長期的な認証情報を保有する IAM ユーザーを作成する代わりに、一時的な認証情報を使用することをお勧めします。ただし、IAM ユーザーでの長期的な認証情報が必要な特定のユースケースがある場合は、アクセスキーをローテーションすることをお勧めします。詳細については、IAM ユーザーガイドの長期的な認証情報を必要とするユースケースのためにアクセスキーを定期的にローテーションするを参照してください。

IAM グループは、IAM ユーザーの集団を指定するアイデンティティです。グループとしてサインインすることはできません。グループを使用して、複数のユーザーに対して一度に権限を指定できます。多数のユーザーグループがある場合、グループを使用することで権限の管理が容易になります。例えば、IAMAdminsという名前のグループを設定して、そのグループにIAM リソースを管理する許可を与えることができます。

ユーザーは、ロールとは異なります。ユーザーは 1 人の人または 1 つのアプリケーションに一意に関連付けられますが、ロールはそれを必要とする任意の人が引き受けるようになっています。ユーザーには永続的な長期の認証情報がありますが、ロールでは一時的な認証情報が提供されます。詳細については、「IAM ユーザーガイド」の「IAM ユーザー (ロールではなく) の作成が適している場合」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 AWS アカウント を持つ 内のアイデンティティです。これは IAM ユーザーに似ていますが、特定のユーザーには関連付けられていません。ロール を切り替える AWS Management Console ことで、 で IAM ロールを一時的に引き受けることができます。ロールを引き受けるには、 または AWS API AWS CLI オペレーションを呼び出すか、カスタム URL を使用します。ロールを使用する方法の詳細については、「IAM ユーザーガイド」の「IAM ロールの使用」を参照してください。

IAM ロールと一時的な認証情報は、次の状況で役立ちます:

- フェデレーションユーザーアクセス – フェデレーティッド ID に許可を割り当てるには、ロールを作成してそのロールの許可を定義します。フェデレーティッド ID が認証されると、その ID はロールに関連付けられ、ロールで定義されている許可が付与されます。フェデレーションの詳細については、「IAM ユーザーガイド」の「Creating a role for a third-party Identity Provider」(サードパーティーアイデンティティプロバイダー向けロールの作成) を参照してください。IAM Identity Center を使用する場合は、許可セットを設定します。アイデンティティが認証後にアクセスできるものを制御するため、IAM Identity Center は、権限セットを IAM のロールに関連付けます。アクセス許可セットの詳細については、「AWS IAM Identity Center ユーザーガイド」の「アクセス許可セット」を参照してください。
- 一時的な IAM ユーザー権限 - IAM ユーザーまたはロールは、特定のタスクに対して複数の異なる権限を一時的に IAM ロールで引き受けることができます。
- クロスアカウントアクセス - IAM ロールを使用して、自分のアカウントのリソースにアクセスすることを、別のアカウントの人物 (信頼済みプリンシパル) に許可できます。クロスアカウントアクセス権を付与する主な方法は、ロールを使用することです。ただし、一部の では AWS のサービス、(ロールをプロキシとして使用する代わりに) ポリシーをリソースに直接アタッチできます。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。
- クロスサービスアクセス — 一部の は、他の の機能 AWS のサービス を使用します AWS のサービス。例えば、あるサービスで呼び出しを行うと、通常そのサービスによって Amazon EC2 でアプリケーションが実行されたり、Amazon S3 にオブジェクトが保存されたりします。サービスでは、呼び出し元プリンシパルの許可、サービスロール、またはサービスリンクロールを使用してこれを行う場合があります。
  - 転送アクセスセッション (FAS) – IAM ユーザーまたはロールを使用して でアクションを実行する場合 AWS、ユーザーはプリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがありま

す。FAS は、 を呼び出すプリンシパルのアクセス許可を AWS のサービス、ダウンストリームサービス AWS のサービス へのリクエストのリクエストと組み合わせて使用します。FAS リクエストは、サービスが他の AWS のサービス またはリソースとのやり取りを完了する必要があるリクエストを受け取った場合にのみ行われます。この場合、両方のアクションを実行するためのアクセス許可が必要です。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。

- サービスロール - 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 のサービスにアクセス許可を委任するロールの作成」を参照してください。
- サービスにリンクされたロール – 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、サービスにリンクされた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。

- Amazon EC2 で実行されているアプリケーション – IAM ロールを使用して、EC2 インスタンスで実行され、 AWS CLI または AWS API リクエストを行うアプリケーションの一時的な認証情報を管理できます。これは、EC2 インスタンス内でのアクセスキーの保存に推奨されます。 AWS ロールを EC2 インスタンスに割り当て、そのすべてのアプリケーションで使用できるようにするには、インスタンスにアタッチされたインスタンスプロファイルを作成します。インスタンスプロファイルにはロールが含まれ、EC2 インスタンスで実行されるプログラムは一時的な認証情報を取得できます。詳細については、IAM ユーザーガイドのAmazon EC2 インスタンスで実行されるアプリケーションに IAM ロールを使用して許可を付与するを参照してください。

IAM ロールと IAM ユーザーのどちらを使用するかについては、IAM ユーザーガイドの(IAM ユーザーではなく) IAM ロールをいつ作成したら良いのか?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、 AWS ID またはリソースにアタッチします。ポリシーは AWS 、アイデンティティまたはリソースに関連付けられているときにアクセス許可を定義する のオブジェクトです。 は、プリンシパル (ユーザー、ルートユーザー、またはロールセッション) 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。ポリシーでの権限により、リクエストが許可されるか拒否されるかが決まります。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構造と内容の詳細については、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 AWS JSON 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んなリソースにどんな条件でアクションを実行できるかということです。

デフォルトでは、ユーザーやロールに権限はありません。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。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロールを引き継ぐことができます。

IAM ポリシーは、オペレーションの実行方法を問わず、アクションの許可を定義します。例えば、`iam:GetRole`アクションを許可するポリシーがあるとします。そのポリシーを持つユーザーは、 AWS Management Console、、 AWS CLIまたは AWS API からロール情報を取得できます。

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ポリシーは、IAM ユーザーグループ、ユーザーのグループ、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 ポリシーの作成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、さらにインラインポリシーまたはマネージドポリシーに分類できます。インラインポリシーは、単一のユーザー、グループ、またはロールに直接埋め込まれています。管理ポリシーは、 内の複数のユーザー、グループ、ロールにアタッチできるスタンドアロンポリシーです AWS アカウント。管理ポリシーには、 AWS 管理ポリシーとカスタマー管理ポリシーが含まれます。マネージドポリシーまたは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 ユーザーガイドのマネージドポリシーとインラインポリシーの比較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。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制御できます。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は、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。

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。

アクセスコントロールリスト (ACL)

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。

Amazon S3、 AWS WAF、および Amazon VPC は、ACLs。ACL の詳細については、Amazon Simple Storage Service デベロッパーガイドのアクセスコントロールリスト (ACL) の概要を参照してください。

その他のポリシータイプ

AWS は、一般的ではない追加のポリシータイプをサポートします。これらのポリシータイプでは、より一般的なポリシータイプで付与された最大の権限を設定できます。

- アクセス許可の境界 - アクセス許可の境界は、アイデンティティベースのポリシーによって IAM エンティティ (IAM ユーザーまたはロール) に付与できる権限の上限を設定する高度な機能です。エンティティにアクセス許可の境界を設定できます。結果として得られる権限は、エンティティのアイデンティティベースポリシーとそのアクセス許可の境界の共通部分になります。`Principal` フィールドでユーザーまたはロールを指定するリソースベースのポリシーでは、アクセス許可の境界は制限されません。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。アクセス許可の境界の詳細については、IAM ユーザーガイドのIAM エンティティのアクセス許可の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 (SCPs) – SCPs は、 の組織または組織単位 (OU) に対する最大アクセス許可を指定する JSON ポリシーです AWS Organizations。 AWS Organizations は、 AWS アカウント ビジネスが所有する複数の をグループ化して一元管理するサービスです。組織内のすべての機能を有効にすると、サービスコントロールポリシー (SCP) を一部またはすべてのアカウントに適用できます。SCP は、各 を含むメンバーアカウントのエンティティのアクセス許可を制限します AWS アカウントのルートユーザー。Organizations と SCP の詳細については、AWS Organizations ユーザーガイドの「SCP の仕組み」を参照してください。
- セッションポリシー - セッションポリシーは、ロールまたはフェデレーションユーザーの一時的なセッションをプログラムで作成する際にパラメータとして渡す高度なポリシーです。結果としてセッションの権限は、ユーザーまたはロールのアイデンティティベースポリシーとセッションポリシーの共通部分になります。また、リソースベースのポリシーから権限が派生する場合もあります。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。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権限を理解するのがさらに難しくなります。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	Yes
ポリシー条件キー	Yes
ACL	なし
ABAC (ポリシー内のタグ)	部分的
一時的な認証情報	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	なし

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を把握するには、「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ポリシーをサポートする	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 ポリシーの作成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。プリンシパルは、それが添付されているユーザーまたはロールに適用されるため、アイデンティティベースのポリシーでは指定できません。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制御できます。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は、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、アカウント全体、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。リソースベースのポリシーにクロスアカウントのプリンシパルを追加しても、信頼関係は半分しか確立されない点に注意してください。プリンシパルとリソースが異なる にある場合 AWS アカウント、信頼されたアカウントの IAM 管理者は、プリンシパルエンティティ (ユーザーまたはロール) にリソースへのアクセス許可も付与する必要があります。IAM 管理者は、アイデンティティベースのポリシーをエンティティにアタッチすることで権限を付与します。ただし、リソースベースのポリシーで、同じアカウントのプリンシパルへのアクセス権が付与されている場合は、アイデンティティベースのポリシーをさらに付与する必要はありません。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に対するサポート	あり

管理者は AWS JSON 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んなリソースにどんな条件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。ポリシーアクションの名前は通常、関連付けられた AWS API オペレーションと同じです。一致する API オペレーションのない許可のみのアクションなど、いくつかの例外があります。また、ポリシーに複数のアクションが必要なオペレーションもあります。これらの追加アクションは、依存アクションと呼ばれます。

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権限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

ポリシーリソースに対するサポート	あり

管理者は AWS JSON 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どのような条件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。ステートメントには、`Resource` または `NotResource` 要素を含める必要があります。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。これは、リソースレベルの許可と呼ばれる特定のリソースタイプをサポートするアクションに対して実行できます。

オペレーションのリスト化など、リソースレベルの権限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 AWS JSON 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んなリソースにどんな条件でアクションを実行できるかということです。

Condition 要素 (または Condition ブロック) を使用すると、ステートメントが有効な条件を指定できます。Condition 要素はオプションです。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することで、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。

1 つのステートメントに複数の Condition 要素を指定する場合、または 1 つの Condition 要素に複数のキーを指定する場合、 AWS では AND 論理演算子を使用してそれらを評価します。1 つの条件キーに複数の値を指定すると、 は論理ORオペレーションを使用して条件 AWS を評価します。ステートメントの権限が付与される前にすべての条件が満たされる必要があります。

条件を指定する際にプレースホルダー変数も使用できます。例えば IAM ユーザーに、IAM ユーザー名がタグ付けされている場合のみリソースにアクセスできる権限を付与することができます。詳細については、IAM ユーザーガイドのIAM ポリシーの要素: 変数およびタグを参照してください。

AWS は、グローバル条件キーとサービス固有の条件キーをサポートします。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許可を持つかをコントロールします。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	部分的

属性ベースのアクセス制御 (ABAC) は、属性に基づいてアクセス許可を定義する認可戦略です。では AWS、これらの属性はタグ と呼ばれます。タグは、IAM エンティティ (ユーザーまたはロール) および多くの AWS リソースにアタッチできます。エンティティとリソースのタグ付けは、ABAC の最初の手順です。その後、プリンシパルのタグがアクセスしようとしているリソースのタグと一致した場合にオペレーションを許可するように ABAC ポリシーをします。

ABAC は、急成長する環境やポリシー管理が煩雑になる状況で役立ちます。

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 とは?を参照してください。ABAC をセットアップするステップ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制御 (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

一部の は、一時的な認証情報を使用してサインインすると機能 AWS のサービス しません。一時的な認証情報 AWS のサービス を使用する などの詳細については、IAM ユーザーガイドのAWS のサービス「IAM と連携する 」を参照してください。

ユーザー名とパスワード以外の AWS Management Console 方法で にサインインする場合、一時的な認証情報を使用します。例えば、会社の Single Sign-On (SSO) リンク AWS を使用して にアクセスすると、そのプロセスによって一時的な認証情報が自動的に作成されます。また、ユーザーとしてコンソールにサインインしてからロールを切り替える場合も、一時的な認証情報が自動的に作成されます。ロールの切り替えに関する詳細については、IAM ユーザーガイドのロールへの切り替え (コンソール)を参照してください。

一時的な認証情報は、 AWS CLI または AWS API を使用して手動で作成できます。その後、これらの一時的な認証情報を使用して . AWS recommends にアクセスできます AWS。これは、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することを推奨しています。詳細については、IAM の一時的セキュリティ認証情報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) をサポート	あり

IAM ユーザーまたはロールを使用して でアクションを実行すると AWS、プリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがあります。FAS は、 を呼び出すプリンシパルのアクセス許可を AWS のサービス、ダウンストリームサービス AWS のサービス へのリクエストのリクエストと組み合わせて使用します。FAS リクエストは、サービスが他の AWS のサービス またはリソースとのやり取りを完了する必要があるリクエストを受け取った場合にのみ行われます。この場合、両方のアクションを実行するためのアクセス許可が必要です。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

サービスロールに対するサポート	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 のサービスにアクセス許可を委任するロールの作成」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、サービスにリンクされた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。また、、 AWS Command Line Interface （ AWS CLI ） AWS Management Console、または AWS API を使用してタスクを実行することはできません。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。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロールを引き受けることができます。

これらサンプルの 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 ポリシーの作成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

- ポリシーの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

ポリシーの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を実行すると、 AWS アカウントに料金が発生する可能性があります。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を付与するには、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して、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。例えば、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、 などの特定の を介してサービスアクションが使用される場合に 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 AWS CloudFormation。詳細については、「IAM ユーザーガイド」の [IAM JSON policy elements: Condition] (IAM JSON ポリシー要素:条件) を参照してください。
- IAM Access Analyzer 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 Access Analyzer 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 ポリシーの検証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、セキュリティを強化するために MFA を有効にします。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 保護 API アクセスの設定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、削除アクションを実行する必要があるユーザーに適切なアクセス許可を決定します。

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、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るため、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。ユースケース別にカスタマーマネージドポリシーを定義して、マネージドポリシー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 管理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) が更新されます。 AWS のサービス は、新しい が起動されたとき、または既存のサービスで新しい API AWS オペレーションが使用可能になったときに、 AWS 管理ポリシーを更新する可能性が最も高くなります。

詳細については、「IAM ユーザーガイド」の「AWS 管理ポリシー」を参照してください。

AWS マネージドポリシー: AdministratorAccess-Amplify

`AdministratorAccess-Amplify` ポリシーは IAM ID 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ときは、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。

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 AWS Identity and Access Management 、任意のアクセス許可を持つ他の ID 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、「 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

AWS 管理ポリシー： AmplifyBackendDeployFullAccess

AmplifyBackendDeployFullAccess ポリシーは IAM ID 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 ロールに延期されます。

許可の詳細

このポリシーには、次の を実行する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。
- AWS CloudFormation– Amplify マネージドスタックを作成、更新、削除します。
- SSM– Amplify マネージド SSM パラメータストアと SecureStringパラメータを作成、更新String、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。目的は、Gen 2 サンドボックス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。目的は、Gen 2 サンドボックスホットスワップ機能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`— AWS Cloud Development Kit (AWS CDK) CLI がデプロイロールを引き受けることを可能にします。
- `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込みます。
- `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。

このポリシーのアクセス許可を確認するには、「 管理ポリシーリファレンスAmplifyBackendDeployFullAccess」の「」を参照してください。 AWS

Amplify の AWS マネージドポリシーの更新

Amplify の AWS マネージドポリシーの更新に関する詳細を、このサービスがこれらの変更の追跡を開始した以降の分について表示します。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 - 既存ポリシーへの更新	`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 がお客様のアカウントの 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
AmplifyBackendDeployFullAccess – 既存ポリシーへの更新	Amazon RDS と Amazon EC2 の読み取り専用アクセス許可を持つ新しい`AmplifyDiscoverRDSVpcConfig` ポリシーステートメントを、リソース条件とアカウント条件の両方によって範囲指定して追加します。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	2024 年 4 月 17 日

変更	説明	日付
	`npx amplify generate schema-from-database` コマンドをサポートします。 `rds:DescribeDBProxies` 、`rds:DescribeDBInstances` 、、`rds:DescribeDBClusters` 、`rds:DescribeDBSubnetGroups` および アクセス`ec2:DescribeSubnets` 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。	

変更	説明	日付
AmplifyBackendDeployFullAccess – 既存ポリシーへの更新	`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。 `lambda:GetFunction` ポリシーアクションを追加して、ホットスワップ関数をサポートします。 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 – 既存のポリシーの更新	AWS 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 – 既存ポリシーへの更新	`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 関数に直接呼び出して、Amazon S3 アセットのホットスワップを実行します。 `lambda:UpdateFunctionCode` ポリシーアクションを追加して、ホットスワップ関数をサポートします。	2024 年 1 月 2 日
AmplifyBackendDeployFullAccess – 既存ポリシーへの更新	`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 – 既存ポリシーへの更新	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 – 既存のポリシーの更新	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 – 既存のポリシーの更新	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。 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。 OpenSearch 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。 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。 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。 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。 `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 AWS CloudFormation スタックの取得をサポートし、スタックが更新された場合に実行が重複しないようにします。 `cloudformation:ListStacks` 権限を `CLICloudformationPolicy` ステートメントブロックに追加します。このアクセス許可は、 CloudFormation DescribeStacks 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 – 既存のポリシーの更新	ポリシーアクションを追加して、Amplify のサーバー側のレンダリング機能がお客様の CloudWatch でアプリケーションメトリクスを にプッシュできるようにします AWS アカウント。	2022 年 8 月 30 日
AdministratorAccess-Amplify – 既存のポリシーの更新	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 – 既存のポリシーの更新	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。 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 – 既存のポリシーの更新	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 – 既存のポリシーの更新	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。 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。 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 – 既存のポリシーの更新	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。 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。 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。 をサポートする CloudFormation アクションを追加します AWS CloudFormation StackSets。 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。 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。 CloudWatch イベントをサポートする CloudWatch ログアクションを追加します。 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。	2021 年 9 月 27 日

変更	説明	日付
	サーバー側レンダリング (SSR) アプリをサポートするポリシーアクションを追加します。	

変更	説明	日付
AdministratorAccess-Amplify – 既存のポリシーの更新	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。 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。 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。 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。 Amplify Studio: Amazon Cognito 、 AWS Lambda、IAM、および AWS 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。 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。 Amplify アプリケーションの Lambda レイヤーをサポートするアクションを追加し	2021 年 7 月 28 日

変更	説明	日付
	ます AWS CloudFormation `ListResources` 。	
Amplify が変更の追跡を開始しました	Amplify が AWS マネージド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 を実行する権限がありません。PassRole
- AWS 自分のアカウント以外のユーザーに 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

「I am not authorized to perform an action in Amazon Bedrock」というエラーが表示された場合、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。サインイン認証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 を実行する権限がありません。PassRole

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、そのサービスに既存のロールを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権限が必要です。

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権限が必要です。メアリーには、ロールをサービスに渡す許可がありません。

```
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。サインイン認証情報を提供した担当者が管理者です。

AWS 自分のアカウント以外のユーザーに 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でのロールとリソースベースのポリシーの使用の違いについては、IAM ユーザーガイドの「IAM でのクロスアカウントリソースアクセス」を参照してください。

Amplify のデータ保護

AWS Amplify は、 AWS 責任共有モデル 。 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を負います。 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。 これには、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、AWS 顧客および APN パートナーが含まれます。 データコントローラーまたはデータ処理者として動作し、 は、 AWS クラウドに保存する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、または SDK を使用して Amplify AWS CLI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 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。Amazon S3

Amplify は、Amazon CloudFront を使用してアプリを顧客に供給します。 は、エッジロケーションの Point of Presence (POPs用に暗号化された SSDs と、リージョン別エッジキャッシュ (RECs) 用に暗号化された EBS ボリューム CloudFront を使用します。Functions の CloudFront 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 SSDs と、 で使用される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 CloudFront。 POPs

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、プライベート認証局によって管理される SHA-256 AWS Certificate Manager 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「ACM PCA 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である 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 AWS Amplify の一環として 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範囲内にあるかどうかを確認するには、コンプライアンスプログラムAWS のサービス による対象範囲内のコンプライアンスプログラムを参照し、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、「 でのレポートのダウンロード AWS Artifact」の」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。 では、コンプライアンスに役立つ以下のリソース AWS を提供しています。

- セキュリティとコンプライアンスのクイックスタートガイド – これらのデプロイガイドでは、アーキテクチャ上の考慮事項について説明し、セキュリティとコンプライアンスに重点を置いたベースライン環境 AWS を にデプロイする手順について説明します。
- アマゾン ウェブ サービスにおける HIPAA セキュリティとコンプライアンスのアーキテクチャ — このホワイトペーパーでは、企業が AWS を使用して HIPAA 対象アプリケーションを作成する方法について説明します。

> **Note**
>
> すべて AWS のサービス HIPAA の対象となるわけではありません。詳細については、HIPAA 対応サービスのリファレンスを参照してください。

- AWS コンプライアンスリソース – このワークブックとガイドのコレクションは、お客様の業界や地域に適用される場合があります。
- AWS カスタマーコンプライアンスガイド — コンプライアンスの観点から責任共有モデルを理解します。このガイドでは、ガイダンスを保護し AWS のサービス 、複数のフレームワーク (米国国立標準技術研究所 (NIST)、Payment Card Industry Security Standards Council (PCI)、国際標準化機構 (ISO) を含む) のセキュリティコントロールにマッピングするためのベストプラクティスをまとめています。
- 「 デベロッパーガイド」の「ルールによるリソースの評価」 – この AWS Config サービスは、リソース設定が社内プラクティス、業界ガイドライン、および規制にどの程度準拠しているかを評価します。 AWS Config
- AWS Security Hub – これにより AWS のサービス 、 内のセキュリティ状態を包括的に確認できます AWS。Security Hub では、セキュリティコントロールを使用して AWS リソースを評価し、セキュリティ業界標準とベストプラクティスに対するコンプライアンスをチェックします。サポートされているサービスとコントロールのリストについては、Security Hub のコントロールリファレンスを参照してください。
- Amazon GuardDuty – これにより AWS アカウント、疑わしいアクティビティや悪意のあるアクティビティがないか環境を監視することで、、、ワークロード、コンテナ、データに対する潜在的な脅威 AWS のサービス を検出します。 GuardDuty は、特定のコンプライアンスフレームワークで義務付けられている侵入検知要件を満たすことで、PCI DSS などのさまざまなコンプライアンス要件への対応に役立ちます。
- AWS Audit Manager – これにより AWS のサービス 、 AWS 使用状況を継続的に監査し、リスクの管理方法と規制や業界標準への準拠を簡素化できます。

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によって保護されています。 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 AWS 環境を設計するには、「 Security Pillar AWS Well-Architected Framework」の「Infrastructure Protection」を参照してください。

が AWS 公開した 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 は必須で TLS 1.3 がお勧めです。

- 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。

また、リクエストには、アクセスキー ID と、IAM プリンシパルに関連付けられているシークレットアクセスキーを使用して署名する必要があります。または、AWS Security Token Service (AWS STS) を使用して、一時セキュリティ認証情報を生成し、リクエストに署名することもできます。

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、およびパフォーマンスを維持する上で重要な部分です。 AWS 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を提供します。

- Amazon CloudWatch は、 AWS リソースと 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、 で Amazon Elastic Compute Cloud (Amazon EC2) インスタンスの CPU 使用率やその他のメトリクス CloudWatch 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、「」を参照してくださいモニタリング。
- Amazon CloudWatch Logs を使用すると、Amazon EC2 インスタンス、、およびその他のソースからログファイルをモニタリング、保存 AWS CloudTrail、およびアクセスできます。CloudWatch ログはログファイル内の情報をモニタリングし、特定のしきい値に達したときに通知できます。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できます。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。を呼び出したユーザーとアカウント AWS、呼び出し元のソース IP アドレス、呼び出しが発生した日時を特定できます。詳細については、「AWS CloudTrailを使用した Amplify API コールのログ記録」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ソースからのデータに簡単に接続できるサーバーレスイベントバスサービスです。 は、独自のアプリケーション、S oftware-as-aサービス (SaaS) アプリケーション、サービス AWS からリアルタイムデータのストリームを EventBridge 配信し、そのデータを などのターゲットにルーティングします AWS Lambda。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 ユーザーガイド EventBridge 」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ためのアクセス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の実行を強制できてしまう場合に生じる、セキュリティ上の問題です。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、アカウント内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、 が別のサービスに AWS Amplify 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。

```
{
  "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。

```
{
  "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

ホスティングのサービスクォータは以下のとおりです。 AWS Amplify 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。

New AWS アカウント では、アプリと同時ジョブの割り当てが減りました。 AWS これらのクォータは使用量に基づいて自動的に引き上げられます。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	はい	AWS 現在のリージョンのこのアカウントでAmplify Console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	No	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	No	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、AWS ビルド後にAmplify

名前	デフォルト	引き上げ可能	説明
			Console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	No	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	はい	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	はい	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	No	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	No	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	No	AWS 現在のリージョンのこのアカウントで1時間あたりにAmplify Consoleで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	サポートされている各リージョン: 20,000	はい	アプリの 1 秒あたりのリクエストトークンの最大数。Amplify Hosting は、消費するリソースの量 (処理時間とデータ転送) に基づいてトークンをリクエストに割り当てます。
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	No	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	はい	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 ホスティングの使用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ます)

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す場合、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時間を超えています。このアプリケーションの制限は、Amplify のサービスREQUEST_TOKENS_PER_SECOND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。

この問題を解決するには、アプリケーションを最適化してリクエスト期間を短縮し、データ転送を短縮してアプリケーションの 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、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、より高い RPS をサポートできます。

同様に、1 MB のレスポンスサイズを返すアプリケーションは、250 KB のレスポンス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、特定のレスポンスが CloudFront キャッシュに保持される時間を最大化するCache-Control ヘッダーを設定して、Amazon キャッシュを活用することをお勧めします。 CloudFront キャッシュから提供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非常に高くスケーリングできます。キャッシュの詳細については CloudFront、「Amazon CloudFront デベロッパーガイド」の「キャッシュと可用性の最適化」を参照してください。

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) ビルドイメージに関する問題のトラブルシューティングに役立ちます。

トピック

- Python ランタイムで Amplify 関数を実行する方法
- 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する方法

Python ランタイムで Amplify 関数を実行する方法

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、および 3.11 がプリインストールされています。

Amazon Linux 2 イメージとの下位互換性のために、AL2023 ビルドイメージには古いバージョンの Python 用のシンボリックリンクがプリインストールされています。したがって、Amplify ホスティングに関する GitHub よくある質問に記載されている手順を使用して、アプリケーションのビルド仕様のビルドコマンドを更新する必要はありません。

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する方法

Amazon Linux 2023 ビルドイメージを使用していて、スーパーユーザー権限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 コマンドを使用してこれらのsudoコマンドを実行する必要があります。例えば、 の実行中にエラーが発生した場合はyum install -y gcc、 を使用しますsudo yum install -y gcc。

Amazon Linux 2 ビルドイメージ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 コマンドを使用してsudo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を必要とするコマンドsudoには、 を使用するのがベストプラクティスです。

カスタムドメインのトラブルシューティング

カスタムドメインを Amplify アプリケーションに接続するときに問題が発生した場合は、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。

サーバー側で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

SSR アプリを Amplify にデプロイする際に問題が発生した場合は、SSR デプロイのトラブルシューティング「」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、AWS Amplify の詳細なリファレンス資料を検索します。

トピック

AWS CloudFormation サポート

AWS CloudFormation テンプレートを使って Amplify のプロビジョニングを行い、反復可能で信頼性の高いウェブアプリデプロイメントを実行できます。AWS CloudFormation 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を提供し、また複数の AWS アカウントやリージョンへの展開を簡素化 (数回クリックするのみ) できます。

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

AWS Command Line Interface を使用して、コマンドラインからプログラムで Amplify の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

AWS Command Line Interface を使用して Amplify リソース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

次の表に、 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ます AWS Amplify。

- ドキュメントの最終更新日: 2024 年 5 月 31 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
開始方法の章を更新しました	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ようにAmplify ホスティングの使用開始章を更新しました。	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	2024 年 4 月 4 日

変更	説明	日付
	ジドポリシー AWS Amplifyトピックを更新しました。	
トラブルシューティングに関する新しい章	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の修正方法を説明するAmplify ホスティングのトラブルシューティング章を追加しました。	2024 年 4 月 2 日
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 サポートについて説明するSSL/TLS 証明書の使用トピックを カスタムドメインのセットアップ章に追加しました。	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 SSR フレームワーク向けの Amplify サポート トピックを追加しました。	2023 年 11 月 19 日

変更	説明	日付
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
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	Amplify の新しい AmplifyBackendDeployFullAccess AWS マネージドポリシーについて説明するAWS の マネージドポリシー AWS Amplifyようにトピックを更新しました。	2023 年 10 月 8 日

変更	説明	日付
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のビルド設定トピックを更新しました。	2023 年 6 月 19 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 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 ホスティングを使用してサーバー側でレンダリングされたアプリをデプロイします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

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 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
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
新しい GitHub アプリ機能の起動	GitHub リポジトリへの Amplify アクセスを承認するための新しい GitHub アプリを説明する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

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 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
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について説明する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ためにこのAmplify ホスティングを使用してサーバー側でレンダリングされたアプリをデプロイします章を更新しました。	2021 年 7 月 22 日

変更	説明	日付
「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のビルド設定トピックを追加しました。	2021 年 7 月 20 日
「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 config の自動ビルド時間生成 (Gen 1 アプリケーションのみ) 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 アプリケーションのみ) 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 アプリのみ) トピックを追加しました。	2021 年 6 月 30 日

変更	説明	日付
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
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 Web アプリの Amplify サポートについて説明するAmplify ホスティングを使用してサーバー側でレンダリングされたアプリをデプロイします章を追加しました。	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

変更	説明	日付
モニタリングトピックを更新	Amazon CloudWatch メトリ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ように、「モニタリング」トピックを更新しました。	2021 年 2 月 2 日
新しい CloudTrail ログ記録トピック	コンソール API リファレンスと管理者 UI API リファレンスのすべての API アクションを がキャプチャしてログに記録する方法を説明する Amplify API コールのログ記録 AWS CloudTrailに関するトピックを追加しました。 AWS CloudTrail AWS Amplify AWS Amplify	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 Management Console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

変更	説明	日付
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
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

変更	説明	日付
カスタムドメインのトピックを更新	Amazon Route 53 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 GoDaddy、カスタムドメインのセットアップ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。