AWS Amplify

Console User Guide

Table of Contents

Welcome to the AWS Amplify Console

The AWS Amplify Console is the control center for fullstack web and mobile application deployments in AWS. Amplify Console provides two main services, hosting and the Admin UI. Amplify Console hosting provides a git-based workflow for hosting fullstack serverless web apps with continuous deployment. The Admin UI is a visual interface for frontend web and mobile developers to create and manage app backends outside the AWS Management Console.

A fullstack serverless web app consists of a backend built with cloud resources such as GraphQL or REST APIs, file and data storage, and a frontend built with a single-page application (SPA) framework such as React, Angular, Vue, or Gatsby. Amplify Console supports the common SPA frameworks, for example, React, Angular, Vue.js, Ionic, and Ember, as well as static site generators like Gatsby, Eleventy, Hugo, VuePress, and Jekyll. Amplify supports applications that use server-side rendering (SSR) that are created using the Next.js framework.

Amplify Console features

Hosting features

- Manage production and staging environments for your frontend and backend by connecting new branches. See feature branch deployments (p. 47).
- Connect your application to a custom domain. See Set up custom domains (p. 22).
- Deploy and host SSR web apps created using the Next.js. framework.
- Preview changes during code reviews by setting up pull request previews (p. 67).
- Improve your app quality with end to end tests. See End-to-end testing (p. 69).
- Password protect your web app so you can work on new features without making them publicly accessible. See Restricting access (p. 78).
- Set up rewrites and redirects to maintain SEO rankings and route traffic based on your client app requirements. See Using redirects (p. 72).
- Instant cache invalidations ensure your app is updated instantly on every code commit.
- Atomic deployments eliminate maintenance windows by ensuring that the web app is updated only after the entire deployment finishes. This eliminates scenarios where files fail to upload properly.
- Get screen shots of your app rendered on different mobile devices to identify layout issues.

Admin UI features

- Visual data modeling enables you to focus on your domain-specific objects instead of cloud infrastructure.
- Set up authentication for your app.
- Powerful and easy to understand authorization.
- Infrastructure-as-code configures all backend capabilities with AWS CloudFormation.
- Works with the Amplify Command Line Interface (CLI). All updates you make in the Admin UI can be pulled into the CLI.
- Invite users via email to configure and manage the backend. These users will also be able to log in to the Amplify CLI with their email.

- Content management with markdown support.
- Manage users and groups for your app.

Getting started

Getting started with hosting

To get started with Amplify Console's hosting features, see the Getting started with existing code (p. 3) tutorial. After completing the tutorial, you will be able to connect your git repository (GitHub, BitBucket Cloud, GitLab, and AWS CodeCommit) to set up continuous deployment. Alternatively, you can get started with one of the fullstack continuous deployment samples (p. 9).

Getting started with the Admin UI

You don't need an AWS account to get started using the Admin UI. Without an AWS account, you can begin modeling data for your backend locally. With an AWS account, you have an expanded set of features for managing your backend environment. For more information, see Getting started with Admin UI.

Modern SPA web applications

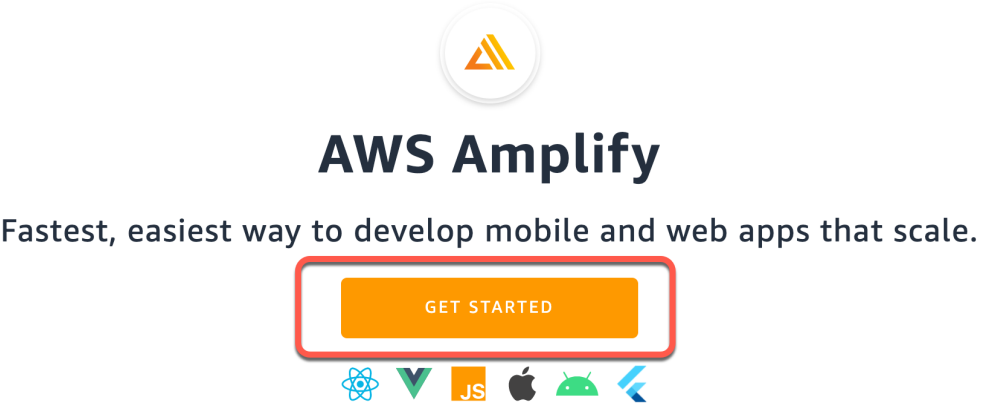
This user guide is intended for customers who have a basic understanding of modern single-page web applications (SPA). Modern web applications are constructed as SPAs that package all application components into static files. Traditional client-server web architectures led to poor experiences; every button click or search required a round trip to the server, re-rendering the entire application. Modern web apps offer a native app-like user experience by serving the app frontend, or user interface, efficiently to browsers as prebuilt HTML/JavaScript files that can then invoke backend functionality without reloading the page.

A modern web application's functionality is often spread across multiple places, such as databases, authentication services, frontend code running in the browser, and backend business logic, or AWS Lambda functions, running in the cloud. This makes application deployments complex and time-consuming as developers need to carefully coordinate deployments across the frontend and backend to avoid partial or failed deployments. The Amplify Console simplifies deployment of the frontend and backend in a single workflow.

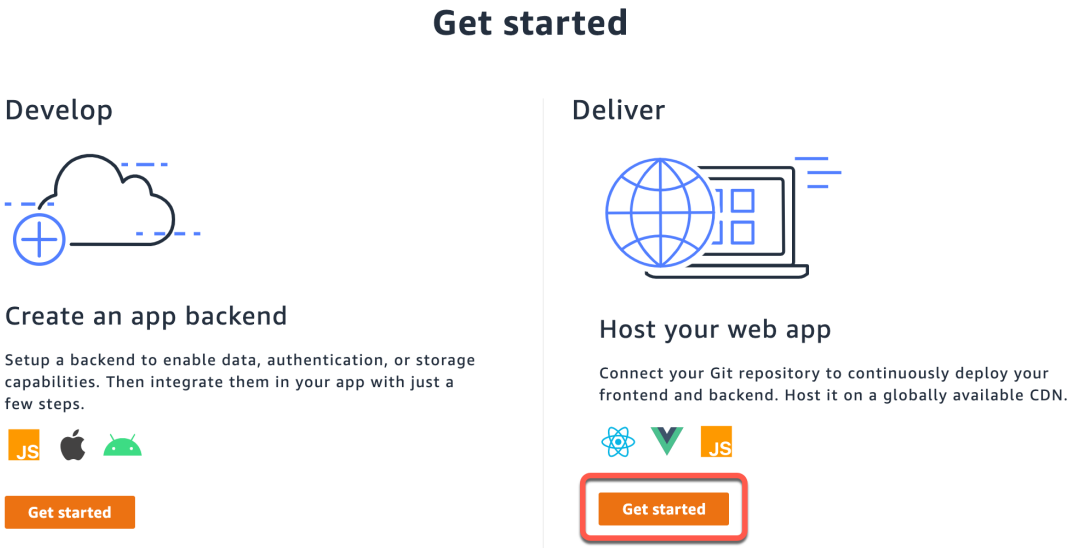
Getting started with existing code

In this walkthrough, you learn how to continuously build, deploy, and host a modern web app. Modern web apps include single-page application (SPA) frameworks (for example, React, Angular, or Vue) and static-site generators (SSGs) (for example, Hugo, Jekyll, or Gatsby). Amplify also supports web apps that use server-side rendering (SSR) and are created using Next.js.

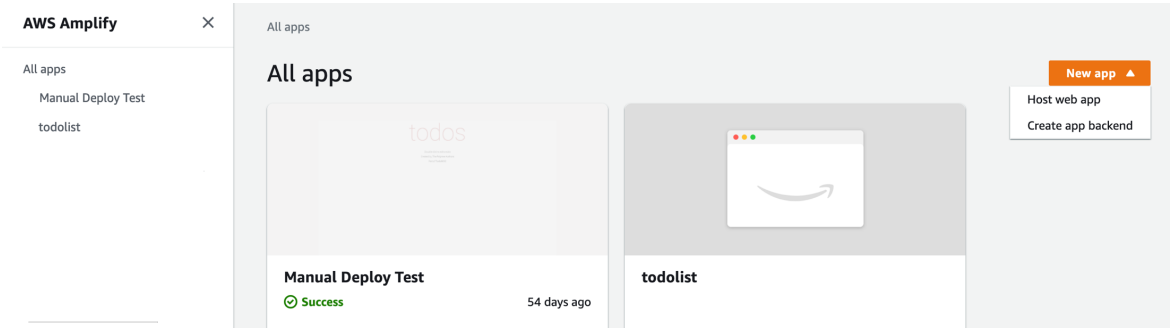
To get started, log in to the Amplify Console. If you are starting from the **AWS Amplify** home page, choose **Get Started** at the top of the page.



Then choose **Get started** under **Deliver**.

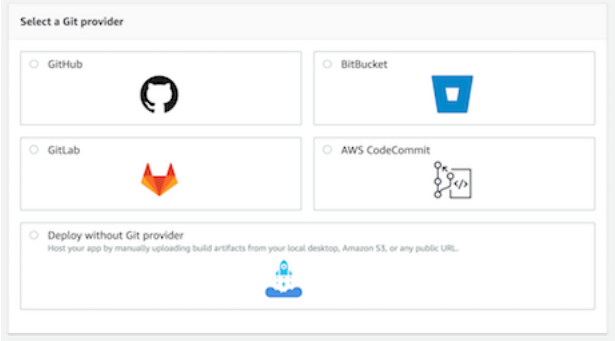


If you are starting from the **All apps** page, choose **New app**, **Host web app** in the upper right corner.

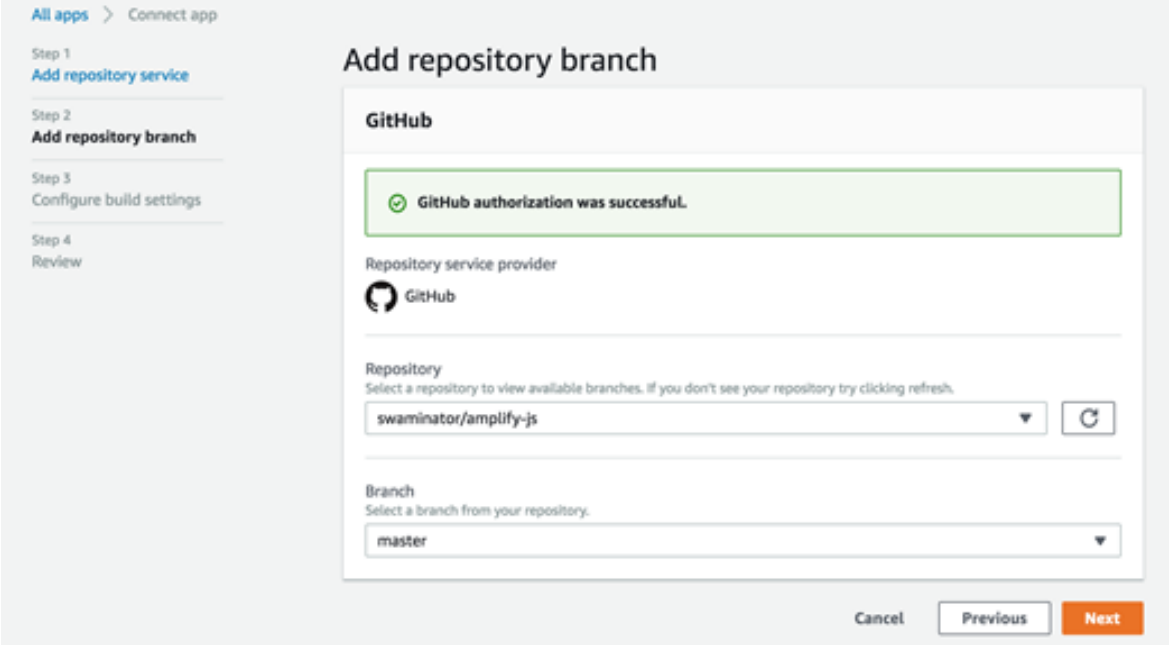


Step 1: Connect repository

Connect your GitHub, Bitbucket, GitLab, or AWS CodeCommit repositories. You also have the option of manually uploading your build artifacts without connecting a Git repository (see Manual Deploys (p. 63)). After you authorize the Amplify Console, Amplify fetches an access token from the repository provider, but it *doesn't store the token* on the AWS servers. Amplify accesses your repository using deploy keys installed in a specific repository only.



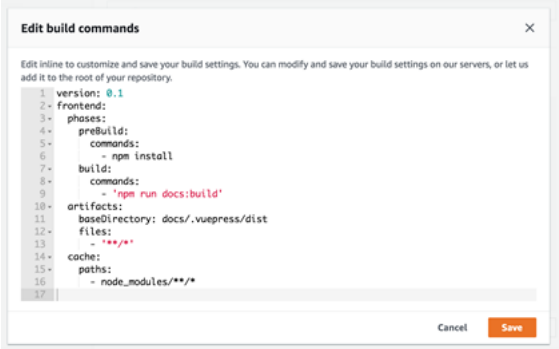
After you connect the repository service provider, choose a repository, and then choose a corresponding branch to build and deploy.



Step 2a: Confirm build settings for the front end

For the selected branch, Amplify inspects your repository to automatically detect the sequence of build commands to be executed.

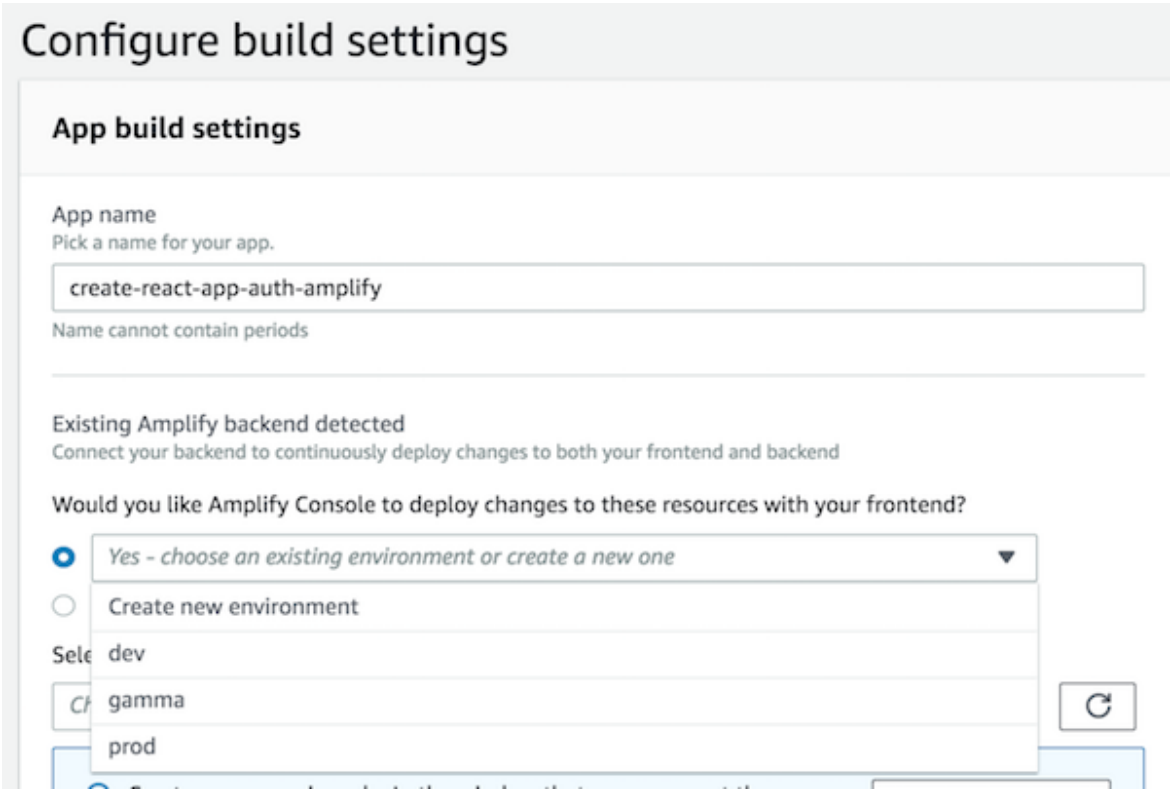
Important: Verify that the build commands and build output directory (that is, artifacts > baseDirectory) is accurate. If you need to modify this information, choose **Edit** to open the YML editor. You can save your build settings on our servers, or you can download the YML and add it to the root of your repo (for monorepos, store the YML at the app's root directory).



For more information, see YML structure (p. 36).

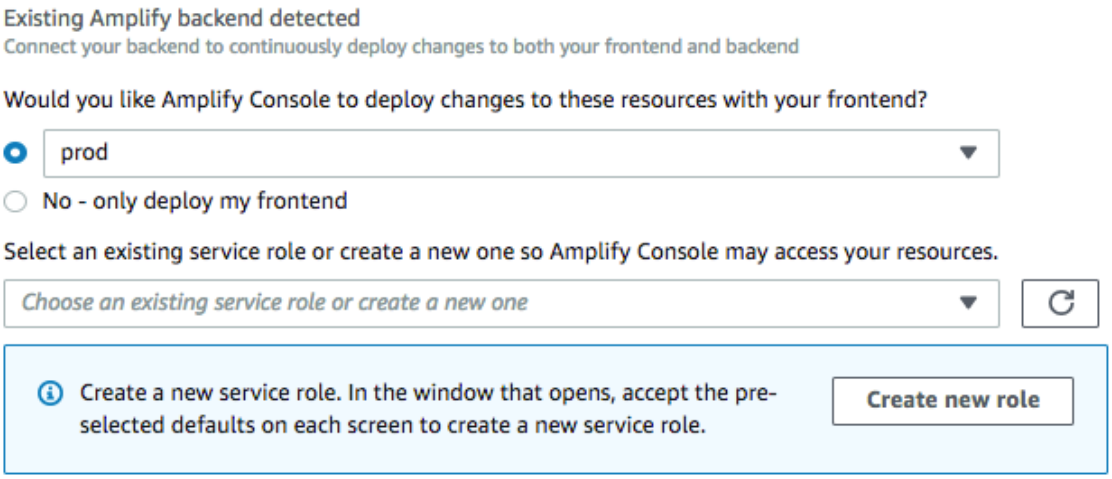
Step 2b: Confirm build settings for the backend

If you connected a repository provisioned by the Amplify CLI v1.0+ (run *amplify -v* to find CLI version), the Amplify Console will deploy or automatically update backend resources (any resource provisioned by the Amplify CLI) in a single workflow with the frontend build. You can choose to point an existing backend environment to your branch, or create a completely new environment. For a step-by-step tutorial, see Deploying a fullstack app (p. 9).



To deploy backend functionality using the Amplify CLI during your build, create or reuse an AWS Identity and Access Management (IAM) service role. IAM roles are a secure way to grant the Amplify Console permissions to act on resources in your account.

Note: The Amplify CLI won't run without an IAM service role enabled.



Step 2c: Add environment variables (optional)

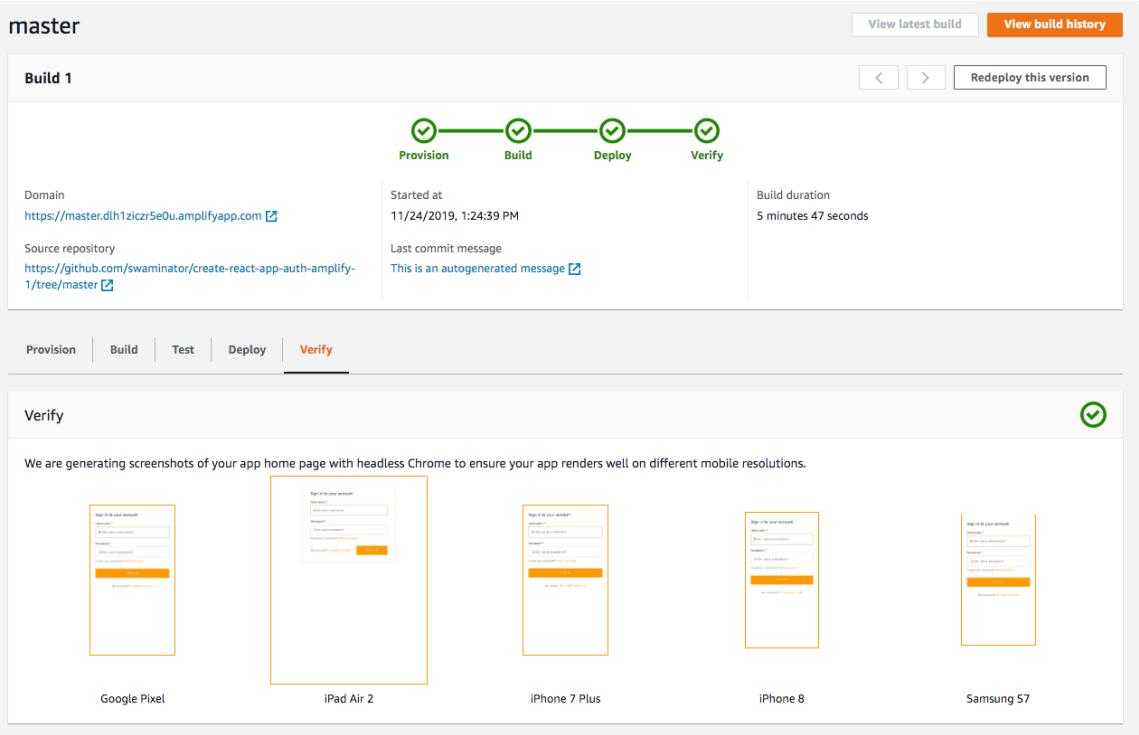
Almost every app needs to get configuration information at runtime. These configurations can be database connection details, API keys, or different parameters. Environment variables provide a means to expose these configurations at build time.

Step 3: Save and deploy

Review all of your settings to ensure everything is set up correctly. Choose **Save and deploy** to deploy your web app to a global content delivery network (CDN). Your front end build typically takes 1 to 2 minutes but can vary based on size of the app.

Access the build logs screen by selecting a progress indicator on the branch tile. A build has the following stages:

1. **Provision** - Your build environment is set up using a Docker image on a host with 4 vCPU, 7GB memory. Each build gets its own host instance, ensuring that all resources are securely isolated. The contents of the Docker file are displayed to ensure that the default image supports your requirements.
2. **Build** - The build phase consists of three stages: setup (clones repository into container), deploy backend (runs the Amplify CLI to deploy backend resources), and build front end (builds your front-end artifacts).
3. **Deploy** - When the build is complete, all artifacts are deployed to a hosting environment managed by Amplify. Every deployment is atomic - atomic deployments eliminate maintenance windows by ensuring that the web app is only updated after the entire deployment has completed.
4. **Verify** - To verify that your app works correctly, Amplify renders screen shots of the index.html in multiple device resolutions using Headless Chrome.



Next steps

- Add a custom domain to your app (p. 22)
- Manage multiple environments (p. 47)
- Preview pull requests before merging (p. 67)

Getting started with fullstack continuous deployments

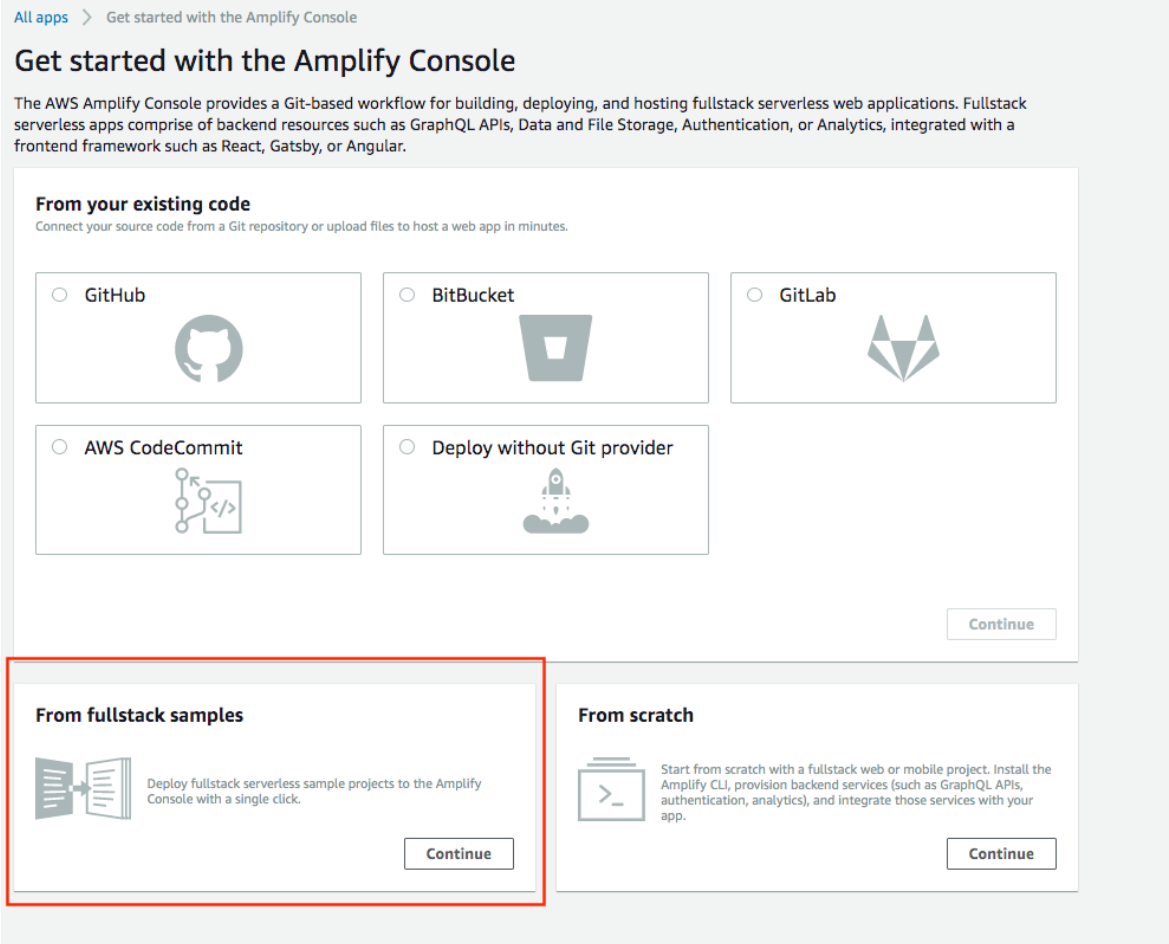
The Amplify Console enables developers building apps with the Amplify Framework to continuously deploy updates to their backend and frontend on every code commit. With the Amplify Console you can deploy serverless backends with GraphQL/REST APIs, authentication, analytics, and storage created by the Amplify CLI on the same commit as your frontend code.

Important
This feature only works with the Amplify CLI v4.0+

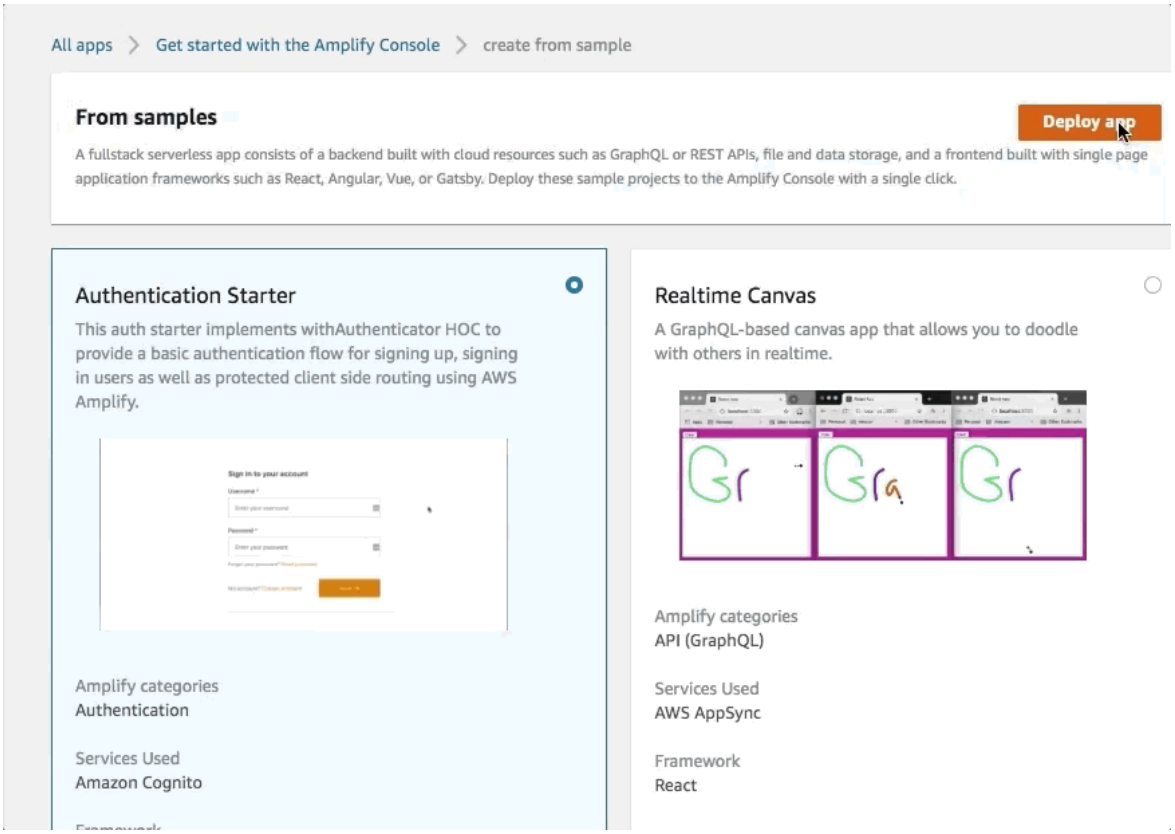
In this tutorial, we are going to create and deploy a React app which implements a basic authentication flow for signing up/signing in users as well as protected client side routing using Amplify.

Step 1: Deploy a fullstack sample

Log in to the Amplify Console and choose **Get Started** under **Deploy**. In the following screen, choose **From fullstack samples**. Alternatively, start your own adventure by building a backend **from scratch** by installing the Amplify CLI.

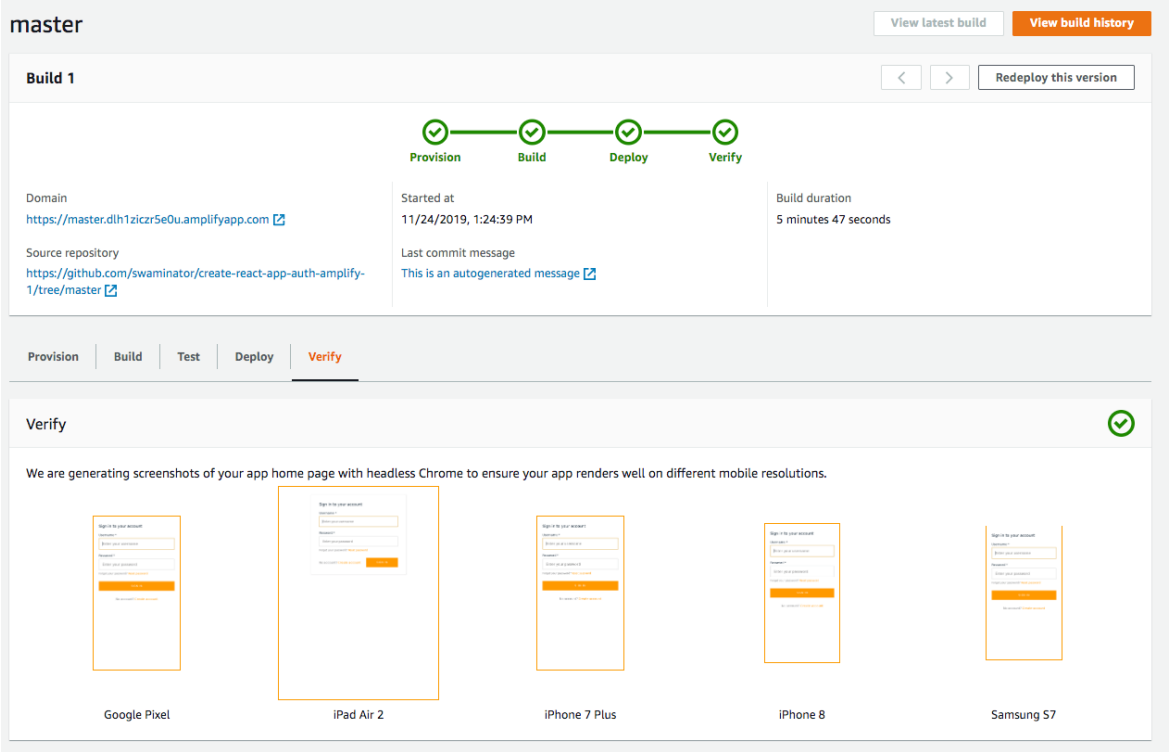


Choose the **Authentication Starter** and **Deploy app**. You will be asked to connect your GitHub account. Connecting your GitHub acccount allows the Amplify Console to create a fork of the repository in your account, deploy the AWS backend services, and build and deploy the frontend. In order to deploy backend resources to AWS, you will need to create a service role (p. 100).

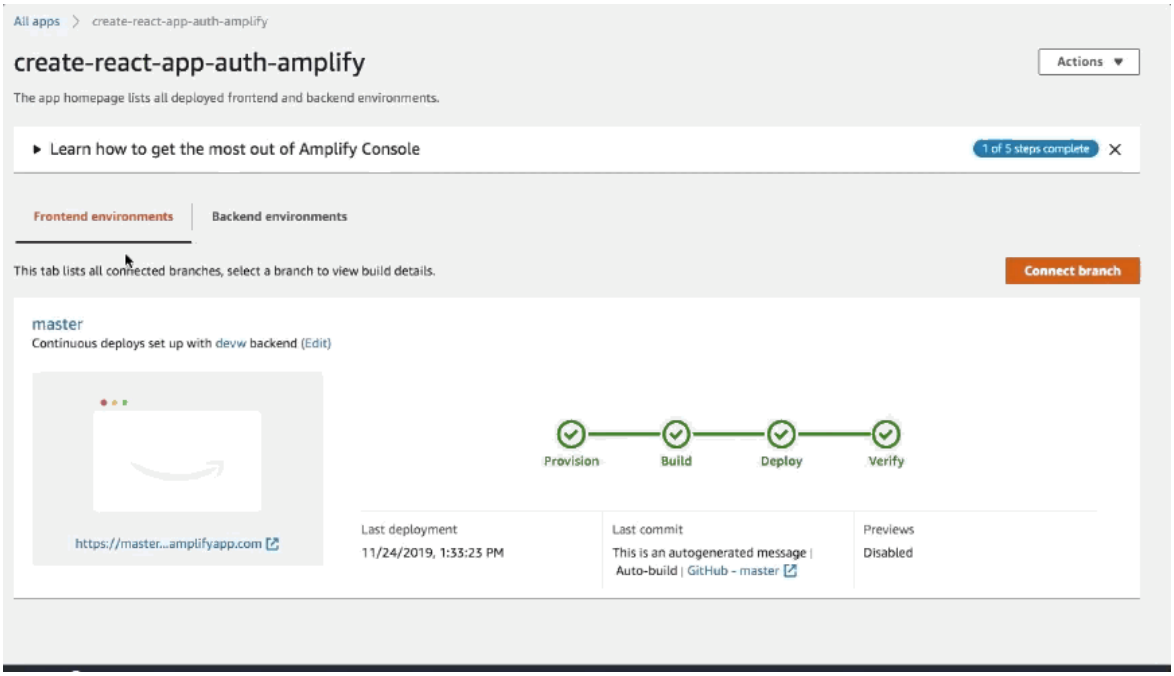


Step 2: Explore the fullstack app

Your app build will start by deploying the backend followed by the frontend. Click on the branch name to see the running build. When the build completes you will be able to see screenshots of your app on different devices.



At the end of the build, you will have one frontend environment (the main branch deployed at *'https://main.appid.amplifyapp.com'*) and one backend environment named devX. To add a user to your app, you can either register a user through the deployed frontend, or choose the **Authentication** tab which links to the Amazon Cognito UserPool. Create a user and try logging in to your app.



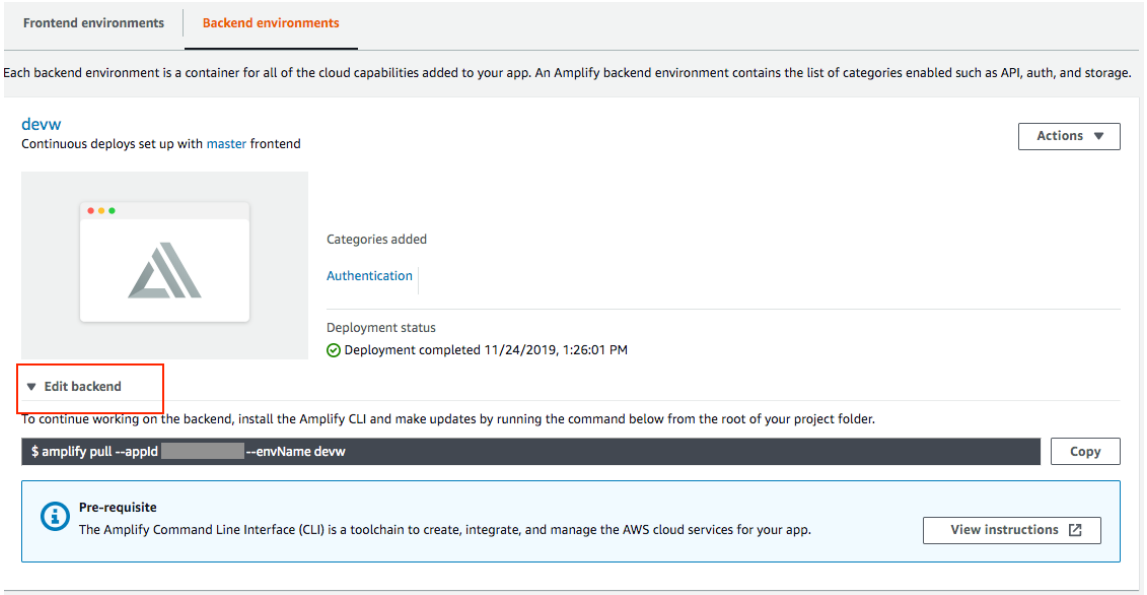
Step 3: Add a GraphQL backend

1. Let's add a GraphQL API backend with a NoSQL database to your app. To start, clone the repository that was forked in your account.

```
git clone git@github.com:<username>/create-react-app-auth-amplify.git
cd create-react-app-auth-amplify
```

2. From the **Backend environments** tab, choose **Edit backend**. As a pre-requisite, follow the instructions to install and configure the Amplify CLI. The Amplify command line toolchain allows you to edit the backend you just created to add more functionality such as GraphQL/REST APIs, analytics, and storage. Once the Amplify CLI is configured, copy the *amplify pull* command to connect to this backend from your local machine.

```
amplify pull --appId XXXXXXXX --envName devw
```



3. Add the GraphQL API using the default todo example. Learn more about modeling your backend with the GraphQL transform.

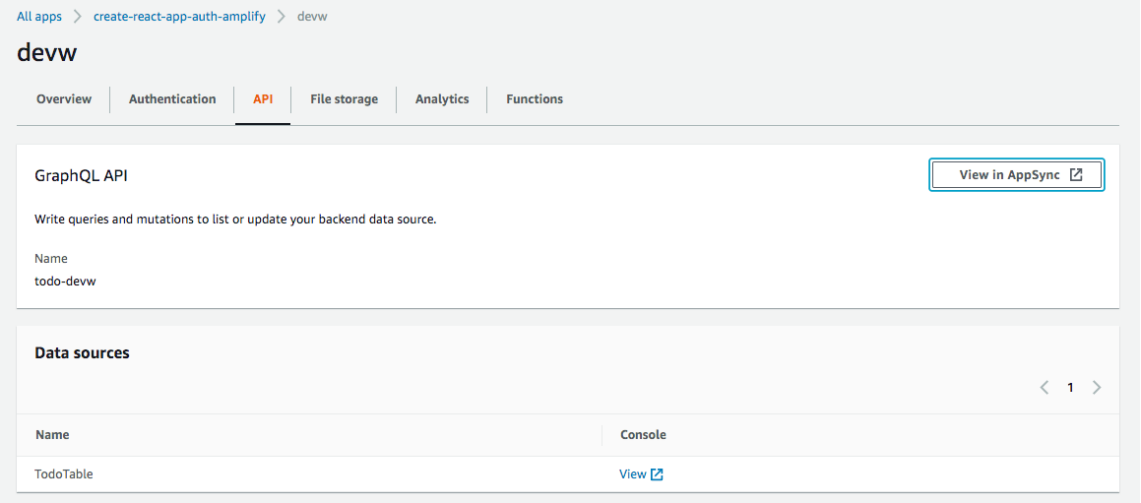
```
amplify add api
? Please select from one of the below mentioned services GraphQL
? Provide API name: todo
? Choose the default authorization type for the API API key
? Enter a description for the API key:
? After how many days from now the API key should expire (1-365): 7
? Do you want to configure advanced settings for the GraphQL API No, I am done.
? Do you have an annotated GraphQL schema? No
? Do you want a guided schema creation? (Y/n) Y
? What best describes your project: Single object with fields (e.g., “Todo” with ID,
 name, description)
? Do you want to edit the schema now? No
...
GraphQL schema compiled successfully.
```

4. To deploy these changes to the cloud run the following commands.

```
amplify push
Current Environment: devw

| Category | Resource name   | Operation | Provider plugin   |
| -------- | --------------- | --------- | ----------------- |
| Api      | todo            | Create    | awscloudformation |
| Auth     | cognitocf0c6096 | No Change | awscloudformation |
? Are you sure you want to continue? (Y/n) Y
..
# Generated GraphQL operations successfully and saved at src/graphql
# All resources are updated in the cloud
GraphQL endpoint: https://gumwpbojgraj5b547y5d3shurq.appsync-api.us-west-2.amazonaws.com/
graphql
GraphQL API KEY: da2-vlthvw5qcffxzl2hibgowv3rdq
```

5. Visit the Amplify Console to view the added API category. Choosing the API category will allow you to navigate to the AppSync Console (to write queries or mutations performing CRUD operations), or the DynamoDB Console (to view your Todo table).



6. Use the Amplify GraphQL client to write frontend code that lists and updates the todos. To deploy the updates to your frontend, simply commit your code and a new build will be triggered in the Amplify Console.

Next steps: Set up feature branch deployments

Follow our recommended workflow to set up feature branch deployments with multiple backend environments.

Deploy and host server-side rendered apps with Amplify

You can use AWS Amplify to deploy and host web apps that use server-side rendering (SSR). Currently, Amplify supports SSR apps created using the Next.js framework. When you deploy your app, Amplify automatically detects SSR–you do not have to perform any manual configuration in the AWS Management Console.

To learn about how Amplify supports SSR, review the following topics.

Topics

What is server-side rendering

Previously, Amplify supported the deployment and hosting of static web apps only. These include apps created with single-page application (SPA) frameworks such as React, and apps created with a static site generator (SSG) such as Gatsby. Static web apps consist of a combination of files, such as HTML, CSS, and JavaScript files, that are stored on a content delivery network (CDN). When a client browser makes a request to the website, the server returns a page to the client with an HTTP response and the client browser interprets the content and displays it to the user.

Amplify now supports web apps with server-side rendering (SSR). When a client sends a request to an SSR page, the HTML for the page is created on the server on each request. SSR enables a developer to customize a website per request and per user. In addition, SSR can improve performance and search engine optimization (SEO) for a website.

Amplify support for Next.js SSR

Currently Amplify supports deployment and hosting for server-side rendered (SSR) web apps created using Next.js only. Next.js is a React framework for developing SPAs with JavaScript. You can deploy apps built with Next.js 11 with features such as image and script optimization. Currently, Amplify doesn't fully support Incremental Static Regeneration (ISR).

Developers can use Next.js to combine static site generation (SSG), and SSR in a single project. SSG pages are prerendered at build time, and SSR pages are prerendered at request time.

Prerendering can improve performance and search engine optimization. Because Next.js prerenders all pages on the server, the HTML content of each page is ready when it reaches the client's browser. This content can also load faster. Faster load times improve the end user's experience with a website and positively impact the site's SEO ranking. Prerendering also improves SEO by enabling search engine bots to find and crawl a website's HTML content easily.

Next.js provides built-in analytics support for measuring various performance metrics, such as Time to first byte (TTFB) and First contentful paint (FCP). For more information about Next.js, see Getting started on the Next.js website.

Pricing for Next.js SSR apps

When deploying your Next.js SSR app, Amplify creates additional backend resources in your AWS account, including:

- An Amazon Simple Storage Service (Amazon S3) bucket that stores the resources for your app's static assets. For information about Amazon S3 charges, see Amazon S3 Pricing.
- An Amazon CloudFront distribution to serve the app. For information about CloudFront charges, see Amazon CloudFront Pricing.
- A Lambda@Edge function to customize the content that CloudFront delivers.

When you use the Amplify Framework (Libraries, CLI, UI components), you pay only for the underlying AWS services you use. For more information about Amplify deployment and hosting charges, see AWS Amplify Pricing.

Deploying a Next.js SSR app with Amplify

To deploy a Next.js SSR app with Amplify Console, follow the same workflow for setting up a static app with continuous deployments. For detailed instructions, see Getting started with existing code (p. 3). Note that you can't set up an SSR app in Amplify with manual deploys (p. 63).

Package.json file settings

When you deploy a Next.js app, Amplify inspects the app's build script in the `package.json` file to detect whether the app is SSR or SSG.

The following is an example of the build script for a Next.js SSR app. The build script `"next build"` indicates that the app supports both SSG and SSR pages.

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

The following is an example of the build script for a Next.js SSG app. The build script `"next build && next export"` indicates that the app supports only SSG pages.

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

Amplify build settings

After inspecting your app's `package.json` file to determine whether you are deploying an SSG or SSR app, Amplify checks the build settings for the app. You can save build settings in the Amplify console

or in an `amplify.yml` file in the root of your repository. For more information, see Configuring build settings (p. 36).

If Amplify detects that you are deploying a Next.js SSR app, and no `amplify.yml` file is present, it generates a buildspec for the app and sets `baseDirectory` to `.next`. If you are deploying an app where an `amplify.yml` file is present, the build settings in the file override any build settings in the console. Therefore, you must manually set the `baseDirectory` to `.next` in the file.

The following is an example of the build settings for an app where `baseDirectory` is set to `.next`. This indicates that the build artifacts are for a Next.js app that supports SSG and SSR pages.

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

If Amplify detects that you are deploying an SSG app, it generates a buildspec for the app and sets `baseDirectory` to out. If you are deploying an app where an `amplify.yml` file is present, you must manually set the `baseDirectory` to `out` in the file.

The following is an example of the build settings for an app where `baseDirectory` is set to `out`. This indicates that the build artifacts are for a Next.js app that supports only SSG pages.

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

Adding SSR functionality to a static Next.js app

You can add SSR functionality to an existing static (SSG) Next.js app deployed with Amplify Console. This requires three steps. First, add a service role to the app. Next, update the output directory in the app's build settings. Lastly, update the app's `package.json` file to indicate that the app uses SSR.

Add a service role

A service role is the AWS Identity and Access Management (IAM) role that Amplify Console assumes when calling other services on your behalf. Follow these steps to add a service role to an SSG app that's already deployed with Amplify Console.

To add a service role

1. Sign in to the AWS Management Console and open the Amplify console.
2. If you haven't already created a service role in your Amplify account, see create a service role (p. 100) to complete this prerequisite step.
3. Choose the static Next.js app that you want to add a service role to.
4. In the navigation pane, choose **App settings**, **General**.
5. On the **App details** page, choose **Edit**
6. For **Service role**, choose the name of an existing service role or the name of the service role that you created in step 2.
7. Choose **Save**.

Update build settings

Before you redeploy your app with SSR functionality, you must update the build settings for the app to set the output directory to `.next`. You can edit the build settings in the Amplify console or in an `amplify.yml` file stored in your repo. For more information see, Configuring build settings (p. 36).

The following is an example of the build settings for an app where `baseDirectory` is set to `.next`.

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

Update the package.json file

After you add a service role and update the build settings, update the app's `package.json` file. As in the following example, set the build script to `"next build"` to indicate that the Next.js app supports both SSG and SSR pages.

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

Amplify detects the change to the `package.json` file in your repo and redeploys the app with SSR functionality.

Updating the Next.js version for an existing app

When you deploy a new Next.js app with Amplify, by default Amplify uses the most recent supported version of Next.js. Currently, Amplify supports Next.js version 11.

For an existing app, use the following instructions to change the version of Next.js that Amplify uses to build the app.

To update the Next.js version for an existing app

1. Sign in to the AWS Management Console and open the Amplify console.
2. Choose the Next.js app that you want to update.
3. In the navigation pane, choose **App settings**, **Build settings**.
4. On the **Build settings** page, in the **Build image settings** section, choose **Edit**.
5. In the **Edit build image settings** dialog box, expand the **Add package version override** list, and choose **Next.js version**.
6. For **Version**, do one of the following:
   - Enter **`9`** for support up to Next.js version 9.4.*x*.
   - Enter **`10`** for support for Next.js versions 9.4.*x* to 10.*x*.*x*.
   - Enter **`latest`**, to always upgrade to the latest Next.js version that Amplify supports.
7. Choose **Save**. The next time the app builds, it can use the features supported by the Next.js version you specified in step 6.

Troubleshooting SSR deployment issues

If you experience unexpected issues when deploying an SSR app with Amplify, review the following troubleshooting topics.

Topics

Your output directory is overridden

The output directory for a Next.js app deployed with Amplify must be set to `.next`. If your app's output directory is being overridden, check the `next.config.js` file. To have the build output directory default to `.next`, remove the following line from the file:

```
distDir: 'build'
```

Verify that the output directory is set to `.next` in your build settings. For information about viewing your app's build settings, see Configuring build settings (p. 36).

The following is an example of the build settings for an app where `baseDirectory` is set to `.next`.

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

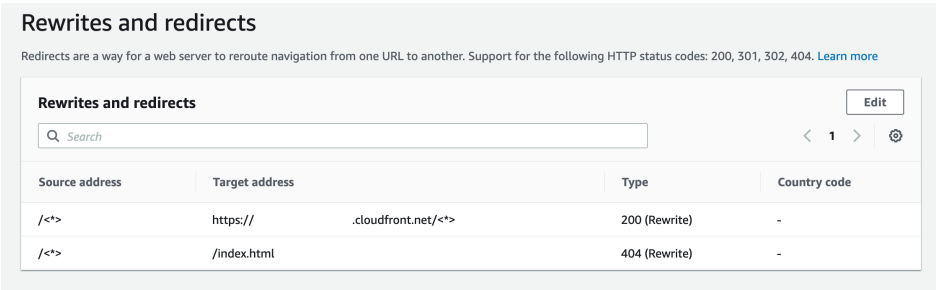
You get a 404 error after deploying your SSR site

If you get a 404 error after deploying your site, the issue could be caused by your output directory being overridden. To check your `next.config.js` file and verify the correct build output directory in your app's build spec, follow the steps in the previous topic, Your output directory is overridden (p. 18).

Your app is missing the rewrite rule for CloudFront SSR distributions

When you deploy an SSR app, Amplify creates a rewrite rule for your CloudFront SSR distributions. If you can't access your app in a web browser, verify that the CloudFront rewrite rule exists in your AWS account. If it's missing, you can either add it manually or redeploy your app.

To view or edit an app's rewrite and redirect rules in the Amplify console, in the navigation pane, choose **App settings**, then **Rewrites and redirects**. The following screenshot shows an example of the rewrite rules that Amplify creates for you when you deploy an SSR app.



Your app is too large to deploy

If you try to deploy a Next.js SSR app to Amplify and get a `RequestEntityTooLargeException` error, your app is too large to deploy. You can attempt to work around this issue by adding cache cleanup code to your `next.config.js` file.

The following is an example of code in the `next.config.js` file that performs cache cleanup.

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

Your app has both SSR and SSG branches

You can't deploy an app that has both SSR and SSG branches. If you need to deploy both SSR and SSG branches, you must deploy one app that uses only SSR branches and another app that uses only SSG branches.

Your app stores static files in a folder with a reserved path

Next.js can serve static files from a folder named `public` that's stored in the project's root directory. When you deploy and host a Next.js app with Amplify, your project can't include folders with the path `public/static`. Amplify reserves the `public/static` path for use when distributing the app. If your app includes this path, you must rename the `static` folder before deploying with Amplify.

Your app has reached a CloudFront limit

CloudFront service quotas limit your AWS account to 25 distributions with attached Lambda@Edge functions. If you exceed this quota, you can either delete any unused CloudFront distributions from your account or request a quota increase. For more information, see Requesting a quota increase in the *Service Quotas User Guide*.

Access control isn't available for your app

Amplify now supports access control for Next.js SSR apps. Previously, Amplify didn't support access control for Next.js SSR apps. If you are working with an SSR app that is already deployed in the Amplify console, you must redeploy the app to enable access control. After you redeploy, **Access control** displays in the **App settings** menu for the app.

Your Next.js app uses unsupported features

Amplify currently supports Next.js version 10.*x.x*, including Optional Catch All Routes, and Image Optimization. Currently, Amplify doesn't fully support Incremental Static Regeneration (ISR). In addition, Amplify supports Next.js version 11. For a list and description of these new features, see Next.js 11 on the Nextjs.org website.

When you deploy a new Next.js app, Amplify uses the most recent supported version of Next.js by default. If you have an existing Next.js app that you deployed to Amplify with an older version of Next.js, you can edit the app's build settings to use a newer version. For instructions, see Updating the Next.js version for an existing app (p. 18).

Unsupported Regions

Amplify doesn't support Next.js SSR app deployment in every AWS region where Amplify Console is available. Currently, Next.js SSR isn't supported in the following Regions: Europe (Milan) eu-south-1, Middle East (Bahrain) me-south-1, and Asia Pacific (Hong Kong) ap-east-1.

Set up custom domains

You can connect a custom domain to an app that you've deployed in the Amplify Console. You can purchase a custom domain through a domain registrar such as Amazon Route 53 , GoDaddy, or Google Domains. Route 53 is Amazon's Domain Name System (DNS) web service. For more information about using Route 53, see What is Amazon Route 53.

When you deploy your web app with the Amplify Console, it is hosted at:

```
https://branch-name.d1m7bkiki6tdw1.amplifyapp.com
```

When you connect a custom domain, users see that your app is hosted on a custom URL, such as the following:

```
https://www.example.com
```

The Amplify Console issues an SSL/TLS certificate for all domains connected to your app so that all traffic is secured through HTTPS/2. The certificate generated by AWS Certificate Manager (ACM) is valid for 13 months and renews automatically as long as your app is hosted with AWS Amplify.

Prior to connecting an app to a custom domain, the app must be deployed in AWS Amplify. For more information about completing this step, see Getting started with existing code (p. 3).

Connecting to a custom domain requires a basic knowledge of domains and DNS terminology. For more information about domains and DNS, see Understanding DNS terminology and concepts (p. 22).

Topics

Understanding DNS terminology and concepts

If you are unfamiliar with the terms and concepts associated with Domain Name System (DNS), the following topics can help you understand the procedures for adding custom domains.

DNS terminology

The following are a list of terms common to DNS. They can help you understand the procedures for adding custom domains.

CNAME

A Canonical Record Name (CNAME) is a type of DNS record that masks the domain for a set of webpages and makes them appear as though they are located elsewhere. A CNAME points a

subdomain to a fully qualified domain name (FQDN). For example, you can create a new CNAME record to map the subdomain **www.example.com**, where **www** is the subdomain, to the FQDN domain **branch-name.d1m7bkiki6tdw1.cloudfront.net** assigned to your app in the Amplify Console.

ANAME

An ANAME record is like a CNAME record, but at the root level. An ANAME points the root of your domain to an FQDN. That FQDN points to an IP address.

Name server

A name server is a server on the internet that's specialized in handling queries regarding the location of a domain name's various services. If you set up your domain in Amazon Route 53 , a list of name servers are already assigned to your domain.

NS record

An NS record points to name servers that look up your domain details.

DNS verification

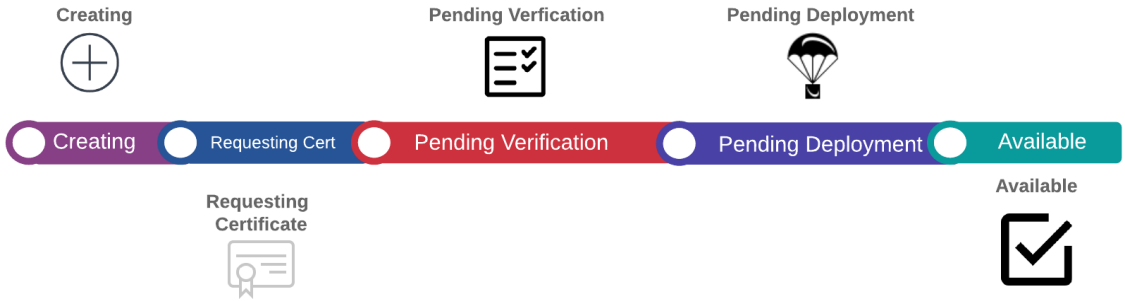
A Domain Name System (DNS) is like a phone book that translates human-readable domain names into computer-friendly IP addresses. When you type **`https://google.com`** into a browser, a lookup operation is performed in the DNS provider to find the IP Address of the server that hosts the website.

DNS providers contain records of domains and their corresponding IP Addresses. The most commonly used DNS records are CNAME, ANAME, and NS records.

The Amplify Console uses a CNAME record to verify that you own your custom domain. If you host your domain with Route 53, verification is done automatically on your behalf. However, if you host your domain with a third-party provider such as GoDaddy or Google, you have to manually update your domain's DNS settings and add a new CNAME record provided by the Amplify Console.

Amplify Console custom domain setup

When you add a custom domain in the Amplify Console, there are a number of steps that need to completed before you can view your app using your custom domain. The following graphic shows the order of the steps that the Amplify Console performs for SSL/TLS certificate creation, certificate configuration and verification, and domain activation.



The following list describes each step in the domain set up process in detail.

SSL/TLS create

The AWS Amplify Console issues an SSL/TLS certificate for setting up a secure custom domain.

SSL/TLS configuration and verification

Before issuing a certificate, the Amplify Console verifies that you are the owner of the domain. For domains managed by Amazon Route 53 , Amplify automatically updates the DNS verification record. For domains managed outside of Route 53, you need to manually add the DNS verification record displayed by the Amplify console into your domain with a third-party DNS provider.

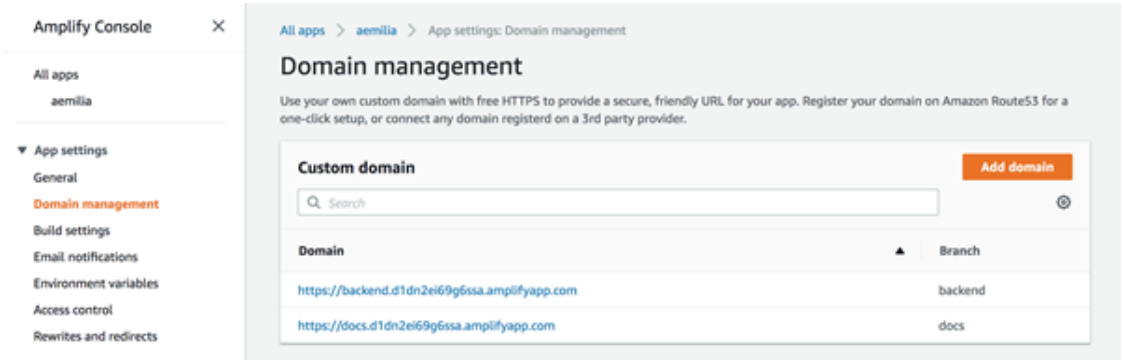
Domain activation

The domain is successfully verified. For domains managed outside of Route 53, you need to manually add the CNAME records displayed by the Amplify console into your domain with a third-party DNS provider.

Add a custom domain managed by Amazon Route 53

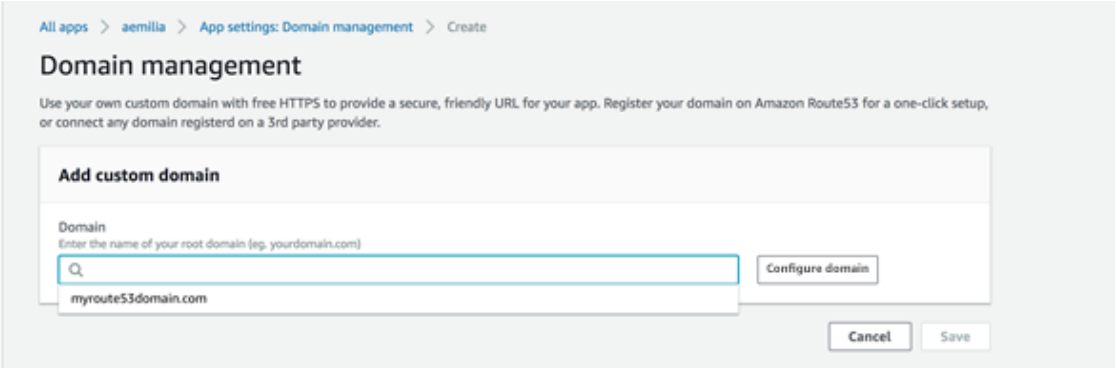
To add a custom domain managed by Amazon Route 53

1. Sign in to the AWS Management Console and open the Amplify Console.
2. Choose your app that you want to connect to a custom domain.
3. In the navigation pane, choose **App Settings**, **Domain management**.
4. On the **Domain management** page, choose **Add domain**.



5. For **Domain**, enter your root domain, choose the domain you want to use when it appears in the list, and then choose **Configure Domain**.

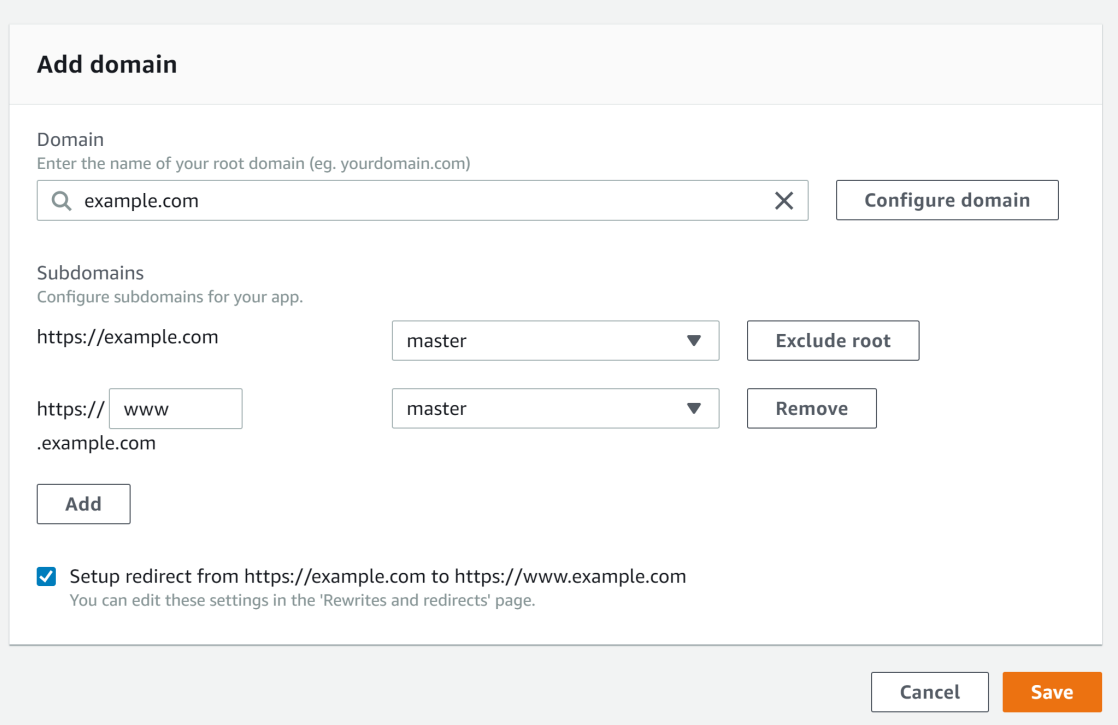
   As you start typing, any root domains that you already manage in Route 53 appear in the list. For example, if the name of your domain is **https://example.com**, enter **example.com** for **Domain**.



6. By default, the Amplify console automatically creates two subdomain entries for your domain. For example, if your domain name is **example.com**, you will see the subdomains **https://**

www.example.com and **https://example.com** with a redirect set up from the root domain to the **www** subdomain.

(Optional) You can modify the default configuration if you want to add subdomains only. To change the default configuration, choose **Rewrites and redirects** from the navigation pane, configure your domain, and then choose **Save**.



Note
It can take up to 24 hours for the DNS to propagate and to issue the certificate. For help with resolving errors that occur, see Troubleshooting custom domains (p. 33).

Add a custom domain managed by a third-party DNS provider

If you are not using Amazon Route 53 to manage your domain, you can add a custom domain managed by a third-party DNS provider to your app deployed in the Amplify Console.

If you are using GoDaddy or Google Domains, see the section called "Add a custom domain managed by GoDaddy" (p. 27) or the section called "Add a custom domain managed by Google Domains" (p. 30) for procedures specific to these providers.

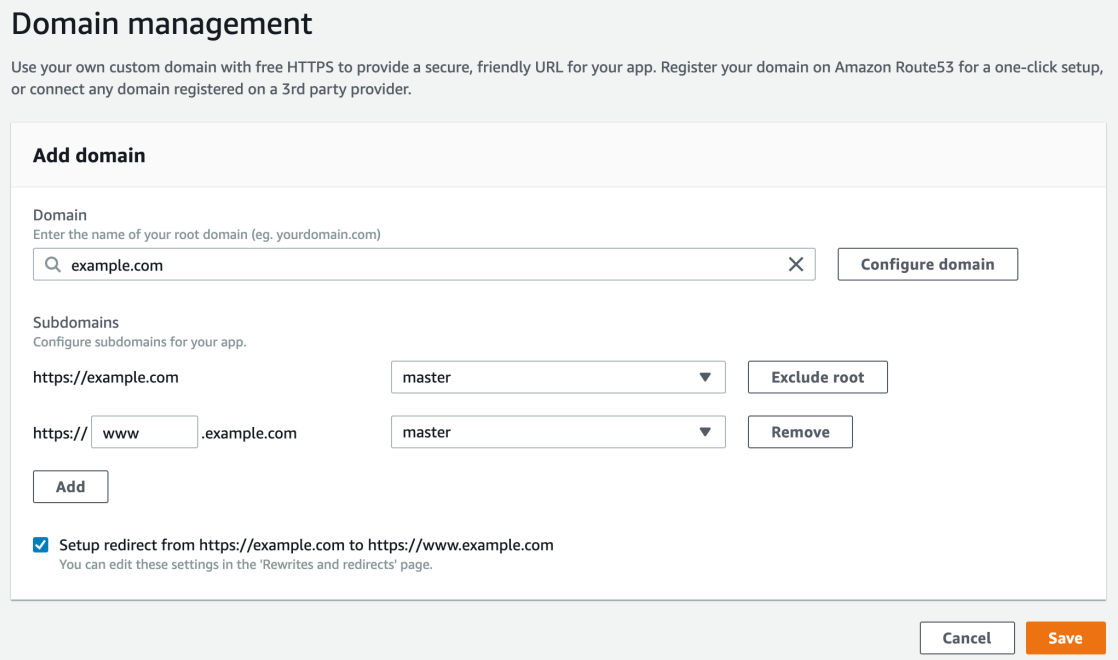
To add a custom domain managed by a third-party DNS provider

1. Sign in to the AWS Management Console and open the Amplify Console.
2. Choose your app that you want to add a custom domain to.
3. In the navigation pane, choose **App Settings**, **Domain management**.
4. On the Domain management page, choose **Add domain**.
5. For **Domain**, enter the name of your root domain, and then choose **Configure domain**. For example, if the name of your domain is **https://example.com**, enter **`example.com`**.

If you don't already own the domain and it is available, you can purchase the domain in Amazon Route 53.

6. By default, the Amplify Console automatically creates two subdomain entries for your domain. For example, if your domain name is **example.com**, you will see the subdomains **https://www.example.com** and **https://example.com** with a redirect set up from the root domain to the **www** subdomain.

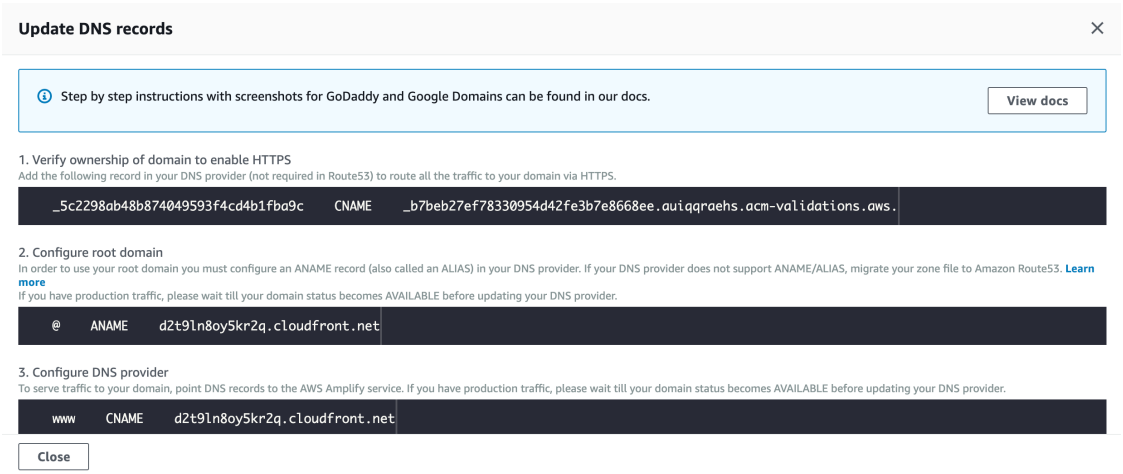
   (Optional) You can modify the default configuration if you want to add subdomains only. To change the default configuration, choose **Rewrites and redirects** from the navigation pane, configure your domain, and then choose **Save**.



7. Do one of the following:

   - If you're using GoDaddy, go to Add a custom domain managed by GoDaddy (p. 27).
   - If you're using Google Domains, go to Add a custom domain managed by Google Domains (p. 30).
   - If you're using a different third-party DNS provider, go to the next step in this procedure.

8. On the **Actions** menu, choose **View DNS records**. Use the DNS records displayed in the Amplify Console to update your DNS records with your third-party domain provider.



9. Go to your DNS provider's website, log in to your account, and locate the DNS management settings for your domain.
10. Configure a CNAME to point to the AWS validation server. For example, if the validation server is **_cjhwou20vhu2exampleuw20vuyb2ovb9.j9s73ucn9vy.acm-validations.aws**, enter **_cjhwou20vhu2exampleuw20vuyb2ovb9.j9s73ucn9vy.acm-validations.aws**. The Amplify Console uses this information to verify ownership of your domain and generate an SSL/TLS certificate for your domain. Once the Amplify Console validates ownership of your domain, all traffic will be served using HTTPS/2.

    Important
    It is important that you perform this step soon after adding your custom domain in the Amplify Console. The AWS Certificate Manager (ACM) immediately starts attempting to verify ownership. Over time, the checks become less frequent. If you add or update your CNAME records a few hours after you create your app, this can cause your app to get stuck in the pending verification state.
11. Configure a second CNAME record (for example, **https://*.example.com**), to point your subdomains to the Amplify domain. If you have production traffic, we recommended you update this CNAME record after your domain status shows as **AVAILABLE** in the Amplify Console.
12. Configure the ANAME/ALIAS record to point to the root domain of your amplifyapp domain (for example **https://example.com**). An ANAME record points the root of your domain to a hostname. If you have production traffic, we recommended that you update your ANAME record after your domain status shows as **AVAILABLE** in the console. For DNS providers that don't have ANAME/ALIAS support, we strongly recommend migrating your DNS to Route 53. For more information, see Configuring Amazon Route 53 as your DNS service.

    Note
    Verification of domain ownership and DNS propagation for third-party domains can take up to 48 hours. For help resolving errors that occur, see Troubleshooting custom domains (p. 33).

Add a custom domain managed by GoDaddy

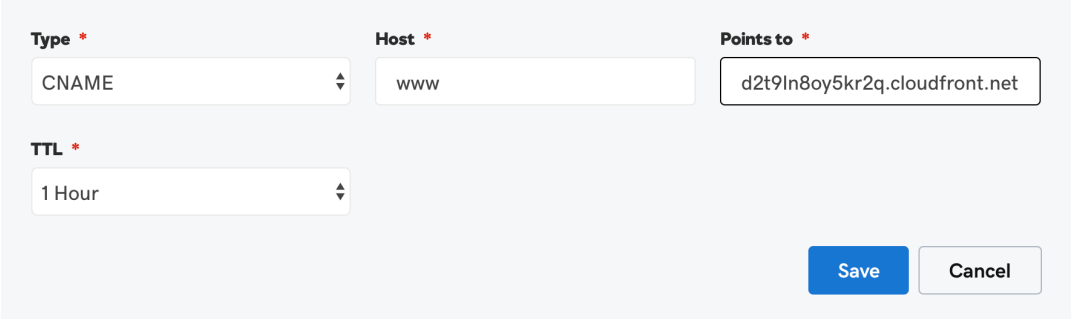
To add a custom domain managed by GoDaddy

1. Follow steps one through six of the procedure the section called “Add a custom domain managed by a third-party DNS provider” (p. 25).
2. Log in to your GoDaddy account.

3. In your list of domains, find the domain to add and choose **DNS**. GoDaddy displays a list of records for your domain. You need to add two new CNAME records.
4. Create the first CNAME record to point your subdomains to the Amplify domain.

   a. For **Host**, enter only the subdomain. For example, if your subdomain is **www.example.com**, enter **www** for **Host**.

   b. For **Points to**, look at your DNS records in the Amplify Console and then enter the value. If the Amplify Console displays the domain for your app as **xxxxxxxxxxxxx.cloudfront.net**, enter **xxxxxxxxxxxxx.cloudfront.net** for **Points to**.



5. Create the second CNAME record to point to the AWS Certificate Manager (ACM) validation server. A single validated ACM generates an SSL/TLS certificate for your domain.

   a. For **Host**, enter the subdomain.

   For example, if the DNS record in the Amplify Console for verifying ownership of your subdomain is **_c3e2d7eaf1e656b73f46cd6980fdc0e.example.com**, enter only **_c3e2d7eaf1e656b73f46cd6980fdc0e** for **Host**.

   b. For **Points to**, enter the ACM validation certificate.

   For example, if the validation server is **_cjhwou20vhu2exampleuw20vuyb2ovb9.j9s73ucn9vy.acm-validations.aws** , enter **_cjhwou20vhu2exampleuw20vuyb2ovb9.j9s73ucn9vy.acm-validations.aws** for **Points to**.

6. This step is not required for subdomains. GoDaddy doesn't support ANAME/ALIAS records. For DNS providers that do not have ANAME/ALIAS support, we strongly recommend migrating your DNS to Amazon Route 53 . For more information, see Configuring Amazon Route 53 as your DNS service.

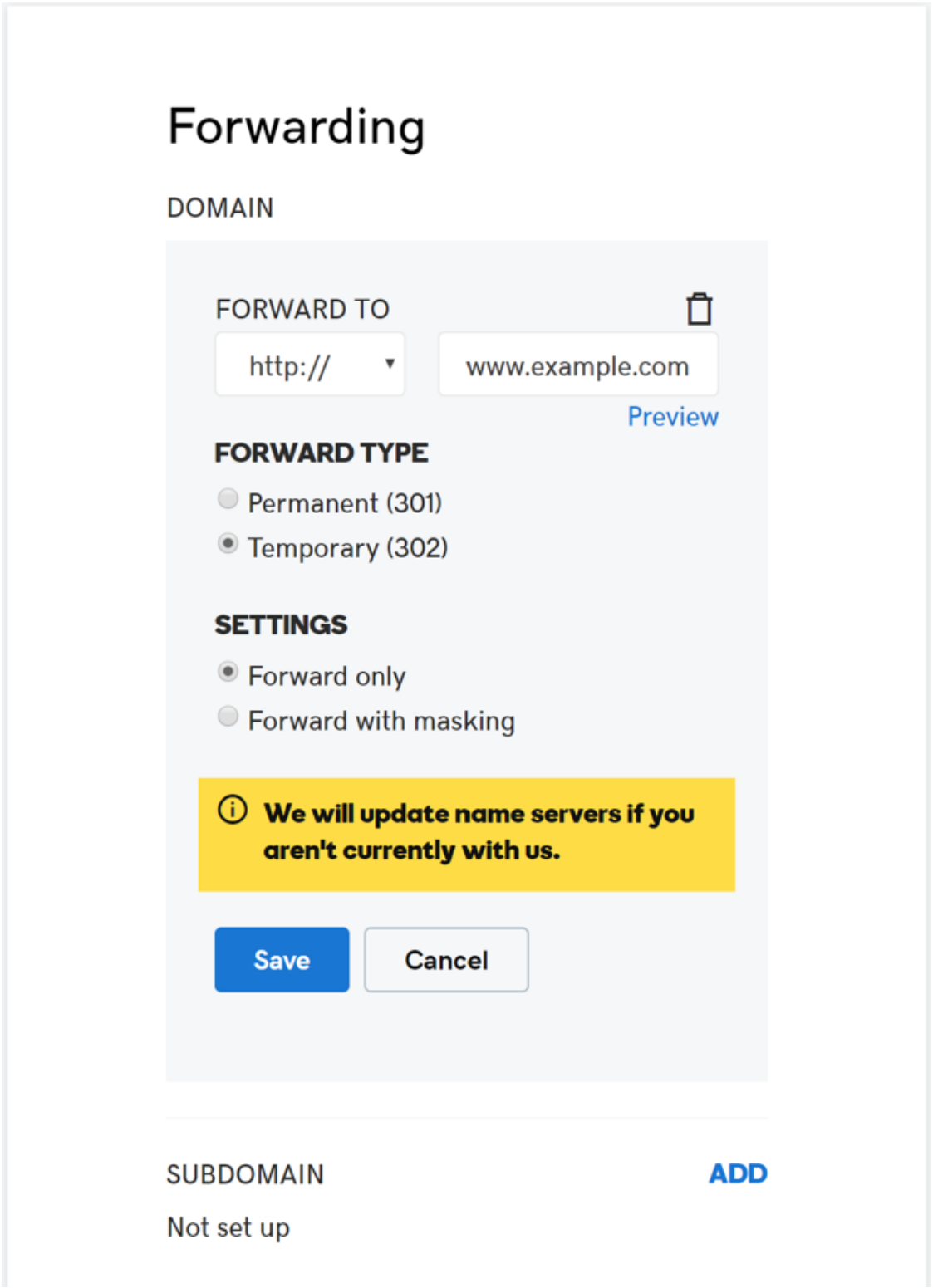
   If you want to keep GoDaddy as your provider and update the root domain, add **Forwarding** and set up a domain forward:

   a. Scroll down to the bottom of the **DNS Management** page to find the **Forwarding** box.

   b. For **Forward to**, choose **http://**, and then enter the name of your subdomain to foward to (for example, **www.example.com**).

   c. For **Forward Type**, choose **Temporary (302)**.

d. For **Settings**, choose **Forward only**.



Add a custom domain managed by Google Domains

To add a custom domain managed by Google Domains

1. Follow steps one through six of the procedure To add a custom domain managed by a third-party DNS provider (p. 25).
2. Log in to your account at https://domains.google.com and choose **DNS** in the left navigation pane.
3. Scroll down the page to **Custom resource records** where you need to add two new CNAME records.
4. Create the first CNAME record to point all subdomains to the Amplify domain as follows:

   a. For **Name**, enter only the subdomain name. For example, if your subdomain is **www.example.com**, enter **www** for **Name**.

   b. For **Data**, enter the value that's available in the Amplify Console.

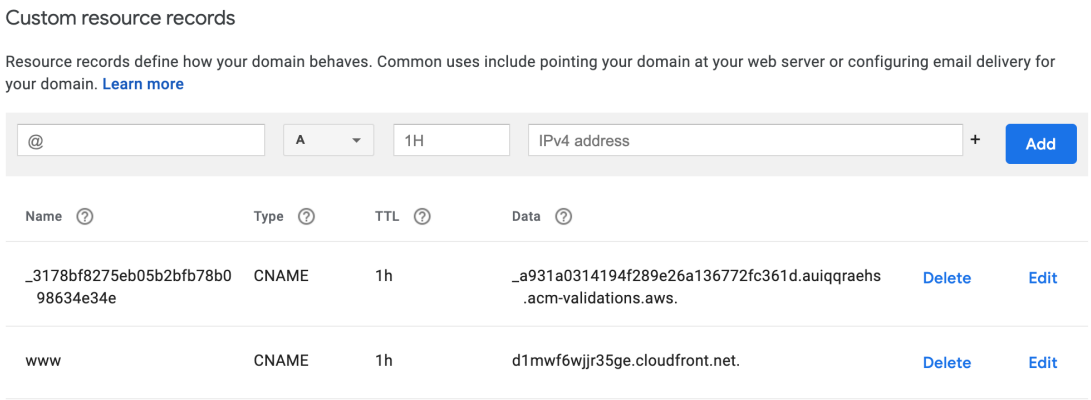
      If the Amplify Console displays the domain for your app as **xxxxxxxxxxxxx.cloudfront.net**, enter **xxxxxxxxxxxxx.cloudfront.net** for **Data**.
5. Create the second CNAME record to point to the AWS Certificate Manager (ACM) validation server. A single validated ACM generates an /TLS certificate for your domain.

   a. For **Name**, enter the subdomain.

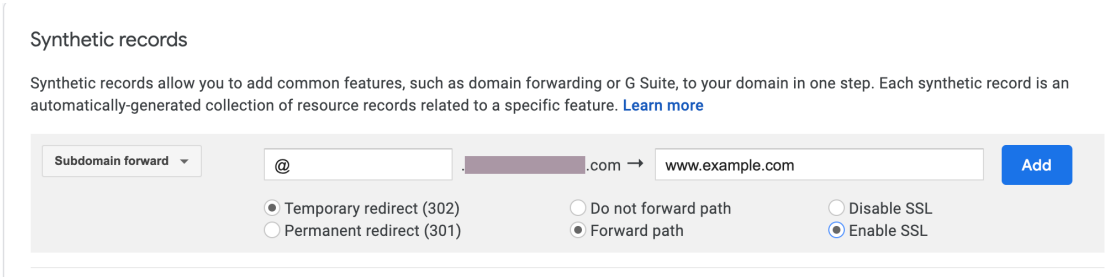
      For example, if the DNS record in the Amplify Console for verifying ownership of your subdomain is **_c3e2d7eaf1e656b73f46cd6980fdc0e.example.com**, enter only **_c3e2d7eaf1e656b73f46cd6980fdc0e** for **Name**.

   b. For **Data**, enter the ACM validation certificate.

      For example, if the validation server is **_68126cb4e8b7ab90c515ea3edb5be60d.hkvuiqjoua.acm-validations.aws.**, enter **_68126cb4e8b7ab90c515ea3edb5be60d.hkvuiqjoua.acm-validations.aws.** for **Data**.



6. Google Domains doesn't support ANAME/ALIAS records. For DNS providers that don't have ANAME/ALIAS support, we strongly recommend migrating your DNS to Amazon Route 53 . For more information, see Configuring Amazon Route 53 as your DNS service. If you want to keep Google Domains as your provider and update the root domain, set up a subdomain forward. Locate the **Synthetic records** pane. For **Subdomain**, enter the @ symbol to specify the root domain. For **Destination URL**, enter your subdomain to forward to.



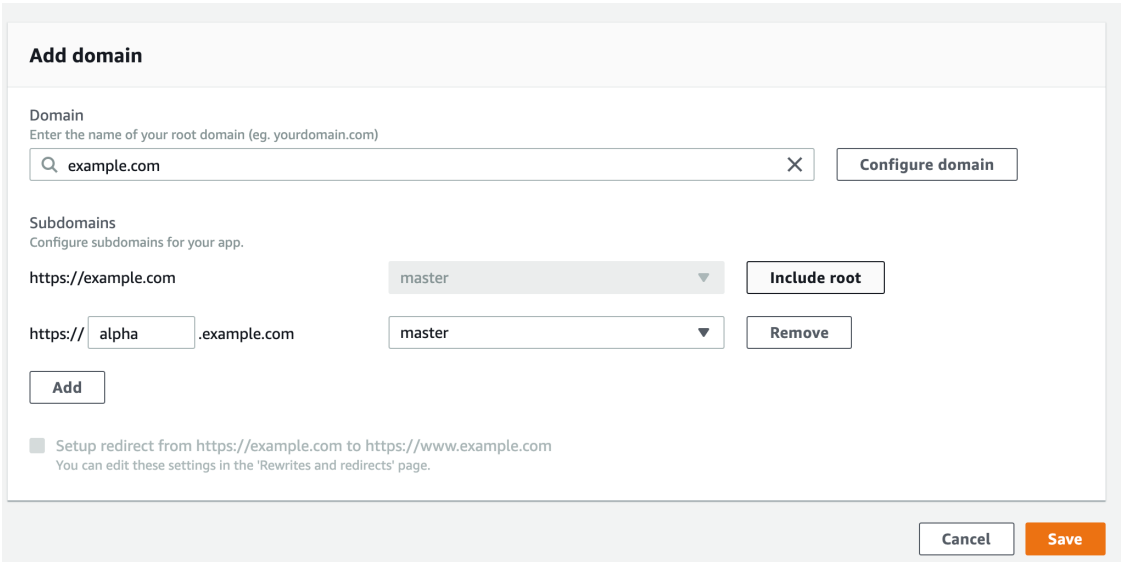
Note
Updates to your DNS settings for a Google domain can take up to 48 hours to take effect. For help with resolving errors that occur, see Troubleshooting custom domains (p. 33).

Manage subdomains

A subdomain is the part of your URL that appears before your domain name. For example, **www** is the subdomain of **www.amazon.com** and **aws** is the subdomain of **aws.amazon.com**. If you already have a production website, you might want to only connect a subdomain. Subdomains can also be multilevel, for example **beta.alpha.example.com** has the multilevel subdomain **beta.alpha**.

To add a subdomain only

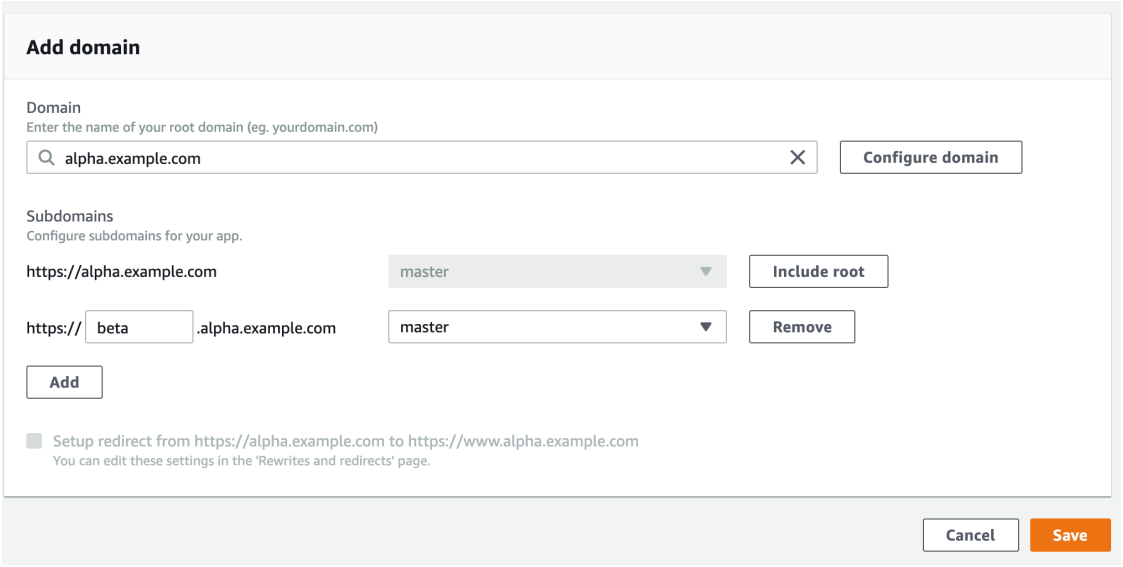
1. Sign in to the AWS Management Console and open the Amplify Console.
2. Choose your app that you want to add a subdomain to.
3. In the navigation pane, choose **App Settings**, and then choose **Domain management**.
4. On the **Domain management** page, choose **Add domain**.
5. For **Domain**, enter the name of your root domain and then choose **Configure domain**. For example, if the name of your domain is **https://example.com**, enter **example.com** for **Domain**.
6. Choose **Exclude root** and modify the name of the subdomain. For example if the domain is **example.com** you can modify it to only add the subdomain **alpha**.



To add a multilevel subdomain

1. Sign in to the AWS Management Console and open the Amplify Console.
2. Choose your app that you want to add a multilevel subdomain to.
3. In the navigation pane, choose **App Settings**, and then choose **Domain management**.
4. On the **Domain management** page, choose **Add domain**.
5. For **Domain**, enter the name of a domain with a subdomain, choose **Exclude root**, and modify the subdomain to add a new level.

   For example, if you have a domain called **alpha.example.com** and you want to create a multilevel subdoman **beta.alpha.example.com**, you would enter **beta** as the subdomain value, as shown in the following screenshot.



To add or edit a subdomain

After adding a custom domain to an app, you can edit an existing subdomain or add a new subdomain.

1. Sign in to the AWS Management Console and open the Amplify Console.
2. Choose your app that you want to manage subdomains for.
3. In the navigation pane, choose **App Settings**, and then choose **Domain management**.
4. On the **Domain management** page, choose **Manage subdomains**.
5. In **Edit domain**, you can edit your existing subdomains as needed.
6. (Optional) To add a new subdomain, choose **Add**.
7. Choose **Update** to save your changes.

Set up automatic subdomains for a Amazon Route 53 custom domain

After an app is connected to a custom domain in Route 53, Amplify Console enables you to automatically create subdomains for newly connected branches. For example, if you connect your **dev**

branch, Amplify can automatically create **dev.exampledomain.com**. When you delete a branch, any associated subdomains are automatically deleted.

To set up automatic subdomain creation for newly connected branches

1. Sign in to the AWS Management Console and open the Amplify Console.
2. Choose an app that is connected to a custom domain managed in Route 53.
3. In the navigation pane, choose **App Settings**, and then choose **Domain management**.
4. On the **Domain management** page, choose **Manage subdomains**.
5. Select the **Sub-domain auto-detection** check box on the bottom left side.

Note
This feature is available only for root domains, for example, **exampledomain.com**. The Amplify console doesn't display this check box if your domain is already a subdomain, such as **dev.exampledomain.com**.

Web previews with subdomains

After you enable **Sub-domain auto-detection** using the preceding instructions, your app's pull request web previews will also be accessible with automatically created subdomains. When a pull request is closed, the associated branch and subdomain are automatically deleted. For more information on setting up web previews for pull requests, see Web previews (p. 67).

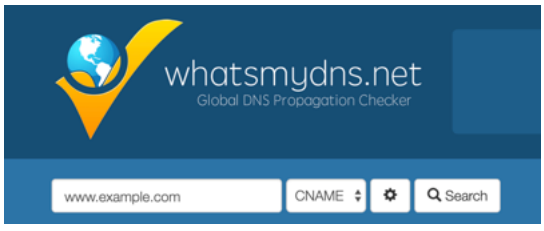
Troubleshooting custom domains

If you encounter issues when adding a custom domain to an app in the AWS Amplify Console, consult the following topics in this section.

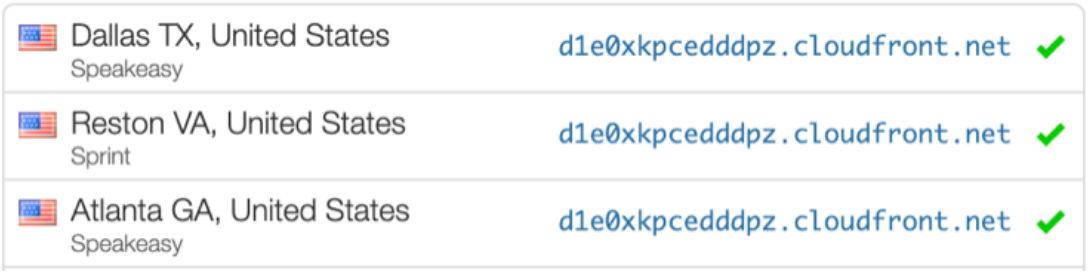
Topics

How do I verify that my CNAME resolves?

1. After you update your DNS records with your third-party domain provider, you can use a tool such as dig or a free website such as https://www.whatsmydns.net/ to verify that your CNAME record is resolving correctly. The following screenshot demonstrates how to use whatsmydns.net to check your CNAME record for the domain **www.example.com**.



2. Choose **Search**, and **whatsmydns.net** displays the results for your CNAME. The following screenshot is an example of a list of results that verify that the CNAME resolves correctly to a cloudfront.net URL.



My domain hosted with a third-party is stuck in the Pending Verification state

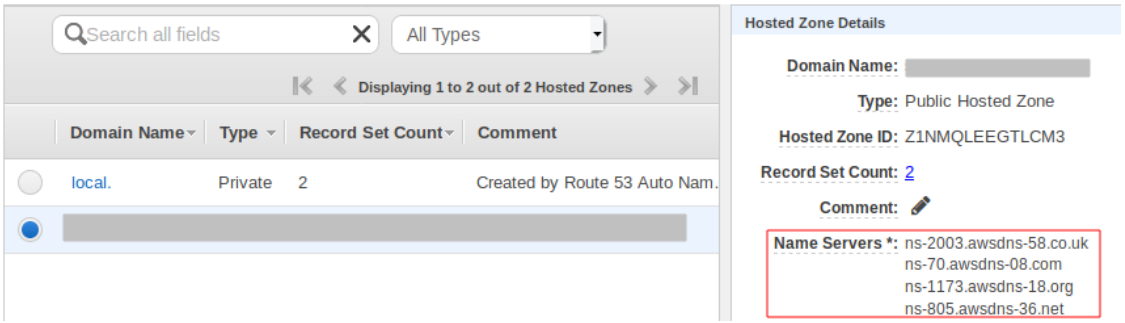
1. If your custom domain is stuck in the **Pending Verification** state, verify that your CNAME records are resolving. See the previous troubleshooting topic, How do I verify that my CNAME resolves, for instructions on performing this task.
2. If your CNAME records are not resolving, confirm that the CNAME entry exists in your DNS settings with your domain provider.

   Important
   It is important to update your CNAME records as soon as you create your custom domain. After your app is created in the Amplify Console, your CNAME record is checked every few minutes to determine if it resolves. If it doesn't resolve after an hour, the check is made every few hours, which can lead to a delay in your domain being ready to use. If you added or updated your CNAME records a few hours after you created your app, this is the most likely cause for your app to get stuck in the **Pending Verification** state.
3. If you have verified that the CNAME record exists, then there may be an issue with your DNS provider. You can either contact the DNS provider to diagnose why the DNS verification CNAME is not resolving or you can migrate your DNS to Route 53. For more information, see Making Amazon Route 53 the DNS service for an existing domain.

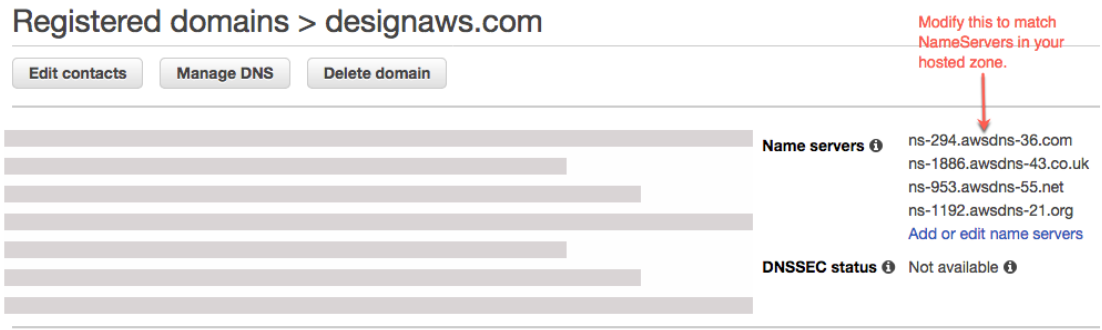
My domain hosted with Amazon Route 53 is stuck in the Pending Verification state

If you transferred your domain to Amazon Route 53 , it is possible that your domain has different name servers than those issued by the Amplify Console when your app was created. Perform the following steps to diagnose the cause of the error.

1. Sign in to the Amazon Route 53 console
2. In the navigation pane, choose **Hosted Zones** and then choose the name of the domain you are connecting.
3. Record the name server values from the **Hosted Zone Details** section. You need these values to complete the next step. The following screenshot of the Route 53 console displays the location of the name server values in the lower-right corner.



4. In the navigation pane, choose **Registered domains**. Verify that the name servers displayed on the **Registered domains** section match the name server values that you recorded in the previous step from the **Hosted Zone** Details section. If they do not match, edit the name server values to match the values in your **Hosted Zone**. The following screenshot of the Route 53 console displays the location of the name server values on the right side.



5. If this doesn't resolve the issue, see GitHub Issues and open a new issue if it doesn't already exist.

I get a CNAMEAlreadyExistsException error

If you get a **CNAMEAlreadyExistsException** error, this means that one of the host names that you tried to connect (a subdomain, or the apex domain) is already deployed to another Amazon CloudFront distribution. Perform the following steps to diagnose the cause of the error.

1. Sign in to the Amazon CloudFront console and verify that you don't have this domain deployed to any other distribution. A single CNAME record can be attached to one CloudFront distribution at a time.
2. If you previously deployed the domain to a CloudFront distribution you must remove it.
   a. Choose **Distributions** on the left navigation menu.
   b. Select the checkbox next to the name of the distribution to edit, then choose **Distribution Settings**.
   c. Choose the **General** tab, and then choose **Edit**.
   d. Remove the domain name from **Alternate Domain Names (CNAMEs)**. Then choose, **Yes, Edit** to save your change.
3. Check to see whether this domain is connected to a different Amplify app that you own. If so, make sure you are not trying to reuse one of the hostnames. If you are using **www.example.com** for another app, you cannot use **www.example.com** with the app that you are currently connecting. You can use other subdomains, such as **blog.example.com**.
4. If this domain was successfully connected to another app and then deleted within the last hour, try again after at least one hour has passed. If you still see this exception after 6 hours, see GitHub Issues and open a new issue if it doesn't already exist.

Configuring build settings

When you deploy an app with the Amplify Console, it automatically detects the front end framework and associated build settings by inspecting the `package.json` file in your repository. You have the following options for storing your app's build settings:

- Save the build settings in the Amplify Console - The Amplify Console autodetects build settings and saves them so that they can be accessed via the Amplify Console. Amplify applies these settings to all of your branches unless there is an `amplify.yml` file stored in your repository.
- Save the build settings in your repository - Download the `amplify.yml` file and add it to the root of your repository.

You can edit an app's build settings in the Amplify Console by choosing **App settings**, **Build settings**. The build settings are applied to all the branches in your app, except for the branches that have an `amplify.yml` file saved in the repository.

> **Note**
> **Build settings** is visible in the Amplify Console's **App settings** menu only when an app is set up for continuous deployment and connected to a git repository. For instructions on this type of deployment, see Getting started with existing code (p. 3).

Build specification YAML syntax

The build specification YAML contains a collection of build commands and related settings that the Amplify Console uses to run your build. The YAML is structured as follows:

```
version: 1
env:
  variables:
    key: value
backend:
  phases:
    preBuild:
      commands:
        - *enter command*
    build:
      commands:
        - *enter command*
    postBuild:
        commands:
        - *enter command*
frontend:
  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*
```

- **version** - Represents the Amplify Console YAML version number.
- **appRoot** - The path within the repository that this application resides in. *Ignored unless multiple applications are defined.*
- **env** - Add environment variables to this section. You can also add environment variables using the console.
- **backend** - Run Amplify CLI commands to provision a backend, update Lambda functions, or GraphQL schemas as part of continuous deployment. Learn how to deploy a backend with your frontend (p. 9).
- **frontend** - Run frontend build commands.
- **test** - Run commands during a test phase. Learn how to add tests to your app (p. 69).
- **The frontend, backend, and test have three phases that represent the commands run during each sequence of the build.**
    - **preBuild** - The preBuild script runs before the actual build starts, but after we have installed dependencies.
    - **build** - Your build commands.
    - **postBuild** - The post-build script runs after the build has finished and we have copied all the necessary artifacts to the output directory.
- **artifacts>base-directory** - The directory in which your build artifacts exist.
- **artifacts>files** - Specify files from your artifact you want to deploy. ***/** is to include all files.
- **cache** - The buildspec's cache field is used to cache build-time dependencies such as the *node_modules* folder, and is automatically suggested based on the package manager and framework that the customer's app is built in. During the first build, any paths here are cached, and on subsequent builds we re-inflate the cache and use those cached dependencies where possible to speed up build time.

Branch-specific build settings

You can use bash shell scripting to set branch-specific build settings. For example, the following script uses the system environment variable *$AWS_BRANCH* to execute one set of commands if the branch name is *main* and a different set of commands if the branch name is *dev*.

```
frontend:
```

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

Navigating to a subfolder

For monorepos, users want to be able to cd into a folder to run the build. After you run the cd command, it applies to all stages of your build so you don't need to repeat the command in separate phases.

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

Deploying the backend with the front end

The `amplifyPush` is a helper script that helps you with backend deployments. The build settings below automatically determine the correct backend environment to deploy for the current branch.

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

Setting the output folder

The following build settings set the output directory to the public folder.

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

Installing packages as part of a build

You can use npm or yarn to install packages during the build.

```
frontend:
  phases:
    build:
      commands:
        - npm install -g pkg-foo
        - pkg-foo deploy
        - yarn run build
  artifacts:
    baseDirectory: public
```

Using a private npm registry

You can add references to a private registry in your build settings or add it as an environment variable.

```
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
```

Installing OS packages

You can install OS packages for missing dependencies.

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

Key-value storage for every build

The `envCache` provides key-value storage at build time. Values stored in the `envCache` can only be modified during a build and can be re-used at the next build. Using the `envCache`, we can store information on the deployed environment and make it available to the build container in successive builds. Unlike values stored in the `envCache`, changes to environment variables during a build are not persisted to future builds.

Example usage:

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

Skip build for a commit

To skip an automatic build on a particular commit, include the text **[skip-cd]** at the end of the commit message.

Disable automatic builds

You can configure Amplify Console to disable automatic builds on every code commit. To set up, choose **App settings**, **General**, and then scroll to the **Branches** section that lists the connected branches. Select a branch, and then choose **Action**, **Disable auto build**. Further commits to that branch will no longer trigger a new build.

Enable or disable diff based frontend build and deploy

You can configure Amplify Console to use diff based frontend builds. If enabled, at the start of each build Amplify Console attempts to run a diff on either your `appRoot`, or the `/src/` folder by default. If Amplify doesn't find any differences, it skips the frontend build, test (if configured), and deploy steps, and does not update your hosted app.

To configure diff based frontend build and deploy

1. Sign in to the AWS Management Console and open the Amplify Console.
2. Choose the app to configure diff based frontend build and deploy for.
3. In the navigation pane, choose **App settings**, **Environment variables**.
4. In the **Environment variables** section, choose **Manage variables**.
5. The procedure for configuring the environment variable varies depending on whether you are enabling or disabling diff based frontend build and deploy.
   - To enable diff based frontend build and deploy
     a. In the **Manage variables** section, under **Variable**, enter `AMPLIFY_DIFF_DEPLOY`.
     b. For **Value**, enter `true`.
   - To disable diff based frontend build and deploy
     - Do one of the following:
       - In the **Manage variables** section, locate `AMPLIFY_DIFF_DEPLOY`. For **Value**, enter `false`.
       - Remove the `AMPLIFY_DIFF_DEPLOY` environment variable.

Optionally, you can set the `AMPLIFY_DIFF_DEPLOY_ROOT` environment variable to override the default path with a path relative to the root of your repo, such as `dist`.

Enable or disable diff based backend builds

You can configure Amplify to use diff based backend builds using the `AMPLIFY_DIFF_BACKEND` environment variable. When you enable diff based backend builds, at the start of each build, Amplify attempts to run a diff on the `amplify` folder in your repository. If Amplify doesn't find any differences, it skips the backend build step, and doesn't update your backend resources. If your project doesn't

have an `amplify` folder in your repository, Amplify ignores the value of the `AMPLIFY_DIFF_BACKEND` environment variable.

To configure diff based backend builds

1. Sign in to the AWS Management Console and open the Amplify Console.
2. Choose the app to configure diff based backend builds for.
3. In the navigation pane, choose **App settings**, **Environment variables**.
4. In the **Environment variables** section, choose **Manage variables**.
5. The procedure for configuring the environment variable varies depending on whether you are enabling or disabling diff based backend builds.
   - To enable diff based backend builds
     a. In the **Manage variables** section, under **Variable**, enter `AMPLIFY_DIFF_BACKEND`.
     b. For **Value**, enter `true`.
   - To disable diff based backend builds
     - Do one of the following:
       - In the **Manage variables** section, locate `AMPLIFY_DIFF_BACKEND`. For **Value**, enter `false`.
       - Remove the `AMPLIFY_DIFF_BACKEND` environment variable.

Monorepo build settings

When you store multiple projects or microservices in a single repository, it is called a monorepo. You can use Amplify to deploy applications in a monorepo without creating multiple build configurations or branch configurations.

You can save the build settings for a monorepo in the Amplify Console or you can download the `amplify.yml` file and add it to the root of your repository. Amplify applies the settings saved in the console to all of your branches unless it finds an `amplify.yml` file in your repository. When an `amplify.yml` file is present, its settings override any build settings saved in the Amplify Console.

Monorepo build specification YAML syntax

The YAML syntax for a monorepo build specification differs from the YAML syntax for a repo that contains a single application. For a monorepo, you declare each project in a list of applications. You must provide the following additional information for each application you declare in your monorepo build specification:

appRoot

The root, within the repository, that the application starts in. This key must exist, and have the same value as the `AMPLIFY_MONOREPO_APP_ROOT` environment variable. For instructions on setting this environment variable, see Setting the AMPLIFY_MONOREPO_APP_ROOT environment variable (p. 43).

The following monorepo build specification example demonstrates how to declare multiple Amplify applications in the same repo. The two apps, `react-app`, and `angular-app` are declared in the `applications` list. The `appRoot` key for each app indicates that the app is located in the `apps` root folder in the repo.

```
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:
    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*
```

Setting the AMPLIFY_MONOREPO_APP_ROOT environment variable

When you deploy an app stored in a monorepo, the app's `AMPLIFY_MONOREPO_APP_ROOT` environment variable must have the same value as the path of the app root, relative to the root of your repository. For example, a monorepo named `ExampleMonorepo` with a root folder named `apps`, that contains, `app1`, `app2`, and `app3` has the following directory structure:

```
ExampleMonorepo
  apps
    app1
    app2
    app3
```

In this example, the value of the `AMPLIFY_MONOREPO_APP_ROOT` environment variable for `app1` is `apps/app1`.

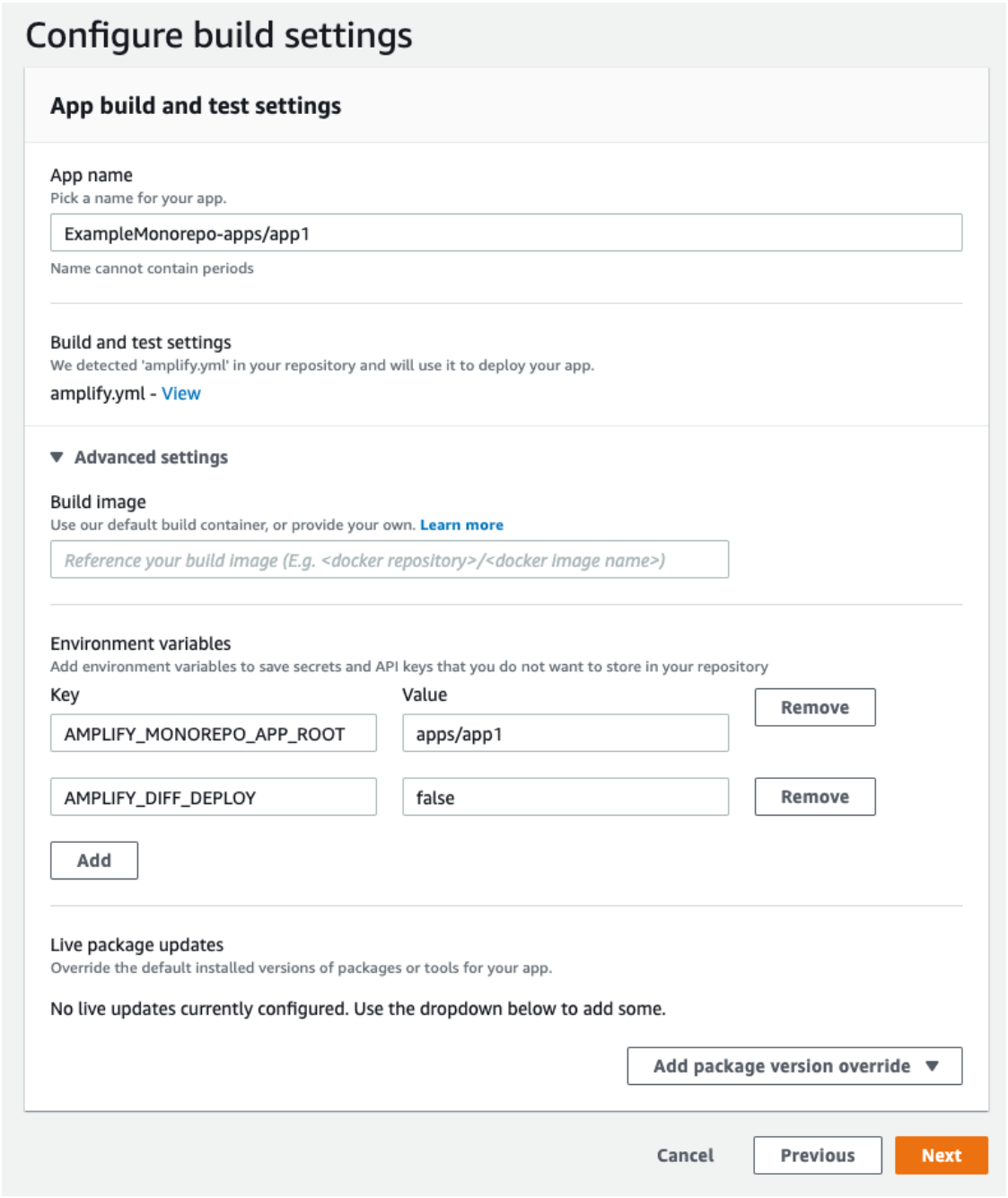
When you deploy a monorepo app using the Amplify console, the console automatically sets the `AMPLIFY_MONOREPO_APP_ROOT` environment variable using the value that you specify for the path to the app's root. However, if your monorepo app already exists in Amplify or is deployed using AWS CloudFormation, you must manually set the `AMPLIFY_MONOREPO_APP_ROOT` environment variable in the **Environment variables** section in the Amplify console.

Setting the AMPLIFY_MONOREPO_APP_ROOT environment variable automatically during deployment

The following instructions demonstrate how to deploy a monorepo app with the Amplify console. Amplify automatically sets the `AMPLIFY_MONOREPO_APP_ROOT` environment variable using the app's root folder that you specify in the console.

To deploy a monorepo app with AmplifyConsole

1. Sign in to the AWS Management Console and open the Amplify Console.
2. Choose **New app**, **Host web app** in the upper right corner.
3. On the **Host your web app** page, choose your Git provider, then choose **Continue**.
4. On the **Add repository branch** page, do the following:
   a. Choose the name of your repository from the list of **Recently updated repositories**.
   b. For **Branch**, choose the name of the branch to use.
   c. Select **Connecting a monorepo? Pick a folder.**
   d. Enter the path to your app in your monorepo, for example, **`apps/app1`**.
   e. Choose **Next**.
5. On the **Configure build settings** page you can use the default settings or customize the build settings for your app. In the following example screenshot, Amplify detects an `amplify.yml` file in the repository to use for the build settings. In the **Environment variables** section, Amplify has set `AMPLIFY_MONOREPO_APP_ROOT` to `apps/app1`, using the path you specified in step 4d.



6. Choose **Next**.
7. On the **Review** page, choose **Save and deploy**.

Setting the AMPLIFY_MONOREPO_APP_ROOT environment variable for an existing app

Use the following instructions to manually set the `AMPLIFY_MONOREPO_APP_ROOT` environment variable for an app that is already deployed to Amplify, or has been created using CloudFormation.

To set the AMPLIFY_MONOREPO_APP_ROOT environment variable for an existing app

1. Sign in to the AWS Management Console and open the Amplify Console.
2. Choose the name of the app to set the environment variable for.
3. In the navigation pane, choose **App Settings**, and then choose **Environment variables**.
4. On the **Environment variables** page, choose **Manage variables**.
5. In the **Manage variables** section, do the following:

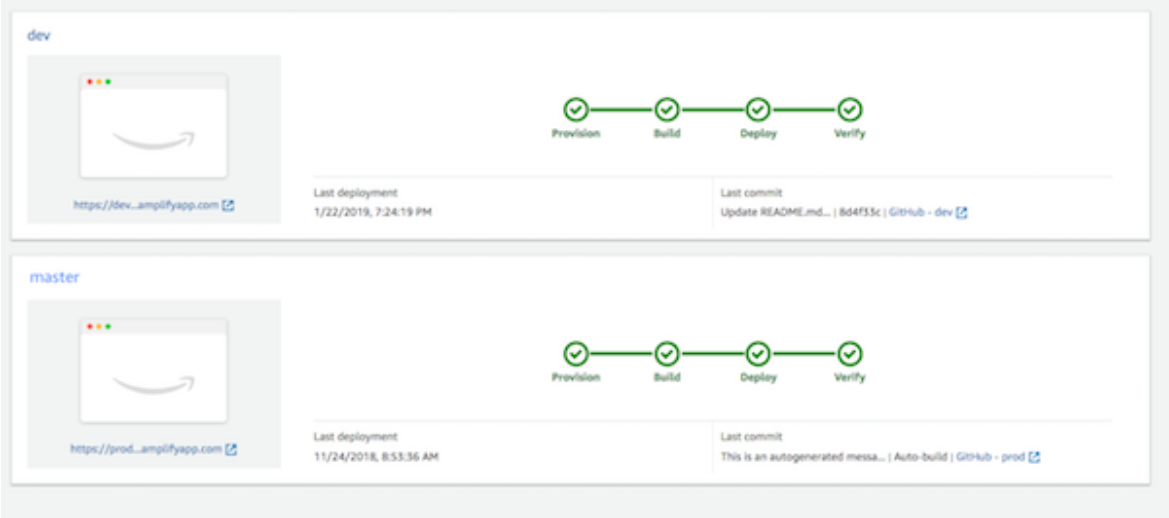
   a. Choose **Add variable**.
   b. For **Variable**, enter the key `AMPLIFY_MONOREPO_APP_ROOT`.
   c. For **Value**, enter the path to the app, for example **`apps/app1`**.
   d. For **Branch**, by default Amplify applies the environment variable to all branches.
6. Choose **Save**.

Feature branch deployments and team workflows

The Amplify Console is designed to work with feature branch and GitFlow workflows. The Amplify Console leverages Git branches to create new deployments every time a developer connects a new branch in their repository. After connecting your first branch, you can create a new feature branch deployment by adding a branch as follows:

1. On the branch list page, choose **Connect branch**.
2. Choose a branch from your repository.
3. Save and then deploy your app.

Your app now has two deployments available at *https://main.appid.amplifyapp.com* and *https://dev.appid.amplifyapp.com*. This may vary from team-to-team, but typically the **main branch** (formerly referred to as the master branch) tracks release code and is your production branch. The **develop branch** is used as an integration branch to test new features. This way beta testers can test unreleased features on the develop branch deployment, without affecting any of the production end users on the main branch deployment.



Topics

Team workflows with Amplify backend environments

A feature branch deployment consists of a **frontend**, and (optionally) a **backend** environment. The frontend is built and deployed to a global content delivery network (CDN), while the backend is deployed by the Amplify CLI to AWS. For more information about this deployment scenario, see Getting started with fullstack continuous deployments (p. 9).

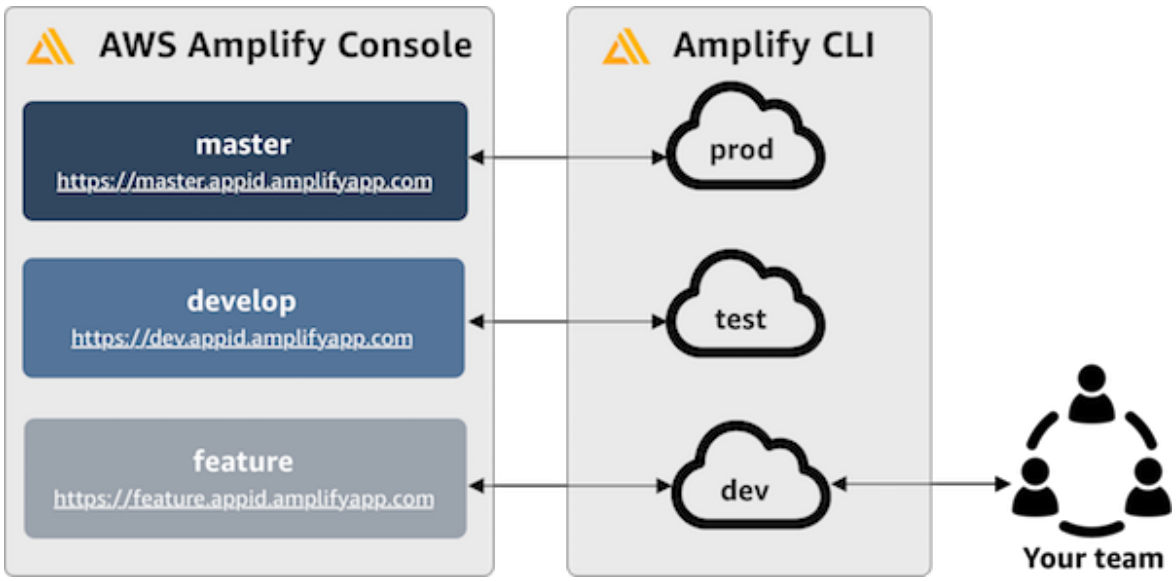
> **Note**
> Now you can easily reuse Amplify backend environments across your Amplify apps. For more information, see Use Amplify backends across apps (p. 58).

You can use the Amplify Console to continuously deploy backend resources such as GraphQL APIs and Lambda functions with your feature branch deployment. You can use the following models to deploy your backend and frontend with the Amplify Console:

Topics

Feature branch workflow

- Create **prod**, **test**, and **dev** backend environments with the Amplify CLI.
- Map **prod** and **test** to **main** (formerly referred to as master) and **develop** branches.
- Teammates can use the **dev** backend environment to test against **feature** branches.



1. Install the Amplify CLI to initialize a new Amplify project.

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

2. Initialize a *prod* backend environment for your project. If you don't have a project, create one using bootstrap tools like create-react-app or Gatsby.

```
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. Add *test* and *dev* backend environments.

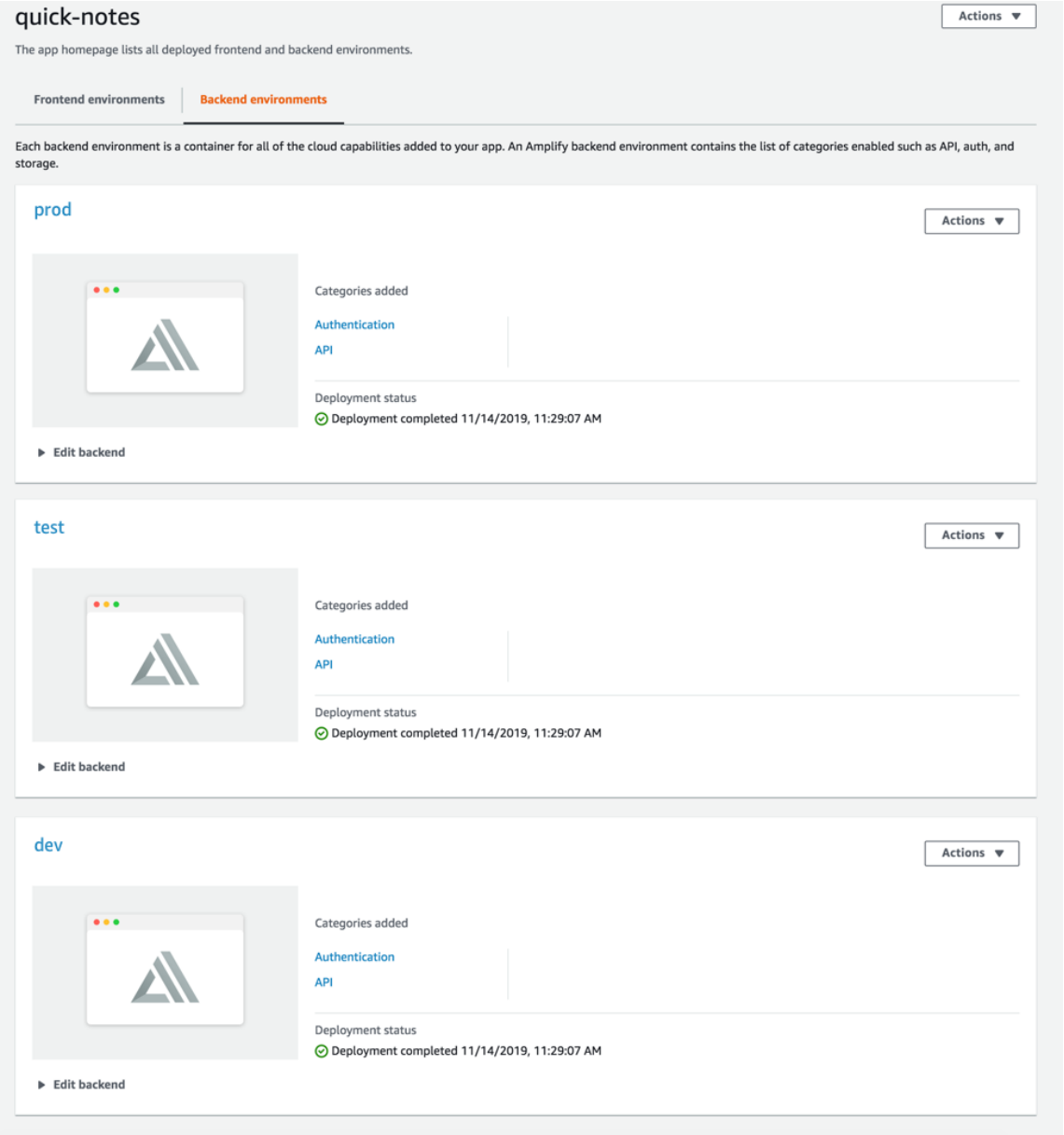
```
amplify env add
 ? Do you want to use an existing environment? (Y/n): n
 ? Enter a name for the environment: test
...
amplify push

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

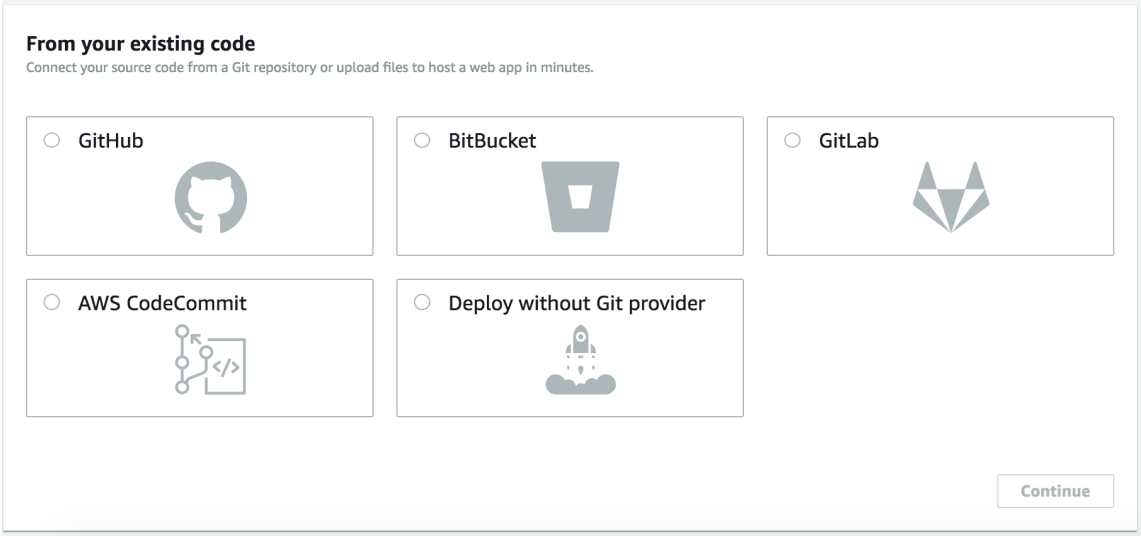
4. Push code to a Git repository of your choice (in this example we'll assume you pushed to main).

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

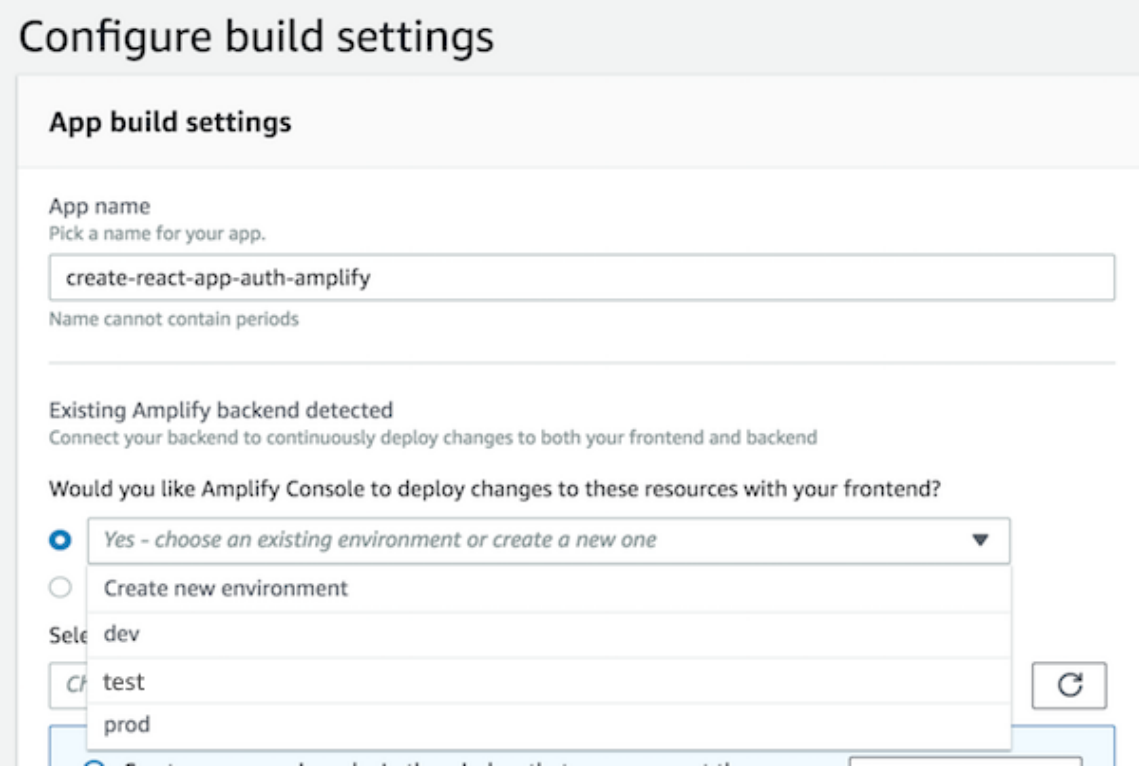
5. Visit the Amplify Console in the AWS Management Console to see your current backend environment. Navigate a level up from the breadcrumb to view a list of all backend environments created in the **Backend environments** tab.



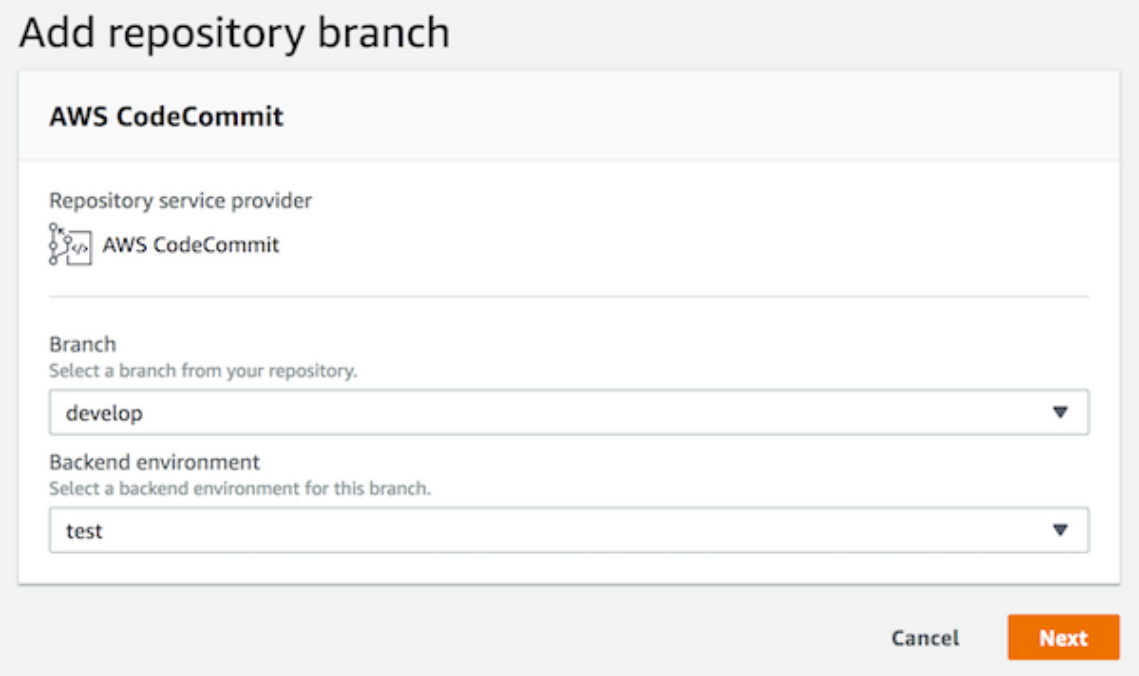
6. Switch to the **Frontend environments** tab and connect your repository provider and *main* branch.



7. In the build settings screen, pick an existing backend environment to set up continuous deployment with the main branch. Choose *prod* from the dropdown and grant the service role to Amplify Console. Choose **Save and deploy**. After the build completes you will get a main branch deployment available at *https://main.appid.amplifyapp.com*.



8. Connect *develop* branch in Amplify Console (assume *develop* and *main* branch are the same at this point). Choose the *test* backend environment.



9. The Amplify Console is now setup. You can start working on new features in a feature branch. Add backend functionality by using the *dev* backend environment from your local workstation.

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

10. After you finish working on the feature, commit your code, create a pull request to review internally.

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

11. To preview what the changes will look like, go to the Console and connect your feature branch. Note: If you have the AWS CLI installed on your system (Not the Amplify CLI), you can connect a branch directly from your terminal. You can find your appid by going to App settings > General > AppARN: *arn:aws:amplify:<region>:<region>:apps/<appid>*

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

12. Your feature will be accessible at *https://newinternet.appid.amplifyapp.com* to share with your teammates. If everything looks good merge the PR to the develop branch.

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

13. This will kickoff a build that will update the backend as well as the frontend in the Amplify Console with a branch deployment at *https://dev.appid.amplifyapp.com*. You can share this link with internal stakeholders so they can review the new feature.

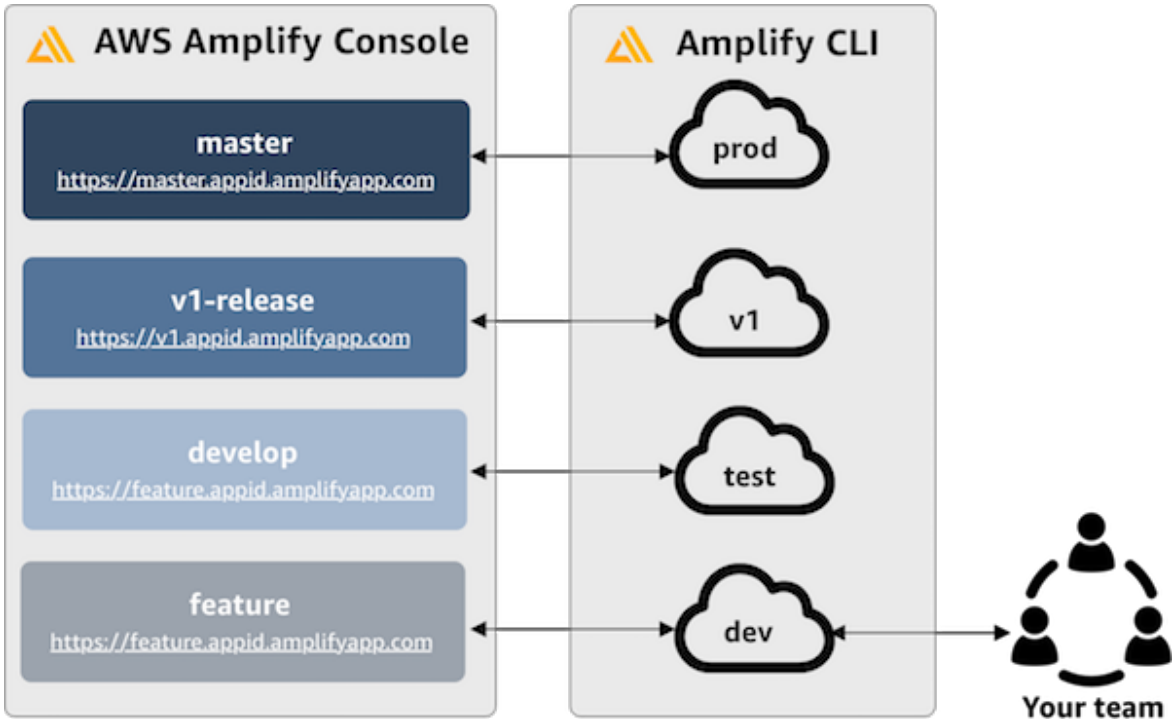
14Delete your feature branch from Git, Amplify Console, and remove the backend environment from the cloud (you can always spin up a new one based on by running 'amplify env checkout prod' and running 'amplify env add').

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

GitFlow workflow

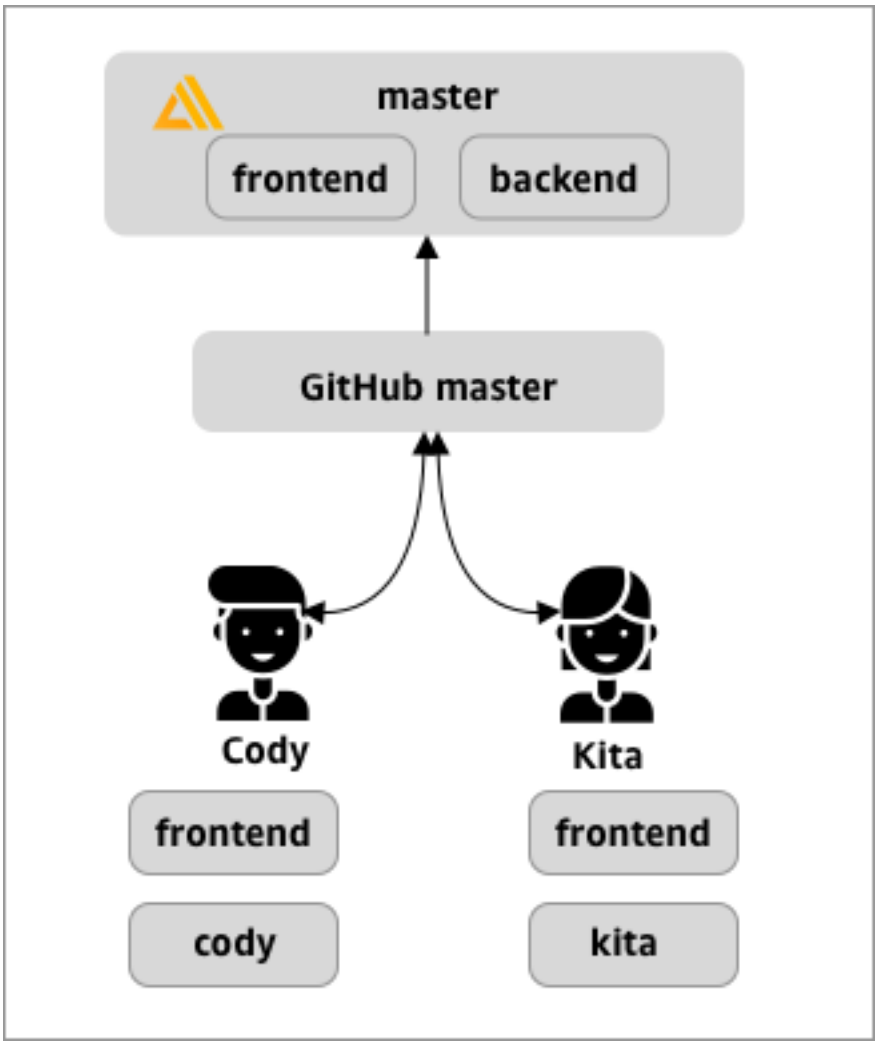
GitFlow uses two branches to record the history of the project. The *main* branch (formerly referred to as master branch) tracks release code only, and the *develop* branch is used as an integration branch for new features. GitFlow simplifies parallel development by isolating new development from completed work. New development (such as features and non-emergency bug fixes) is done in *feature* branches. When the developer is satisfied that the code is ready for release, the *feature* branch is merged back into the integration *develop* branch. The only commits to the main branch are merges from *release* branches and *hotfix* branches (to fix emergency bugs).

The diagram below shows a recommended setup with GitFlow. You can follow the same process as described in the feature branch workflow section above.



Per-developer sandbox

- Each developer in a team creates a sandbox environment in the cloud that is separate from their local computer. This allows developers to work in isolation from each other without overwriting other team members' changes.
- Each branch in the Amplify Console has its own backend. This ensures that the Amplify Console uses the Git repository as a single source of truth from which to deploy changes, rather than relying on developers on the team to manually push their backend or front end to production from their local computers.



1. Install the Amplify CLI to initialize a new Amplify project.

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

2. Initialize a *kita* backend environment for your project. If you don't have a project, create one using bootstrap tools like create-react-app or Gatsby.

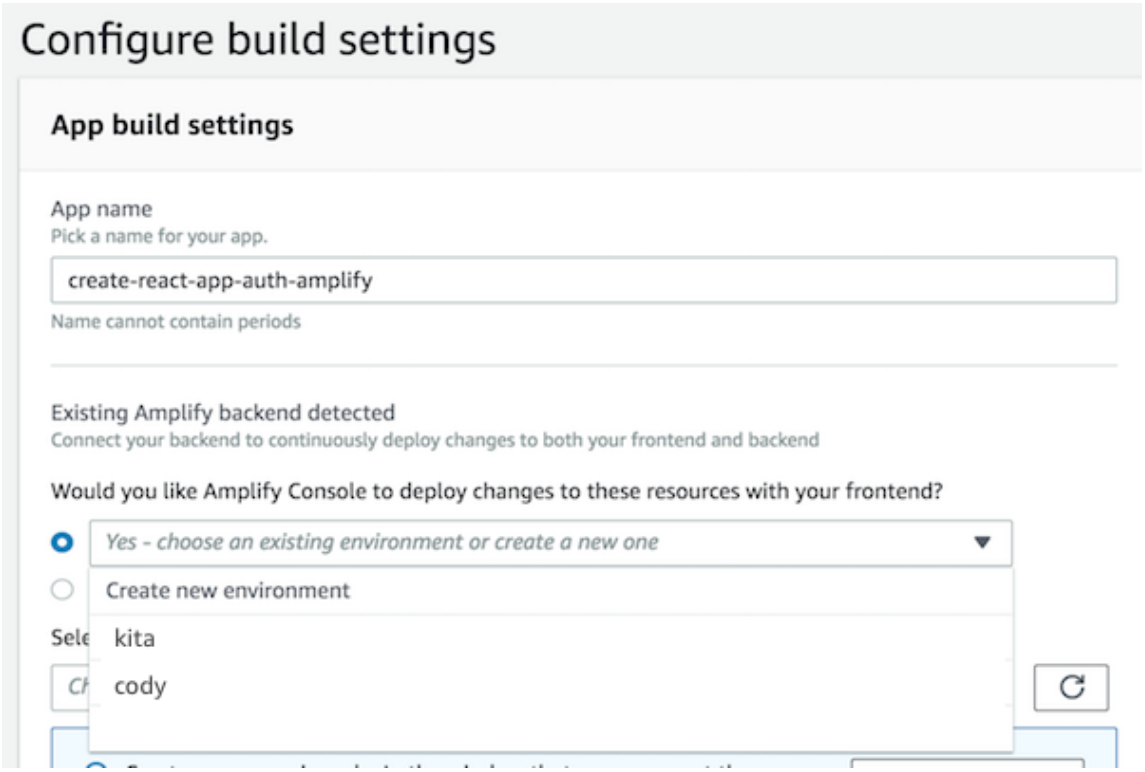
```
cd next-unicorn
amplify init
 ? Do you want to use an existing environment? (Y/n): n
 ? Enter a name for the environment: kita
...
amplify push
```

3. Push code to a Git repository of your choice (in this example we'll assume you pushed to main (formerly referred to as master).

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

4. Connect your repo > *main* to the Amplify Console.

5. The Amplify Console will detect backend environments created by the Amplify CLI. Choose *Create new environment* from the dropdown and grant the service role to Amplify Console. Choose **Save and deploy**. After the build completes you will get a main branch deployment available at *https://main.appid.amplifyapp.com* with a new backend environment that is linked to the branch.

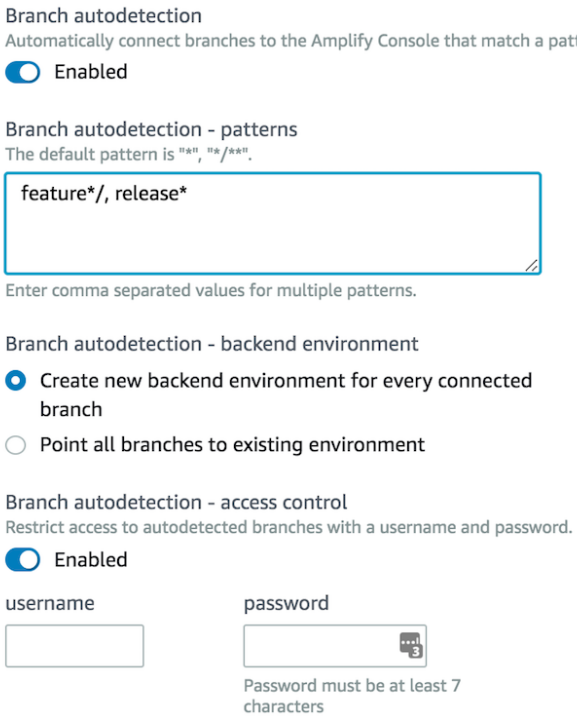


6. Connect *develop* branch in Amplify Console (assume *develop* and *main* branch are the same at this point) and choose *Create new environment*. After the build completes you will get a develop branch deployment available at *https://develop.appid.amplifyapp.com* with a new backend environment that is linked to the branch.

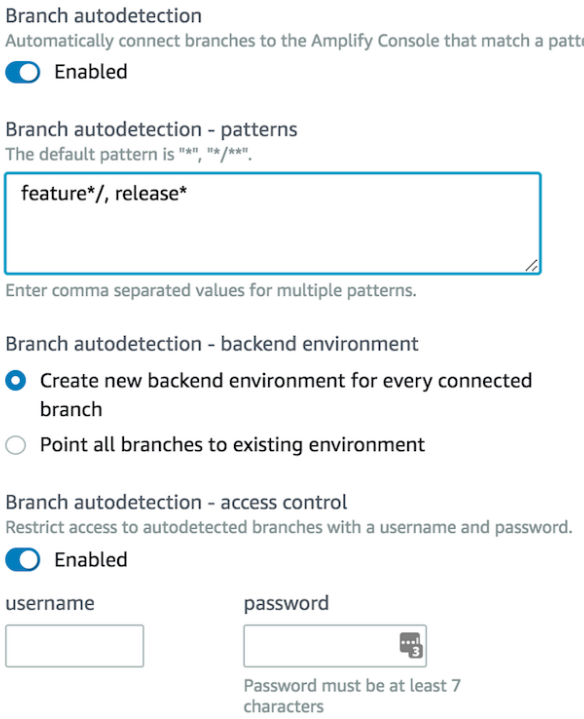
Pattern-based feature branch deployments

Pattern-based branch deployments allow you to automatically deploy branches that match a specific pattern to the Amplify Console. Product teams using feature branch or GitFlow workflows for their releases, can now define patterns such as 'release**' to automatically deploy Git branches that begin with 'release' to a shareable URL. This blog post describes using this feature with different team workflows.

1. Choose **App settings > General > Edit**.
2. Flip the branch autodetection switch to **Enabled**.



1. Define patterns for automatically deploying branches.
   - **`*`** – Deploys all branches in your repository.
   - **`release*`**' – Deploys all branches that being with the word 'release.
   - **`release*/`** – Deploys all branches that match a 'release /' pattern.
   - Specify multiple patterns in a comma-separated list. For example, `release*, feature*`.
2. Set up automatic password protection for all branches that are automatically created by setting **Branch autodetection - access control** to **Enabled**.
3. For applications built with an Amplify backend, you can choose to create a new environment or point all branches to an existing backend.



Pattern-based feature branch deployments for an app connected to a custom domain

You can use pattern-based feature branch deployments for an app connected to an Amazon Route 53 custom domain.

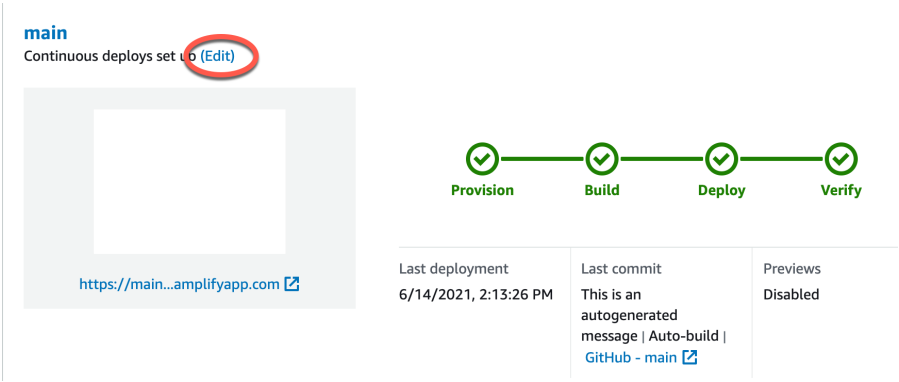
- For instructions on setting up pattern-based feature branch deployments, see Set up automatic subdomains for a Amazon Route 53 custom domain (p. 32)
- For instructions on connecting an Amplify app to a custom domain managed in Route 53, see Add a custom domain managed by Amazon Route 53 (p. 24)
- For more information about using Route 53, see What is Amazon Route 53.

Automatic build-time generation of Amplify config

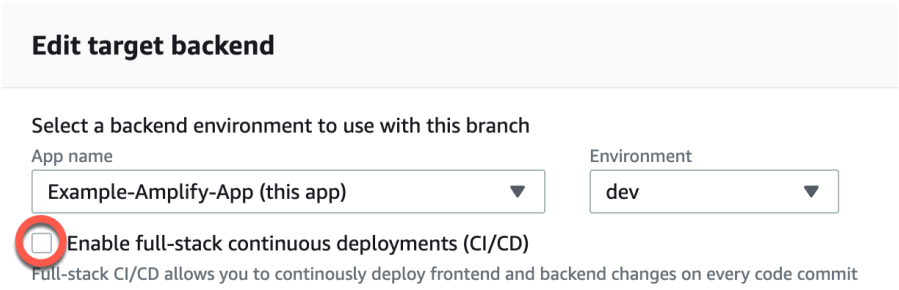
Amplify now supports the automatic build-time generation of the Amplify config `aws-exports.js` file. By turning off full stack CI/CD deployments, you enable your app to autogenerate the `aws-exports.js` file and ensure that updates are not made to your backend at build-time.

To autogenerate `aws-exports.js` at build-time

1. Sign in to the AWS Management Console and open the Amplify Console.
2. Choose the app to edit.
3. Choose the **Frontend environments** tab.
4. Locate the branch to edit and choose **Edit**.



5. On the **Edit target background** page, uncheck **Enable full-stack continuous deployments (CI/CD)** to turn off full-stack CI/CD for this backend.



6. Select an existing service role to give Amplify the permissions it requires to make changes to your app backend. If you need to create a service role, choose **Create new role**. For more information about creating a service role, see Adding a service role to the Amplify Console when you connect an app (p. 100).
7. Choose **Save**. The Amplify Console applies these changes the next time you build the app.

Conditional backend builds

Amplify now supports conditional backend builds on all branches in an app. To configure conditional backend builds, set the `AMPLIFY_DIFF_BACKEND` environment variable to `true`. Enabling conditional backend builds will help speed up builds where changes are made only to the frontend.

When you enable diff based backend builds, at the start of each build, Amplify attempts to run a diff on the `amplify` folder in your repository. If Amplify doesn't find any differences, it skips the backend build step, and doesn't update your backend resources. If your project doesn't have an `amplify` folder in your repository, Amplify ignores the value of the `AMPLIFY_DIFF_BACKEND` environment variable. For instructions on setting the `AMPLIFY_DIFF_BACKEND` environment variable, see Enable or disable diff based backend builds (p. 40).

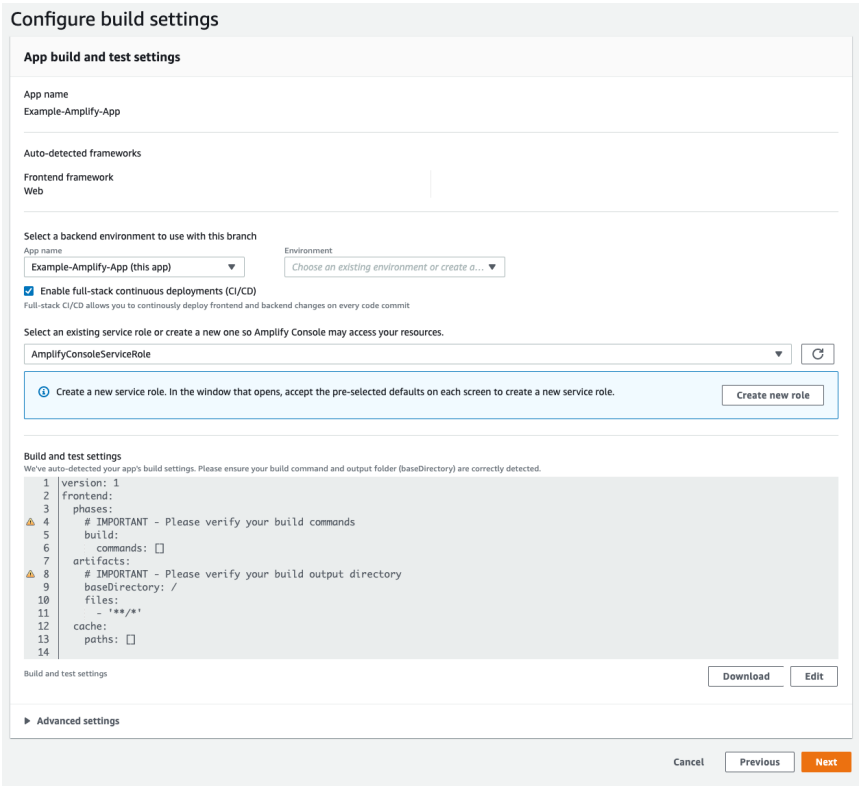
Use Amplify backends across apps

Amplify Console now enables you to easily reuse existing backend environments across all of your apps in a given region. You can do this when you create a new app, connect a new branch to an existing app, or update an existing frontend to point to a different backend environment.

Reuse backends when creating a new app

To reuse a backend when creating a new Amplify app

1. Sign in to the AWS Management Console and open the Amplify Console.
2. To create a new backend to use for this example, do the following:
   a. In the navigation pane, choose **All apps**.
   b. Choose **New app**, **Create app backend**.
   c. Enter a name for your app, such as **`Example-Amplify-App`**.
   d. Choose **Confirm deployment**.
3. To connect a frontend to your new backend, choose the **Frontend environments** tab.
4. Choose your git provider, and then choose **Connect branch**.
5. On the **Add repository branch** page, for **Recently updated repositories**, choose your repository name. For **Branch**, select the branch from your repository to connect.
6. On the **Configure build settings page**, do the following:
   a. For **App name**, select the app to use for adding a backend environment. You can choose the current app or any other app in the current region.
   b. For **Environment**, select the name of the backend environment to add. You can use an existing environment or create a new one.
   c. Select an existing service role to give Amplify the permissions it requires to make changes to your app backend. If you need to create a service role, choose **Create new role**. For more information about creating a service role, see Adding a service role to the Amplify Console when you connect an app (p. 100).
   d. By default, full-stack CI/CD is enabled. Uncheck this option to turn off full-stack CI/CD for this backend. Turning off full-stack CI/CD causes the app to run in *pull only* mode. At build time, Amplify will automatically generate the `aws-exports.js` file only, without modifying your backend environment.
   e. Choose **Next**.

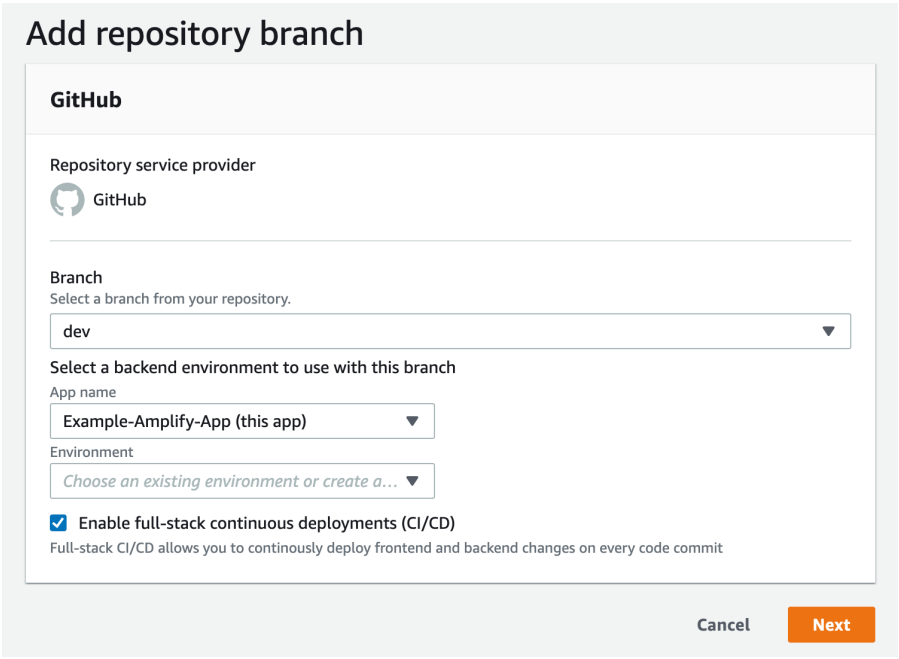


7. Choose **Save and deploy**.

Reuse backends when connecting a branch to an existing app

To reuse a backend when connecting a branch to an existing Amplify app

1. Sign in to the AWS Management Console and open the Amplify Console.
2. Choose the app to connect a new branch to.
3. In the navigation pane, choose **App Settings**, **General**.
4. In the **Branches** section, choose **Connect a branch**.
5. On the **Add repository branch** page, for **Branch**, select the branch from your repository to connect.
6. For **App name**, select the app to use for adding a backend environment. You can choose the current app or any other app in the current region.
7. For **Environment**, select the name of the backend environment to add. You can use an existing environment or create a new one.
8. If you need to set up a service role to give Amplify the permissions it requires to make changes to your app backend, the console prompts you to perform this task. For more information about creating a service role, see Adding a service role to the Amplify Console when you connect an app (p. 100).
9. By default, full-stack CI/CD is enabled. Uncheck this option to turn off full-stack CI/CD for this backend. Turning off full-stack CI/CD causes the app to run in *pull only* mode. At build time, Amplify will automatically generate the `aws-exports.js` file only, without modifying the backend environment.
10. Choose **Next**.

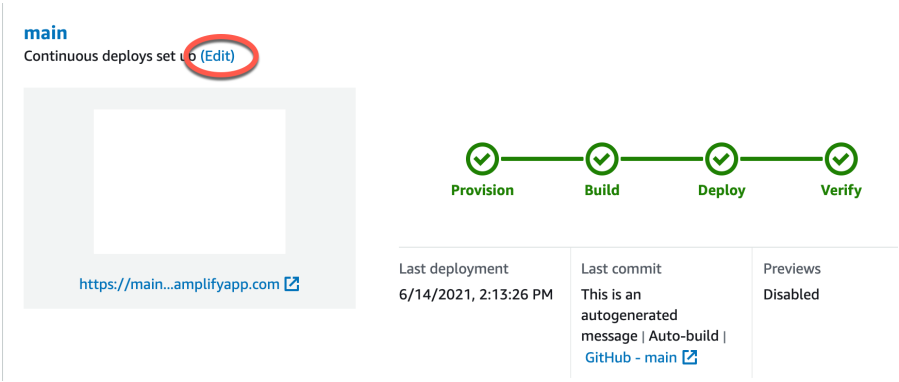


11. Choose **Save and deploy**.

Edit an existing frontend to point to a different backend

To edit a frontend Amplify app to point to a different backend

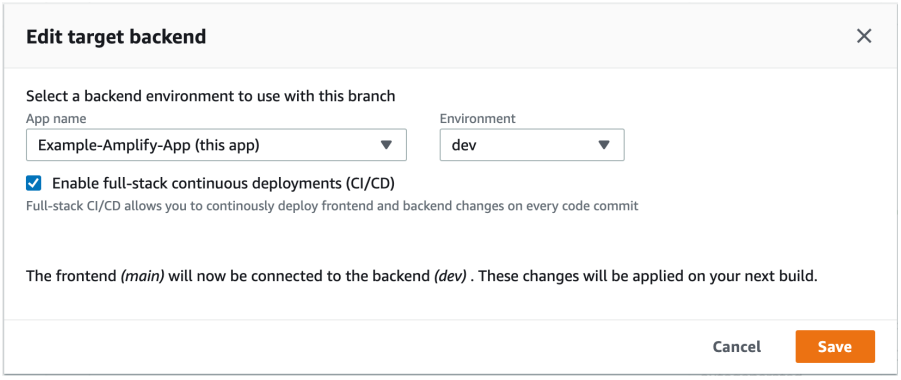
1. Sign in to the AWS Management Console and open the Amplify Console.
2. Choose the app to edit the backend for.
3. Choose the **Frontend environments** tab.
4. Locate the branch to edit and choose **Edit**.



5. On the **Edit target background** page, for **App name**, select the app to use for adding a backend environment. You can choose the current app or any other app in the current region.
6. For **Environment**, select the name of the backend environment to add.
7. By default, full-stack CI/CD is enabled. Uncheck this option to turn off full-stack CI/CD for this backend. Turning off full-stack CI/CD causes the app to run in *pull only* mode. At build time,

Amplify will automatically generate the `aws-exports.js` file only, without modifying the backend environment.

8. Choose **Save**. The Amplify Console applies these changes the next time you build the app.



Manual deploys

Manual deploys allows you to publish your web app to the Amplify Console without connecting a Git provider. You can choose to drag and drop a folder from your desktop and host your site in seconds. Alternatively, you can reference assets in an Amazon S3 bucket. You can also specify a public URL to the location where your files are stored.

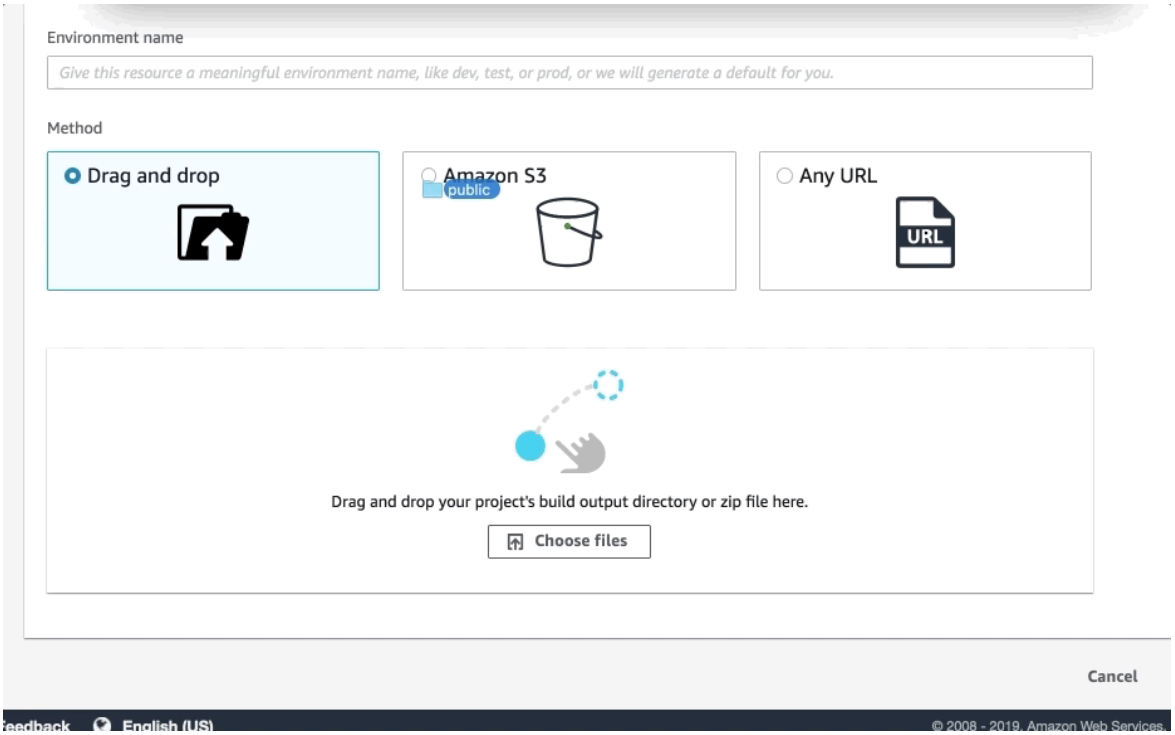
For Amazon S3, you can also set up AWS Lambda triggers to update your site each time new assets are uploaded. This blog post describes the process for setting up a Lambda trigger to automatically deploy changes to Amplify when updates are made to an Amazon S3 bucket.

Amplify does not support manual deploys for server-side rendered (SSR) apps. For more information, see Deploy and host server-side rendered apps with Amplify (p. 14).

Drag and drop

To manually deploy an app using drag and drop

1. Sign in to the AWS Management Console and open the Amplify Console.
2. How you get to the **Host your web app** page depends on whether you are starting from the Amplify home page or the **All apps** page.
   - From the Amplify home page
     a. Choose **Get started**.
     b. In the **Deliver** section, choose **Get started**.
   - From the **All apps** page
     - In the upper right corner, choose **New app**, **Host web app**
3. On the **Host your web app** page, choose **Deploy without Git provider**. Then, choose **Continue**.
4. In the **Start a manual deployment** section, for **App name**, enter the name of your app.
5. For **Environment name**, enter a meaningful name for the environment, such as **`development`** or **`production`**.
6. For **Method**, choose **Drag and drop**.
7. Either drag and drop files from your desktop onto the drop zone or use **Choose files** to select the files from your computer. The files that you drag and drop or select can be a folder or a zip file that contains the root of your site.
8. Choose **Save and deploy**.



Amazon S3 or any URL

To manually deploy an app from Amazon S3 or a public URL

1. Sign in to the AWS Management Console and open the Amplify Console.
2. At the top of the page, choose **Get started**.
3. In the **Deliver** section, choose **Get started**.
4. On the **Host your web app** page, choose **Deploy without Git provider**. Then, choose **Continue**.
5. In the **Start a manual deployment** section, for **App name**, enter the name of your app.
6. For **Environment name**, enter a meaningful name for the environment, such as **`development`** or **`production`**.
7. For **Method**, choose either **Amazon S3** or **Any URL**.
8. The procedure for uploading your files depends on the upload method.
   - Amazon S3
     a. For **Bucket**, select the name of the bucket from the list.
     b. For **Zip file**, select the name of the zip file to deploy.
   - Any URL
     - For **Resource URL**, enter the URL to the zipped file to deploy.
9. Choose **Save and deploy**.

Note
When you create the zip folder, make sure you zip the contents of your build output and not the top level folder. For example, if your build output generates a folder named “build” or “public”, first navigate into that folder, select all of the contents, and zip it from there. If you do not do

this, you will see an “Access Denied” error because the site's root directory will not be initialized properly.

```
<Error>
  <Code>AccessDenied</Code>
  <Message>Access Denied</Message>
  <RequestId>4442587FB7D0A2F9</RequestId>
  <HostId>...</HostId>
</Error>
```

Deploy to Amplify Console button

The **Deploy to Amplify Console** button enables you to share GitHub projects publicly or within your team. The following is an image of the button:

Add 'Deploy to Amplify Console' button to your repository or blog

Add this button to your GitHub README.md file, blog post, or any other markup page that renders HTML. The button has the following two components:

1. An SVG image: `https://oneclick.amplifyapp.com/button.svg`
2. The Amplify Console URL with a link to your GitHub repository. Copy your repo URL (e.g. `https://github.com/username/repository`) only or provide a deep link into a specific folder (e.g. `https://github.com/username/repository/tree/branchname/folder`). The Amplify Console will deploy the default branch in your repository. Additional branches can be connected after the app is connected.

To add the button to a markdown file (e.g. your GitHub README.md), replace `https://github.com/username/repository` with your repository name.

```
[![amplifybutton](https://oneclick.amplifyapp.com/button.svg)](https://
console.aws.amazon.com/amplify/home#/deploy?repo=https://github.com/username/repository)
```

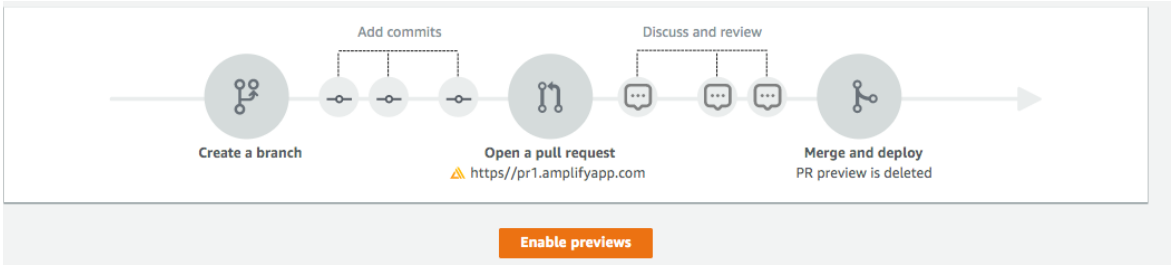
To add the button to any HTML document, use the following html example:

```
<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 Console">
</a>
```

Web previews

Web previews offer development and quality assurance (QA) teams a way to preview changes from pull requests (PRs) before merging code to a production or integration branch. Pull requests let you tell others about changes you've pushed to a branch in a repository. After a pull request is opened, you can discuss and review the potential changes with collaborators and add follow-up commits before your changes are merged into the base branch.

A web preview deploys every pull request made to your GitHub repository to a unique preview URL which is completely different from the URL your main site uses. For apps with backend environments provisioned using the Amplify CLI, every pull request (**private Git repositories only**) spins up an ephemeral backend that is deleted when the PR is closed.



Note
Previews is visible in the Amplify Console's App settings menu only when an app is set up for continuous deployment and connected to a git repository. For instructions on this type of deployment, see Getting started with existing code (p. 3).

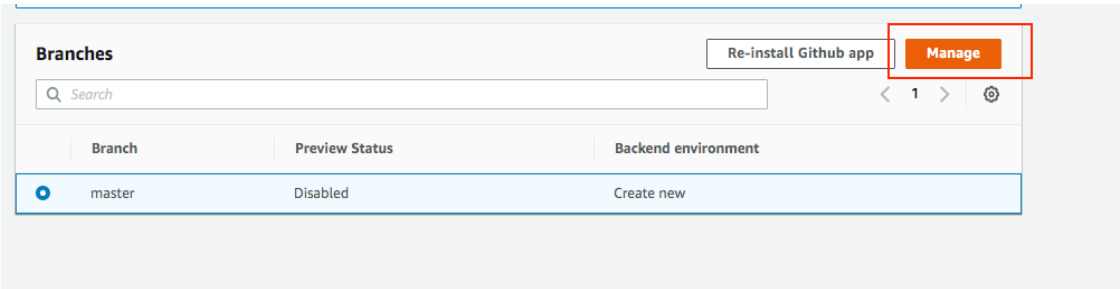
Enable web previews

To enable web previews for pull requests

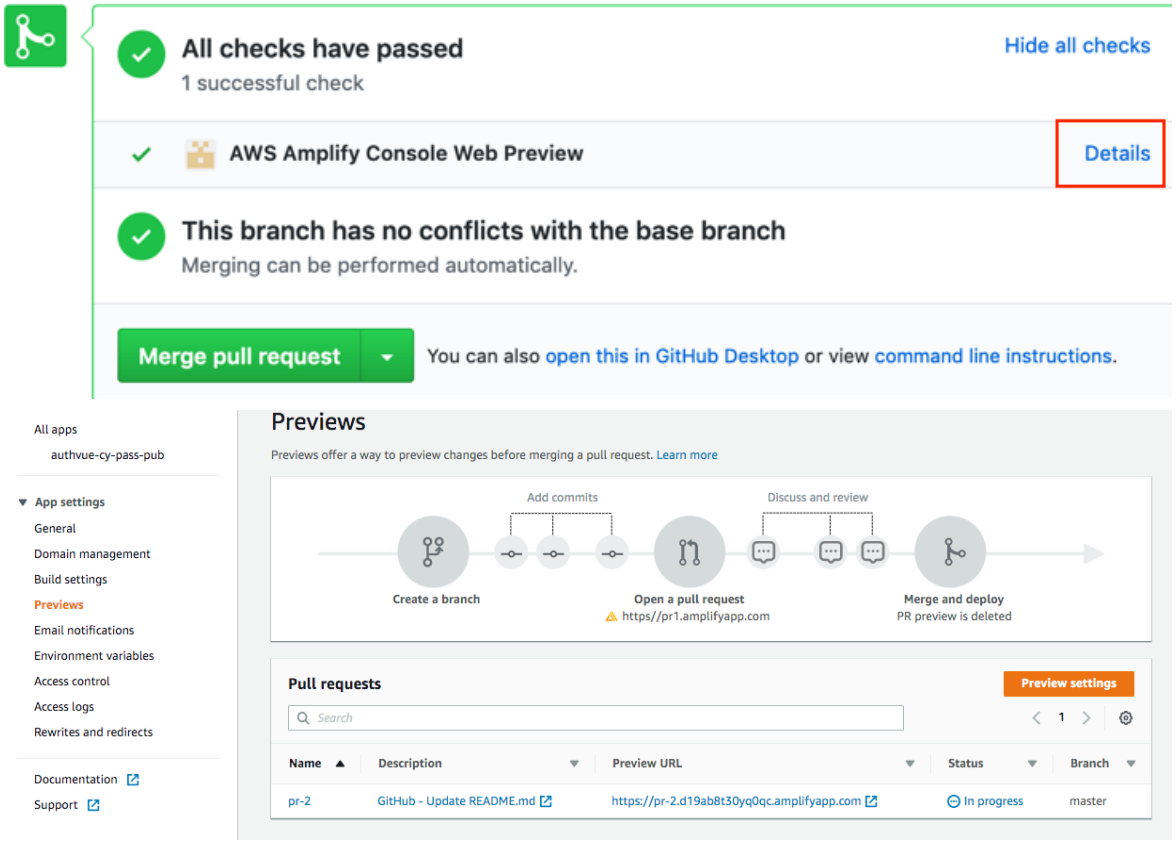
1. Choose **App settings**, **Previews** and then choose **Enable previews**. For GitHub repositories only, you are required to install a GitHub app in your account to enable this feature. You can give the Amplify Console permission to all repositories or the current repository only.

   Important
   For security purposes, previews only work with private repositories for fullstack apps using the Amplify CLI.

2. After you enable previews, return to the Amplify Console to enable previews for specific branches. Choose a branch from the list and choose **Manage**. For fullstack applications, you can choose to create a new backend for every pull request, or point all pull requests to an existing backend environment. By choosing the first option, you can test changes without impacting production.



The next time you submit a pull request for the branch, the Amplify Console builds and deploys your PR to a preview URL. For GitHub repositories only, you can view a preview of your URL directly from the pull request.



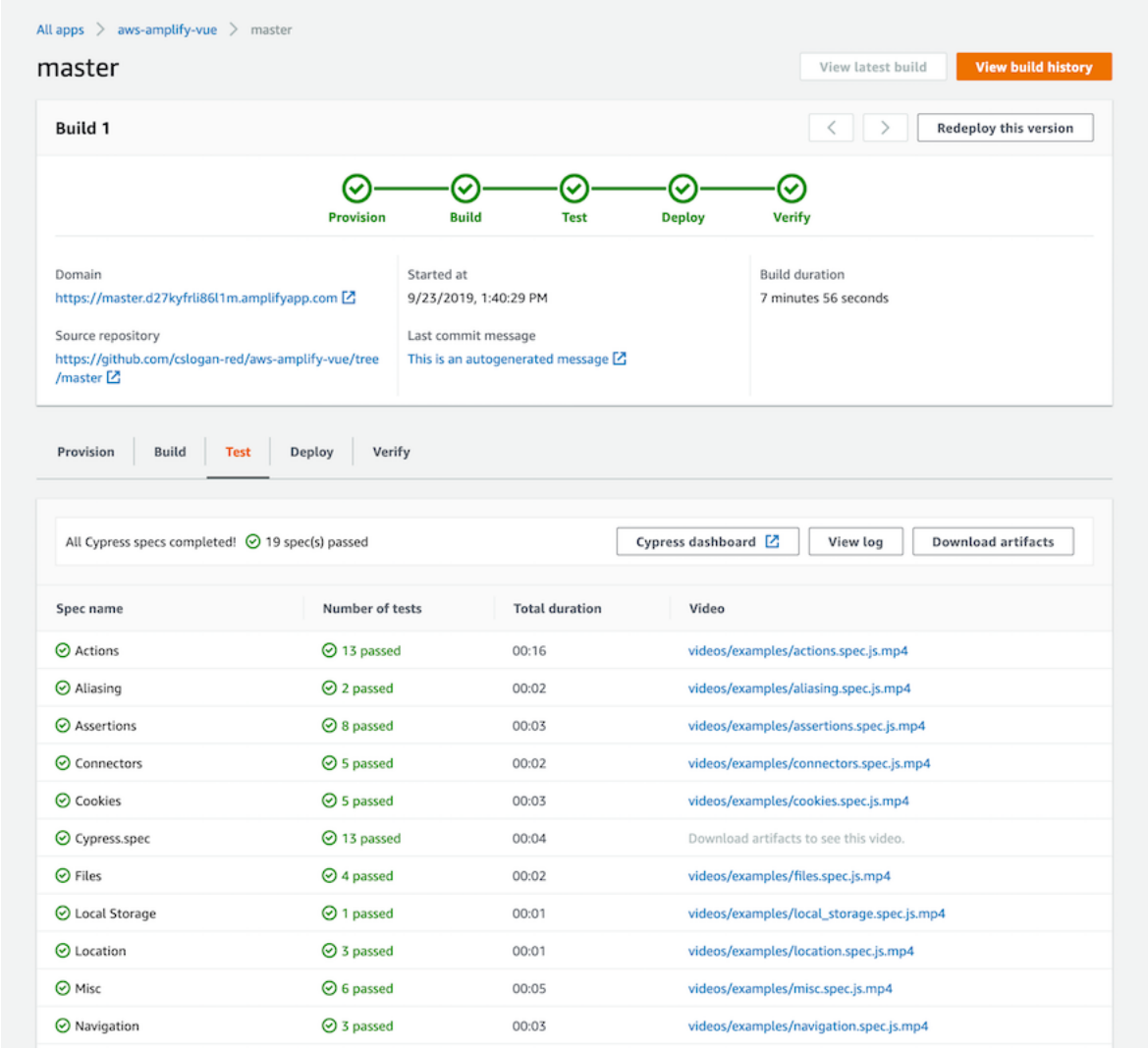
After the pull request is closed, the preview URL is deleted, and any temporary backend environment linked to the pull request is deleted.

Web preview access with subdomains

Web previews from pull requests are accessible with subdomains for an Amplify app that is connected to a custom domain managed by Amazon Route 53 . When the pull request is closed, branches and subdomains associated with the pull request are automatically deleted. This is the default behavior for web previews after you set up pattern-based feature branch deployments for your app. For instructions on setting up automatic subdomains, see Set up automatic subdomains for a Amazon Route 53 custom domain (p. 32).

Add end-to-end tests to your Amplify app

You can run end-to-end (E2E) tests in the test phase of your Amplify app to catch regressions before pushing code to production. The test phase can be configured in the build specification YML and can be used to run any testing framework of your choice during a build.



Tutorial: Set up end-to-end tests with Cypress

Cypress is a JavaScript-based framework that allows you to run E2E tests on a browser. This tutorial will demonstrate how to set up E2E tests from scratch.

Add tests to your existing Amplify app

You can use the test step to run any test commands at build time. For E2E tests, the Amplify Console offers a deeper integration with Cypress that allows you to generate a UI report for your tests. To add Cypress tests to an existing app, update your amplify.yml build settings with the following values.

```
test:
  phases:
    preTest:
      commands:
        - npm ci
        - npm install wait-on
        - npm install pm2
        - npm install mocha@5.2.0 mochawesome mochawesome-merge mochawesome-report-
generator
        - npx pm2 start npm -- start
        - 'npx 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
        - npx pm2 kill
  artifacts:
    baseDirectory: cypress
    configFilePath: '**/mochawesome.json'
    files:
      - '**/*.png'
      - '**/*.mp4'
```

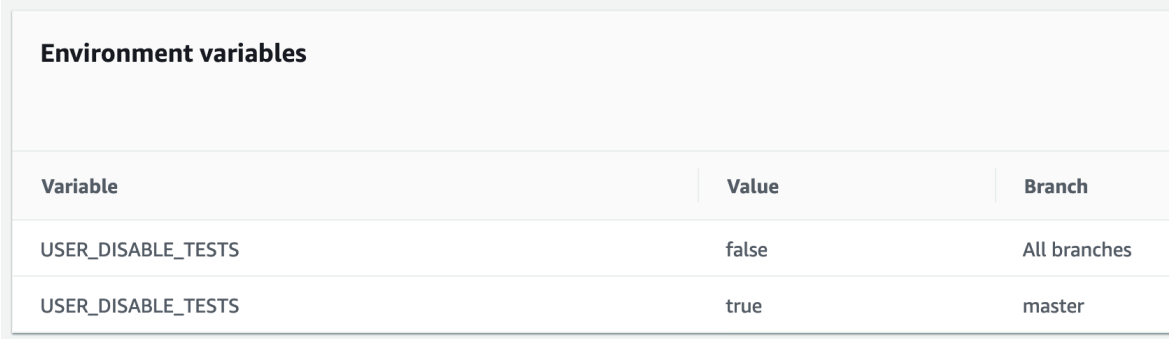
- **preTest** - Install all the dependencies required to run Cypress tests. Amplify Console uses mochaawesome to generate a report to view your test results and wait-on to set up the localhost server during the build.
- **test** - Run cypress commands to execute tests using mochawesome.
- **postTest** - The mochawesome report is generated from the output JSON.
- **artifacts>baseDirectory** - The directory from which tests are run.
- **artifacts>configFilePath** - The generated test report data.
- **artifacts>files** - The generated artifacts (screenshots and videos) available for download.

Disabling tests

Once the “test” config has been added to your amplify.yml build settings, the test step will be executed for every build, on every branch. If you would like to globally disable tests from running, or you would only like tests to run for specific branches, you can use the “USER_DISABLE_TESTS” environment variable to do so without modifying your build settings.

To **globally** disable tests for all branches, add the USER_DISABLE_TESTS environment variable with a value of true for all branches, as shown below:

To disable tests for a **specific branch**, add the USER_DISABLE_TESTS environment variable with a value of false for all branches, and then add an override for each branch you would like to disable with a value of true. In the following example, tests are disabled on the “main” branch, and enabled for every other branch:



Disabling tests with this variable will cause the test step to be skipped altogether during a build. To re-enable tests, set this value to *false*, or delete the environment variable.

Using redirects

Redirects enable a web server to reroute navigation from one URL to another. Common reasons for using redirects include: to customize the appearance of a URL, to avoid broken links, to move the hosting location of an app or site without changing its address, and to change a requested URL to the form needed by a web app.

Types of redirects

There are several types of redirects that support specific scenarios.

Permanent redirect (301)

301 redirects are intended for lasting changes to the destination of a web address. Search engine ranking history of the original address applies to the new destination address. Redirection occurs on the client-side, so a browser navigation bar shows the destination address after redirection. Common reasons to use 301 redirects include:

- To avoid a broken link when the address of a page changes.
- To avoid a broken link when a user makes a predictable typo in an address.

Temporary redirect (302)

302 redirects are intended for temporary changes to the destination of a web address. Search engine ranking history of the original address doesn't apply to the new destination address. Redirection occurs on the client-side, so a browser navigation bar shows the destination address after redirection. Common reasons to use 302 redirects include:

- To provide a detour destination while repairs are made to an original address.
- To provide test pages for A/B comparison of user interface.

Rewrite (200)

200 redirects (rewrites) are intended to show content from the destination address as if it were served from the original address. Search engine ranking history continues to apply to the original address. Redirection occurs on the server-side, so a browser navigation bar shows the original address after redirection. Common reasons to use 200 redirects include:

- To redirect an entire site to a new hosting location without changing the address of the site.
- To redirect all traffic to a single page web app (SPA) to its index.html page for handling by a client-side router function.

Not Found (404)

404 redirects occur when a request points to an address that doesn't exist. The destination page of a 404 is displayed instead of the requested one. Common reasons a 404 redirect occurs include:

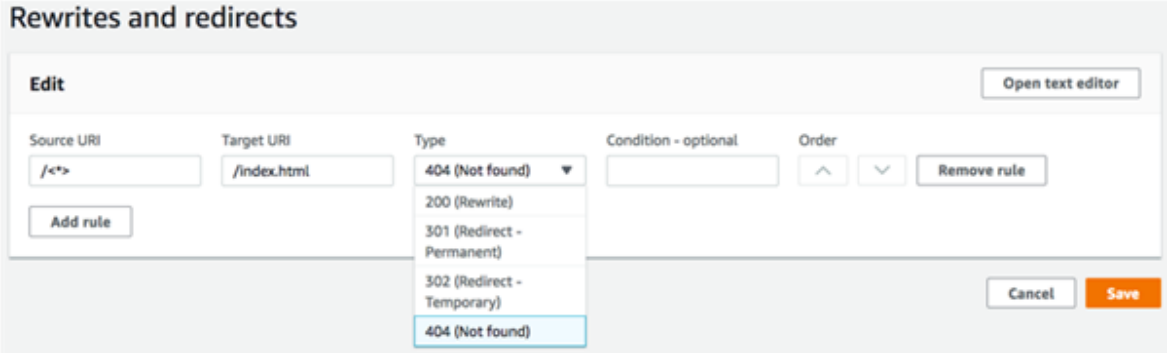
- To avoid a broken link message when a user enters a bad URL.
- To point requests to nonexistent pages of a web app to its index.html page for handling by a client-side router function.

Parts of a redirect

Redirects consist of the following:

- An original address - The address the user requested.
- A destination address - The address that actually serves the content that the user sees.
- A redirect type - Types include a permanent redirect (301), a temporary redirect (302), a rewrite (200), or not found (404).
- A two letter country code (optional) - a value you can include to segment the user experience of your app by region.

To create and edit redirects, choose **Rewrites and redirects settings** in the left navigation pane.



To bulk edit redirects in a JSON editor, choose **Open text editor**.

Order of redirects

Redirects are executed from the top of the list down. Make sure that your ordering has the effect you intend. For example, the following order of redirects causes all requests for a given path under */docs/* to redirect to the same path under */documents/*, except */docs/specific-filename.html* which redirects to */documents/different-filename.html*:

```
/docs/specific-filename.html /documents/different-filename.html 301
/docs/<*> /documents/<*>
```

The following order of redirects ignores the redirection of *specific-filename.html* to *different-filename.html*:

```
/docs/<*> /documents/<*>
/docs/specific-filename.html /documents/different-filename.html 301
```

Simple redirects and rewrites

In this section we include example code for common redirect scenarios.

You can use the following example code to permanently redirect a specific page to a new address.

Original address	Destination Address	Redirect Type	Country Code
`/original.html`	`/destination.html`	`permanent redirect (301)`	

JSON: [{"source": "/original.html", "status": "301", "target": "/destination.html", "condition": null}]

You can use the following example code to redirect any path under a folder to the same path under a different folder.

Original address	Destination Address	Redirect Type	Country Code
`docs/<*>`	`/documents/<*>`	`permanent redirect (301)`	

JSON [{"source": "/docs/<*>", "status": "301", "target": "/documents/<*>", "condition": null}]

You can use the following example code to redirect all traffic to index.html as a rewrite. In this scenario, the rewrite makes it appear to the user that they have arrived at the original address.

Original address	Destination Address	Redirect Type	Country Code
`<*>`	`/index.html`	`rewrite (200)`	

JSON [{"source": "/<*>", "status": "200", "target": "/index.html", "condition": null}]

You can use the following example code to use a rewrite to change the subdomain that appears to the user.

Original address	Destination Address	Redirect Type	Country Code
`https://mydomain.com`	`https://www.mydomain.com`	`rewrite (200)`	

JSON [{"source": "https://mydomain.com", "status": "200", "target": "https://www.mydomain.com", "condition": null}]

You can use the following example code to redirect paths under a folder that can't be found to a custom 404 page.

Original address	Destination Address	Redirect Type	Country Code
`/<*>`	`/404.html`	`not found (404)`	

JSON [{"source": "/<*>", "status": "404", "target": "/404.html", "condition": null}]

Redirects for single page web apps (SPA)

Most SPA frameworks support HTML5 history.pushState() to change browser location without triggering a server request. This works for users who begin their journey from the root (or */index.html*), but fails for users who navigate directly to any other page. Using regular expressions, the following example sets up a 200 rewrite for all files to index.html except for the specific file extensions specified in the regular expression.

Original address	Destination Address	Redirect Type	Country Code
`</^[^.]+$\|\.(?!(css\|gif\|ico\|jpg\|js\|png\|txt\|svg\|woff\|woff2\|ttf\|map\|json)$)([^.]+$)/>`	`/index.html`	`200`	

JSON [{"source": "</^[^.]+$|\.(?!(css|gif|ico|jpg|js|png|txt|svg|woff|ttf|map|json)$)([^.]+$)/>", "status": "200", "target": "index.html", "condition": null}]

Reverse proxy rewrite

The following example uses a rewrite to proxy content from another location so that it appears to user that the domain hasn't changed:

Original address	Destination Address	Redirect Type	Country Code
`/images`	`https://images.otherdomain.com`	`rewrite (200)`	

JSON [{"source": "/images", "status": "200", "target": "https://images.otherdomain.com", "condition": null}]

Trailing slashes and clean URLs

To create clean URL structures like *about* instead of *about.html*, static site generators such as Hugo generate directories for pages with an index.html (*/about/index.html*). The Amplify Console automatically creates clean URLs by adding a trailing slash when required. The table below highlights different scenarios:

User inputs in browser	URL in the address bar	Document served
`/about`	`/about`	`/about.html`
`/about (when about.html returns 404)`	`/about/`	`/about/index.html`

User inputs in browser	URL in the address bar	Document served
/about/	/about/	/about/index.html

Placeholders

You can use the following example code to redirect paths in a folder structure to a matching structure in another folder.

Original address	Destination Address	Redirect Type	Country Code
/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}]

Query strings and path parameters

You can use the following example code to redirect a path to a folder with a name that matches the value of a query string element in the original address:

Original address	Destination Address	Redirect Type	Country Code
/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-post-id-value>", "condition": null}]

You can use the following example code to redirect all paths that can't be found at a given level of a folder structure to index.html in a specified folder.

Original address	Destination Address	Redirect Type	Country Code
/documents/<folder>/<child-folder>/<grand-child-folder>	/documents/index.html	404	

JSON [{"source": "/documents/<x>/<y>/<z>", "status": "404", "target": "/documents/index.html", "condition": null}]

Region-based redirects

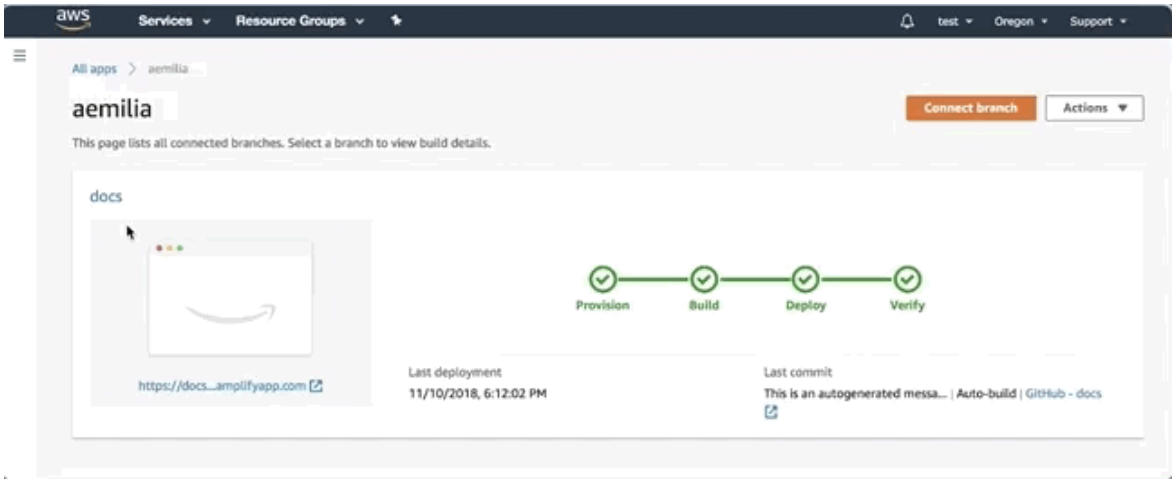
You can use the following example code to redirect requests based on region.

Original address	Destination Address	Redirect Type	Country Code
/documents	/documents/us/	302	<US>

JSON [{"source": "/documents", "status": "302", "target": "/documents/us/", "condition": "<US>"}]

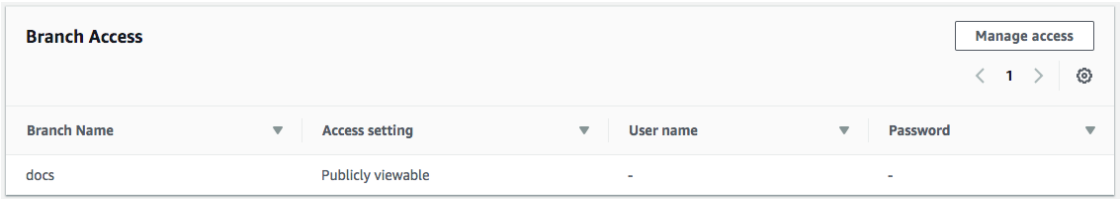
Restricting access

If you are working on unreleased features you can password protect feature branches that are not ready to be accessed publicly.

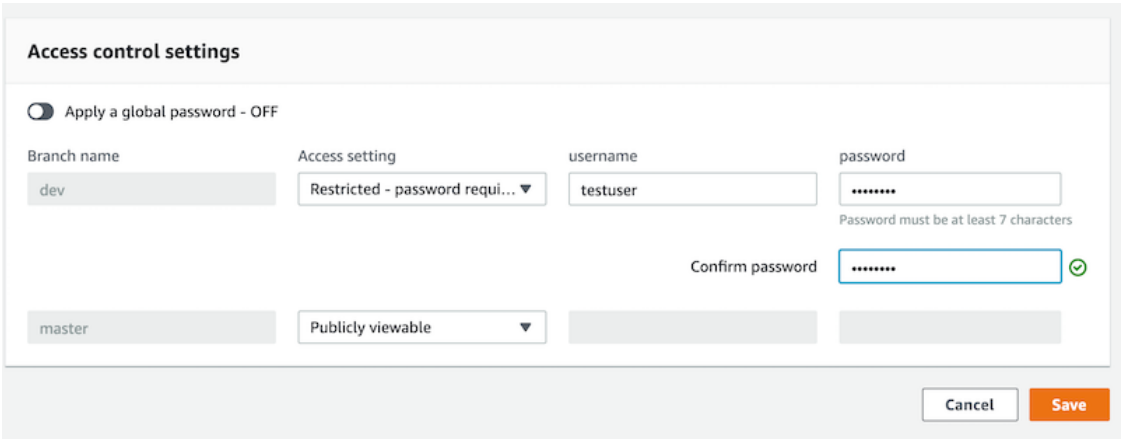


To set passwords on feature branches

1. Sign in to the AWS Management Console and open the Amplify Console.
2. Choose the app you want to set feature branch passwords on.
3. In the navigation pane, choose **App settings**, and then choose **Access control**.
4. In the **Access control settings** section, choose **Manage access**.



5. Do one of the following in **Access control settings**:
   - To set a username and password that applies to all connected branches, turn on **Apply a global password**. For example, if you have **main**, **dev**, and **feature** branches connected, you can use a global password to set the same username and password for all branches.
   - To apply a username and password to an individual branch, turn off **Apply a global password**. For the branch that you want to enter a unique username and password for, choose **Restricted-password required** for **Access setting** and enter a username and password.


Access control settings
Apply a global password - OFF
Branch name
Access setting
username
password
dev
Restricted - password requi...
testuser
Password must be at least 7 characters
Confirm password
master
Publicly viewable
Cancel
Save

Environment variables

Environment variables are key-value pairs that are available at build time. These configurations can be anything, including the following:

- Database connection details
- Third-party API keys
- Different customization parameters
- Secrets

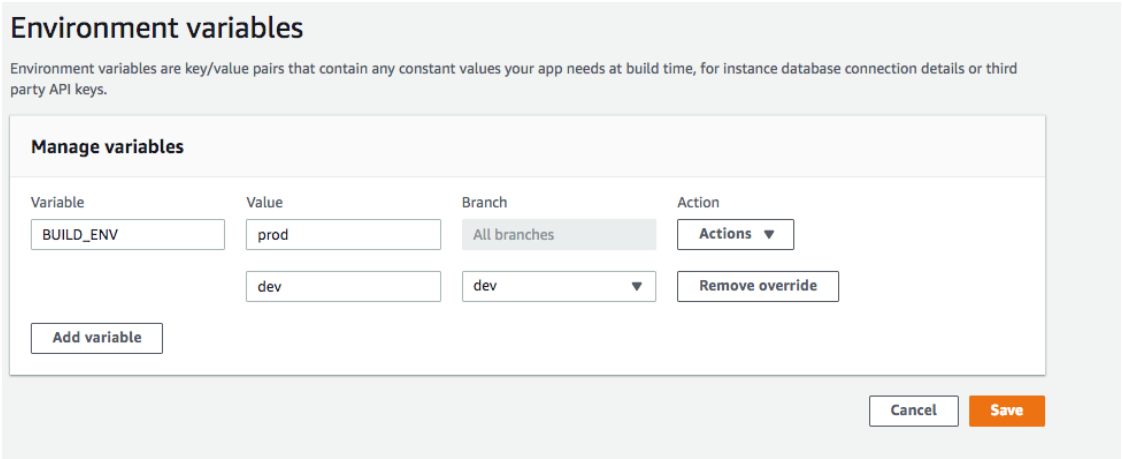
As a best practice, you can use environment variables to expose these configurations. All environment variables that you add are encrypted to prevent rogue access, so you can use them to store secret information.

Note
Environment variables is visible in the Amplify Console's **App settings** menu only when an app is set up for continuous deployment and connected to a git repository. For instructions on this type of deployment, see Getting started with existing code (p. 3).

Set environment variables

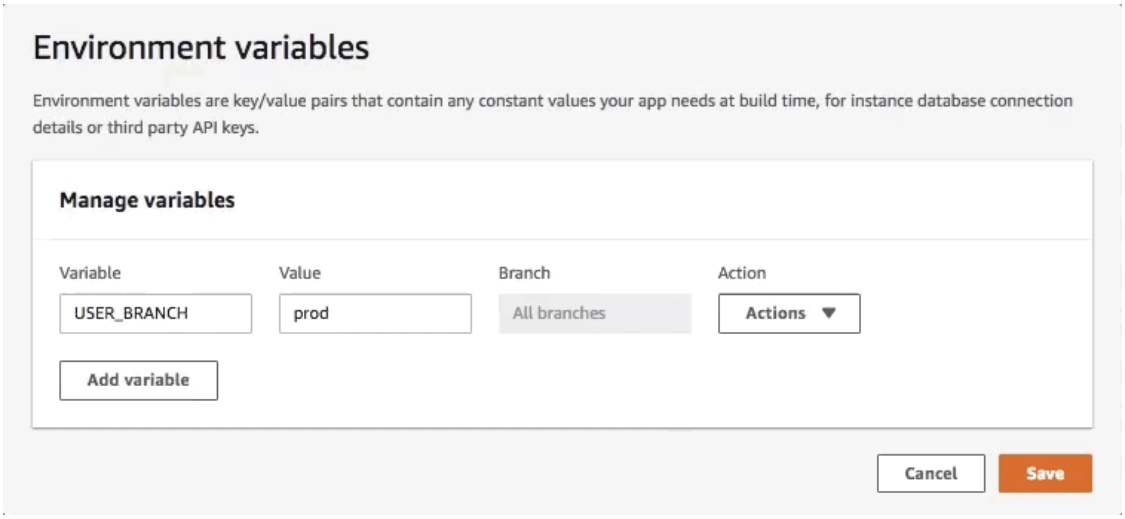
To set environment variables

1. Sign in to the AWS Management Console and open the Amplify Console.
2. In the Amplify Console, choose **App Settings**, and then choose **Environment variables**.
3. In the **Environment variables** section, choose **Manage variables**.
4. In the **Manage variables** section, under **Variable**, enter your key. For **Value**, enter your value. By default, the Amplify console applies the environment variables across all branches, so you don't have to re-enter variables when you connect a new branch.



5. (Optional) To customize an environment variable specifically for a branch, add a branch override as follows:

   a. Choose **Actions** and then choose **Add variable override**.
   b. You now have a set of environment variables specific to your branch.



6. Choose **Save**.

Access environment variables

To access an environment variable during a build, edit your build settings to include the environment variable in your build commands.

To edit build settings to include an environment variable

1. Sign in to the AWS Management Console and open the Amplify Console.
2. In the Amplify Console, choose **App Settings**, then choose **Build settings**.
3. In the **App build specification** section, choose **Edit**.
4. Add the environment variable to your build command. You should now be able to access your environment variable during your next build. This example changes the npm's behavior (`BUILD_ENV`) and adds an API token (`TWITCH_CLIENT_ID`) for an external service to an environment file for later use:

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

Each command in your build configuration is executed inside a Bash shell. For more information on working with environment variables in Bash, see Shell Expansions in the GNU Bash Manual.

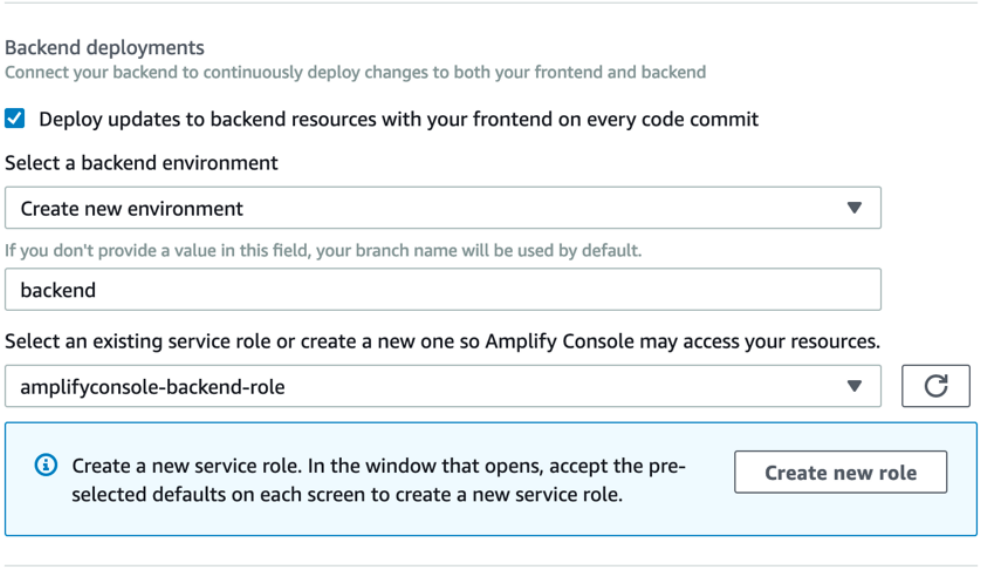
Create a new backend environment with authentication parameters for social sign-in

To connect a branch to an app

1. Sign in to the AWS Management Console and open the Amplify Console.

2. The procedure for connecting a branch to an app varies depending on whether you are connecting a branch to a new app or an existing app.
   - **Connecting a branch to a new app**

     a. When connecting a branch to a new app, in the **Configure build settings** step of the wizard, choose **Create new environment**, and enter the name of your backend environment. The following screenshot shows the **Backend deployments** section of the Amplify console with **`backend`** entered for the backend environment name.



     b. Expand the **Advanced settings** section in the build settings configuration wizard and add environment variables for social sign-in keys. For example, **`AMPLIFY_FACEBOOK_CLIENT_SECRET`** is a valid environment variable. For the list of Amplify system environment variables that are available by default, see the table in Amplify Console environment variables (p. 83).

   - **Connecting a branch to an existing app**

     a. If you are connecting a new branch to an existing app, set the social sign-in environment variables before connecting the branch. In the navigation pane, choose **App Settings**, **Environment variables**.

     b. In the **Environment variables** section, choose **Manage variables**.

     c. In the **Manage variables** section, for **Variable** (key), enter your client ID. For **Value**, enter your client secret. For the list of Amplify system environment variables that are available by default, see the table in Amplify Console environment variables (p. 83).

Frontend framework environment variables

If you are developing your app with a frontend framework that supports its own environment variables, it is important to understand that these are not the same as the environment variables you configure in the Amplify Console. For example, React (prefixed REACT_APP) and Gatsby (prefixed GATSBY), enable you to create runtime environment variables that those frameworks automatically bundle into your frontend production build. To understand the effects of using these environment variables to store values, refer to the documentation for the frontend framework you are using.

Storing sensitive values, such as API keys, inside these frontend framework prefixed environment variables is not a best practice and is highly discouraged. For an example of using the Amplify Console's build time environment variables for this purpose, see Access environment variables (p. 81).

Amplify Console environment variables

You can use the following environment variables that are accessible by default within the Amplify Console.

Variable name	Description	Example value
AWS_APP_ID	The app ID of the current build	abcd123
AWS_BRANCH	The branch name of the current build	`main`, `develop`, `beta`, `v2.0`
AWS_BRANCH_ARN	The branch ARN of the current build	`aws:arn:amplify:us-west-2:111122223333:appname/branch/...`
AWS_CLONE_URL	The clone URL used to fetch the git repository contents	git@github.com:<user-name>/<repo-name>.git
AWS_COMMIT_ID	The commit ID of the current build. “HEAD” for rebuilds	xxxxxxxxxxxxxxxxx
AWS_JOB_ID	The job ID of the current build. This includes some padding of ‘0’ so it always has the same length.	0000000001
_LIVE_UPDATES	The tool will be upgraded to the latest version.	[{“name”:”Amplify CLI”,”pkg”:”@aws-amplify/cli”,”type”:”npm”,”version”:”latest”}]
AMPLIFY_FACEBOOK_CLIENT_ID	The Facebook client ID.	123456
AMPLIFY_FACEBOOK_CLIENT_SECRET	The Facebook client secret.	example123456
AMPLIFY_GOOGLE_CLIENT_ID	The Google client ID.	123456
AMPLIFY_GOOGLE_CLIENT_SECRET	The Google client secret.	example123456
AMPLIFY_AMAZON_CLIENT_ID	The Amazon client ID.	123456
AMPLIFY_AMAZON_CLIENT_SECRET	The Amazon client secret.	example123456
AMPLIFY_DIFF_DEPLOY	Enable or disable diff based frontend deployment. For more information, see Enable or disable diff based frontend build and deploy (p. 40).	`true`
AMPLIFY_DIFF_DEPLOY_ROOT	The path to use for diff based frontend deployment comparisons, relative to the root of your repository.	`dist`

Variable name	Description	Example value
AMPLIFY_DIFF_BACKEND	Enable or disable diff based backend builds. For more information, see Enable or disable diff based backend builds (p. 40)	`true`
AMPLIFY_BACKEND_PULL_ONLY	The Amplify Console manages this environment variable. For more information, see Edit an existing frontend to point to a different backend (p. 61)	`true`
AMPLIFY_BACKEND_APP_ID	The Amplify Console manages this environment variable. For more information, see Edit an existing frontend to point to a different backend (p. 61)	`abcd123`
AMPLIFY_SKIP_BACKEND_BUILD	If you do not have a backend section in your build spec and want to disable backend builds, set this environment variable to `true`.	`true`
AMPLIFY_MONOREPO_APP_ROOT	The path to use to specify the app root of a monorepo app, relative to the root of your repository.	`apps/react-app`
_BUILD_TIMEOUT	The build timeout duration in minutes.	30

Note
The `AMPLIFY_AMAZON_CLIENT_ID` and `AMPLIFY_AMAZON_CLIENT_SECRET` environment variables are OAuth tokens, not an AWS access key and secret key.

Environment secrets

Environment secrets are similar to environment variables, but they are AWS Systems Manager (SSM) Parameter Store key value pairs that can be encrypted. Some values must be encrypted, such as the Sign in with Apple private key for Amplify Console.

Set environment secrets

Use the following instructions to set an environment secret for an Amplify app using the AWS Systems Manager console.

To set an environment secret

1. Sign in to the AWS Management Console and open the AWS Systems Manager console.
2. In the navigation pane, choose **Application Management**, then choose **Parameter Store**.
3. On the **AWS Systems Manager Parameter Store** page, choose **Create parameter**.

4. On the **Create parameter** page, in the **Parameter details** section, do the following:
   a. For **Name**, enter a parameter in the format **`/amplify/{your_app_id}/{your_backend_environment_name}/{your_parameter_name}`**.
   b. For **Type**, choose **SecureString**.
   c. For **KMS key source**, choose **My current account** to use the default key for your account.
   d. For **Value**, enter your secret value to encrypt.
5. Choose, **Create parameter**.

Note
Amplify only has access to the keys under the `/amplify/{your_app_id}/{your_backend_environment_name}` for the specific environment build. You must specify the default AWS KMS key to allow Amplify to decrypt the value.

Access environment secrets

Accessing an environment secret during a build is similar to accessing environment variables (p. 81), except that environment secrets are stored in `process.env.secrets` as a JSON string.

Amplify Console environment secrets

You can use the following environment secrets that are accessible by default within the Amplify Console.

Variable name	Description	Example value
AMPLIFY_SIWA_CLIENT_ID	The Sign in with Apple client ID	com.yourapp.auth
AMPLIFY_SIWA_TEAM_ID	The Sign in with Apple team ID	ABCD123
AMPLIFY_SIWA_KEY_ID	The Sign in with Apple key ID	ABCD123
AMPLIFY_SIWA_PRIVATE_KEY	The Sign in with Apple private key	-----BEGIN PRIVATE KEY----- ****..... -----END PRIVATE KEY-----

Custom headers

Custom HTTP headers enable you to specify headers for every HTTP response. Response headers can be used for debugging, security, and informational purposes. You can specify headers in the AWS Management Console, or by downloading and editing an app's `customHttp.yml` file and saving it in the project's root directory. For detailed procedures, see Setting custom headers (p. 87).

Previously, custom HTTP headers were specified for an app either by editing the build specification (buildspec) in the AWS Management Console or by downloading and updating the `amplify.yml` file and saving it in the project's root directory. Custom headers specified in this way should be migrated out of the buildspec and the `amplify.yml` file. For instructions, see Migrating custom headers (p. 88).

Custom header YAML format

Specify custom headers using the following YAML format:

```
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'
```

For a monorepo, use the following YAML format:

```
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'
```

When you add custom headers to your app, you will specify your own values for the following:

pattern

Custom headers are applied all URL file paths that match the pattern.

headers

Defines the headers that match the file pattern.

key

The name of the custom header.

value

The value of the custom header.

To learn more about HTTP headers, see Mozilla's list of HTTP Headers.

Setting custom headers

There are two ways to specify custom HTTP headers for an AWS Amplify app. You can specify headers in the AWS Management Console or you can specify headers by downloading and editing an app's `customHttp.yml` file and saving it in your project's root directory.

To set custom headers for an app in the AWS Management Console

1. Sign in to the AWS Management Console and open the Amplify Console.
2. Choose the app to set custom headers for.
3. In the navigation pane, choose **App settings**, **Custom headers**.
4. In the **Custom header specification** section, choose **Edit**.
5. In the **Edit** window, enter the information for your custom headers using the custom header YAML format (p. 86).
   a. For `pattern`, enter the pattern to match.
   b. For `key`, enter the name of the custom header.
   c. For `value`, enter the value of the custom header.
6. Choose **Save**.
7. If you're working with an app in a monorepo or an app that uses server-side rendering (SSR) (p. 14), you must redeploy the app to apply the new custom headers.

To set custom headers using the customHttp.yml file

1. Sign in to the AWS Management Console and open the Amplify Console.
2. Choose the app to set custom headers for.
3. In the navigation pane, choose **App settings**, **Custom headers**.
4. In the **Custom header specification** section, choose **Download**.
5. Open the downloaded `customHttp.yml` file in the code editor of your choice and enter the information for your custom headers using the custom header YAML format (p. 86).
   a. For `pattern`, enter the pattern to match.
   b. For `key`, enter the name of the custom header.
   c. For `value`, enter the value of the custom header.
6. Save the edited `customHttp.yml` file in your project's root directory. If you are working with a monorepo, save the `customHttp.yml` file in the root of your repo.
7. Deploy your app to apply the new custom headers.
   - For a CI/CD app, perform a new build from your Git repository that includes the new `customHttp.yml` file.
   - For a manual deploy app, deploy the app again in the Amplify Console and include the new `customHttp.yml` file with the artifacts that you upload.

Note
Custom headers set in the `customHttp.yml` file and deployed in the app's root directory will override custom headers defined in the **Custom headers** section in the AWS Management Console.

Migrating custom headers

Previously, custom HTTP headers were specified for an app either by editing the buildspec in the AWS Management Console or by downloading and updating the `amplify.yml` file and saving it in the project 's root directory. It is strongly recommended that you migrate your custom headers out of the buildspec and the `amplify.yml` file.

Specify your custom headers in the **Custom headers** section of the AWS Management Console or by downloading and editing the `customHttp.yml` file.

To migrate custom headers stored in the Amplify Console

1. Sign in to the AWS Management Console and open the Amplify Console.
2. Choose the app to perform the custom header migration on.
3. In the navigation pane, choose **App settings**, **Build settings**. In the **App build specification** section, you can review your app's buildspec.
4. Choose **Download** to save a copy of your current buildspec. You can reference this copy later if you need to recover any settings.
5. When the download is complete, choose **Edit**.
6. Take note of the custom header information in the file, as you will use it later in step 9. In the **Edit** window, delete any custom headers from the file and choose **Save**.
7. In the navigation pane, choose **App settings**, **Custom headers**.
8. In the **Custom header specification** section, choose **Edit**.
9. In the **Edit** window, enter the information for your custom headers that you deleted in step 6.
10. Choose **Save**.
11. Redeploy any branch that you want the new custom headers to be applied to.

To migrate custom headers from amplify.yml to customHttp.yml

1. Navigate to the `amplify.yml` file currently deployed in your app's root directory.
2. Open `amplify.yml` in the code editor of your choice.
3. Take note of the custom header information in the file, as you will use it later in step 8. Delete the custom headers in the file. Save and close the file.
4. Sign in to the AWS Management Console and open the Amplify Console.
5. Choose the app to set custom headers for.
6. In the navigation pane, choose **App settings**, **Custom headers**.
7. In the **Custom header specification** section, choose **Download**.
8. Open the downloaded `customHttp.yml` file in the code editor of your choice and enter the information for your custom headers that you deleted from `amplify.yml` in step 3.
9. Save the edited `customHttp.yml` file in your project's root directory. If you are working with a monorepo, save the file in the root of your repo.
10. Deploy your app to apply the new custom headers.
    - For a CI/CD app, perform a new build from your Git repository that includes the new `customHttp.yml` file.

- For a manual deploy app, deploy the app again in the Amplify Console and include the new `customHttp.yml` file with artifacts that you upload.

Note
Custom headers set in the `customHttp.yml` file and deployed in the app's root directory will override the custom headers defined in the **Custom headers** section of the AWS Management Console.

Monorepo custom headers

When you specify custom headers for an app in a monorepo, be aware of the following set up requirements:

- There is a specific YAML format for a monorepo. For the correct syntax, see Custom header YAML format (p. 86).
- You can specify custom headers for an application in a monorepo using the **Custom headers** section of the AWS Management Console. Note that you must redeploy your application to apply the new custom headers.
- As an alternative to using the console, you can specify custom headers for an app in a monorepo in a `customHttp.yml` file. You must save the `customHttp.yml` file in the root of your repo and then redeploy the application to apply the new custom headers. Custom headers specified in the `customHttp.yml` file will override any custom headers specified using the **Custom headers** section of the AWS Management Console.

Security headers example

Custom security headers enable enforcing HTTPS, preventing XSS attacks, and defending your browser against clickjacking. Use the following YAML syntax to apply custom security headers to your app.

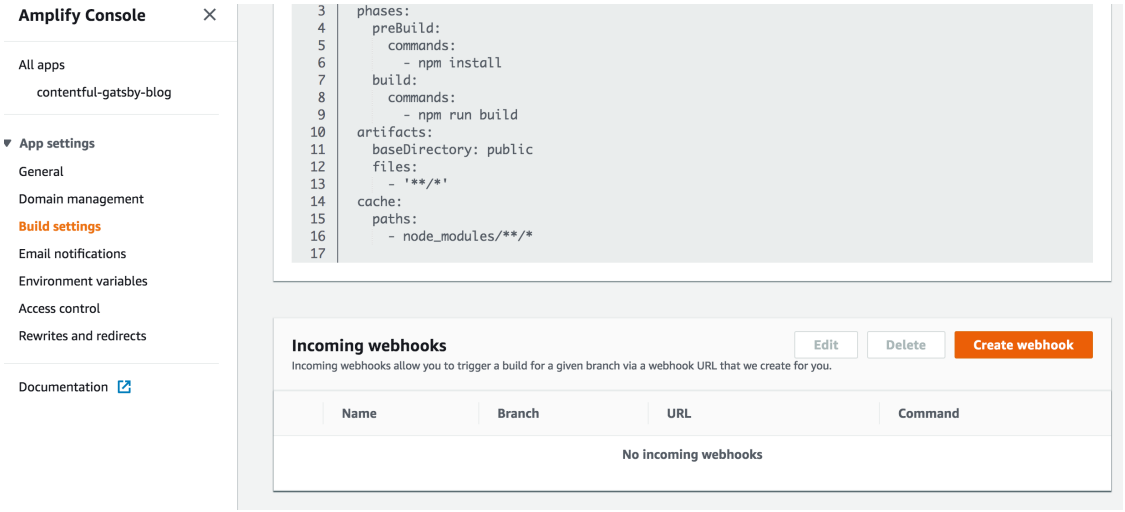
```
customHeaders:
  - pattern: '**/*'
    headers:
      - key: 'Strict-Transport-Security'
        value: 'max-age=31536000; includeSubDomains'
      - key: 'X-Frame-Options'
        value: 'SAMEORIGIN'
      - key: 'X-XSS-Protection'
        value: '1; mode=block'
      - key: 'X-Content-Type-Options'
        value: 'nosniff'
      - key: 'Content-Security-Policy'
        value: "default-src 'self'"
```

Incoming webhooks

Set up an incoming webhook in the Amplify Console to trigger a build without comitting code to your Git repository. You can use webhook triggers with headless CMS tools (such as Contentful or GraphCMS) to start a build whenever content changes, or to perform daily builds using services such as Zapier.

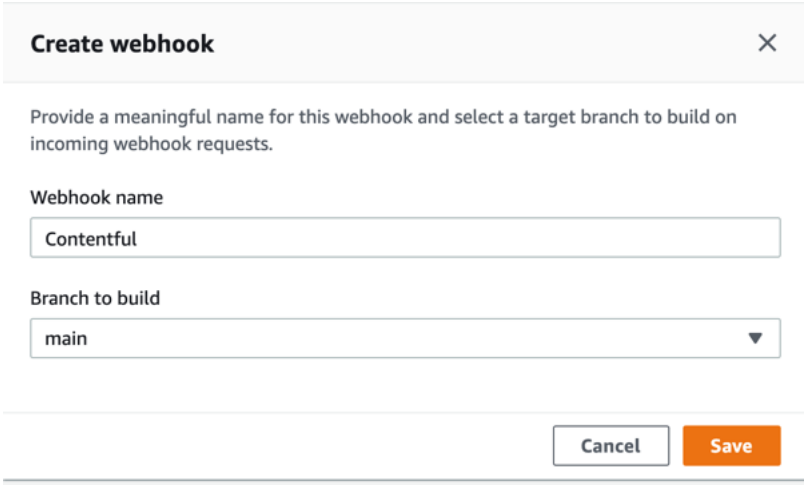
To create an incoming webhook

1. Sign in to the AWS Management Console and open the Amplify Console.
2. Choose the app that you want to create a webhook for.
3. In the navigation pane, choose **Build settings**.
4. On the **Build settings** page, scroll down to the **Incoming webhooks** section and choose **Create webhook**.

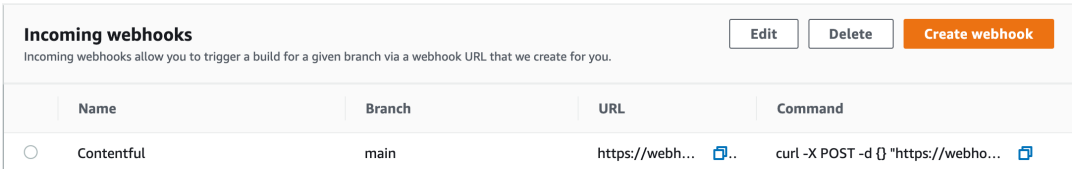


5. In the **Create webhook** dialog box, do the following:

   a. For **Webhook name** enter a name for the webhook.
   b. For **Branch to build**, select the branch to build on incoming webhook requests.
   c. Choose **Save**.



6. In the **Incoming webhooks** section, do one of the following:
   - Copy the webhook URL and provide it to a headless CMS tool or other service to trigger builds.
   - Run the curl command in a terminal window to trigger a new build.



Monitoring

AWS Amplify emits metrics through Amazon CloudWatch and provides access logs with detailed information about each request made to your app. Use the topics in this section to learn how to use these metrics and logs to monitor your app.

Topics

Monitoring with CloudWatch

AWS Amplify is integrated with Amazon CloudWatch, allowing you to monitor metrics for your Amplify applications in near real-time. You can create alarms that send notifications when a metric exceeds a threshold you set. For more information about how the CloudWatch service works, see the Amazon CloudWatch User Guide.

Metrics

Amplify supports six CloudWatch metrics in the `AWS/AmplifyHosting` namespace for monitoring traffic, errors, data transfer, and latency for your apps. These metrics are aggregated at one minute intervals. CloudWatch monitoring metrics are free of charge and don't count against the CloudWatch service quotas.

Not all available statistics are applicable for every metric. In the following table, the most relevant statistics are listed in the description for each metric.

Metrics	Description
Requests	The total number of viewer requests received by your app. The most relevant statistic is `Sum`. Use the `Sum` statistic to get the total number of requests.
BytesDownloaded	The total amount of data transferred out of your app (downloaded) in bytes by viewers for `GET`, `HEAD`, and `OPTIONS` requests. The most relevant statistic is `Sum`.
BytesUploaded	The total amount of data transferred into your app (uploaded) in bytes using `POST` and `PUT` requests. The most relevant statistic is `Sum`.
4XXErrors	The number of requests that returned an error in the HTTP status code 400-499 range. The most relevant statistic is `Sum`. Use the `Sum` statistic to get the total occurrences of these errors.

Metrics	Description
5XXErrors	The number of requests that returned an error in the HTTP status code 500-599 range. The most relevant statistic is `Sum`. Use the `Sum` statistic to get the total occurrences of these errors.
Latency	The time to first byte in seconds. This is the total time between when Amplify Console receives a request and when it returns a response to the network. This doesn't include the network latency encountered for a response to reach the viewer's device. The most relevant statistics are `Average`, `Maximum`, `Minimum`, `p10`, `p50`, `p90`, `p95`, and `p100`. Use the `Average` statistic to evaluate expected latencies.

Amplify provides the following CloudWatch metric dimensions.

Dimension	Description
App	Metric data is provided by app.
AWS Account	Metric data is provided across all apps in the AWS account.

You can access CloudWatch metrics in the AWS Management Console at https://console.aws.amazon.com/cloudwatch/. Alternatively, you can access metrics in the Amplify Console using the following procedure.

To access metrics in the Amplify console

1. Sign in to the AWS Management Console and open the Amplify Console.
2. Choose the app that you want to view metrics for.
3. In the navigation pane, choose **App Settings**, **Monitoring**.
4. On the **Monitoring** page, choose **Metrics**.

Alarms

You can create CloudWatch alarms in the Amplify console that send notifications when specific criteria are met. An alarm watches a single CloudWatch metric and sends an Amazon Simple Notification Service notification when the metric breaches the threshold for a specified number of evaluation periods.

You can create more advanced alarms that use metric math expressions in the CloudWatch console or using the CloudWatch APIs. For example, you can create an alarm that notifies you when the percentage of 4XXErrors exceeds 15% for three consecutive periods. For more information, see Creating a CloudWatch Alarm Based on a Metric Math Expression in the *Amazon CloudWatch User Guide*.

Standard CloudWatch pricing applies to alarms. For more information, see Amazon CloudWatch pricing.

Use the following procedure to create an alarm in the Amplify console.

To create a CloudWatch alarm for an Amplify metric

1. Sign in to the AWS Management Console and open the Amplify Console.
2. Choose the app that you want to set an alarm on.
3. In the navigation pane, choose **App Settings**, **Monitoring**.
4. On the **Monitoring** page, choose **Alarms**.
5. Choose **Create alarm**.
6. In the **Create alarm** window, configure your alarm as follows:

   a. For **Metric**, choose the name of the metric to monitor from the list.
   b. For **Name of alarm**, enter a meaningful name for the alarm. For example, if you are monitoring *Requests*, you could name the alarm `HighTraffic`. The name must contain only ASCII characters.
   c. For **Set up notifications**, do one of the following:
      - i. Choose **New** to set up a new Amazon SNS topic.
        ii. For **Email address**, enter the email address for the recipient of the notifications.
        iii. Choose **Add new email address** to add additional recipients.
      - i. Choose **Existing** to reuse an Amazon SNS topic.
        ii. For **SNS topic**, select the name of an existing Amazon SNS topic from the list.
   d. For **Whenever the *Statistic* of *Metric***, set the conditions for your alarm as follows:
      i. Specify whether the metric must be greater than, less than, or equal to the threshold value.
      ii. Specify the threshold value.
      iii. Specify the number of consecutive evaluation periods that must be in the alarm state to trigger the alarm.
      iv. Specify the length of time of the evaluation period.
   e. Choose **Create alarm**.

Note
Each Amazon SNS recipient that you specify receives a confirmation email from AWS Notifications. The email contains a link that the recipient must follow to confirm their subscription and receive notifications.

Access logs

Amplify stores access logs for all of the apps you host in Amplify Console. Access logs contain information about all requests that are made to your hosted apps. You can retrieve these access logs for any two week window that you specify.

Use the following procedure to retrieve access logs.

To view access logs

1. Sign in to the AWS Management Console and open the Amplify Console.
2. Choose the app that you want to view access logs for.
3. In the navigation pane, choose **App Settings**, **Monitoring**.
4. On the **Monitoring** page, choose **Access logs**.

5. Choose **Edit time range**.
6. In the **Edit time range** window, for **Start date** specify the first day of the two week interval to retrieve logs for. For **Start time**, choose the time on the first day to start the log retrieval.
7. The console displays the logs for your specified time range in the **Access logs** section. Choose **Download** to save the logs in a CSV format.

Analyzing access logs

To analyze access logs you can store the CSV files in an Amazon S3 bucket. One way to analyze your access logs is to use Amazon Athena. Athena is an interactive query service that can help you analyze data for AWS services. You can follow the step-by-step instructions here to create a table. Once your table has been created, you can query data as follows.

```
SELECT SUM(bytes) AS total_bytes
FROM logs
WHERE "date" BETWEEN DATE '2018-06-09' AND DATE '2018-06-11'
LIMIT 100;
```

Notifications

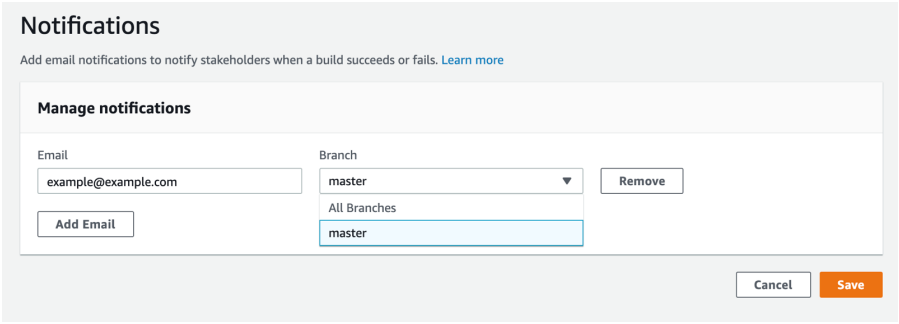
You can set up notifications for an AWS Amplify app to alert stakeholders or team members when a build succeeds or fails. Amplify Console creates an Amazon Simple Notification Service (SNS) topic in your account and uses it to configure email notifications. This Amazon SNS topic can be used to send notifications to other tools such as Slack. Notifications can be configured to apply to all branches or specific branches of an Amplify app.

Email notifications

Use the following procedures to set up email notifications for all branches or specific branches of an Amplify app.

To set up email notifications for an Amplify app

1. Sign in to the AWS Management Console and open the Amplify Console.
2. Choose the app that you want to set up email notifications for.
3. In the navigation pane, choose **App settings**, **Notifications**, and then in the **Email notifications** section, choose **Add notification**.
4. Do one of the following in the **Manage notifications** section:
   - To send notifications for a single branch, for **Email**, enter the email address to send notifications to. For **Branch**, select the name of the branch to send notifications for.
   - To send notifications for all connected branches, for **Email**, enter the email address to send notifications to. For **Branch**, select *All Branches*.
5. Choose **Save** when you are finished.



Custom build images and live package updates

Topics

Custom build images

You can use a custom build image to provide a customized build environment for an Amplify app. If you have specific dependencies that take a long time to install during a build using Amplify Console's default container, you can create your own Docker image and reference it during a build. Images can be hosted on Docker Hub or Amazon Elastic Container Registry Public.

> **Note**
> **Build settings** is visible in the Amplify Console's App settings menu only when an app is set up for continuous deployment and connected to a git repository. For instructions on this type of deployment, see Getting started with existing code (p. 3).

Configuring a custom build image

To configure a custom build image

1. Sign in to the AWS Management Console and open the Amplify Console.
2. Choose the app that you want to configure a custom build image for.
3. In the navigation pane, choose **App Settings**, **Build settings**.
4. On the **Build settings** page, in the **Build image settings** section, choose **Edit**.
5. In the **Edit build image settings** dialog box, expand the **Build image** menu, and choose **Build image**.
6. Enter the name of your build image. For example, if the name of your Docker Hub repo is *exampledockerrepo*, and your image name is *exampleimage* you would enter **`exampledockerrepo/exampleimage:latest`**.
7. Choose **Save**.

To configure a custom build image hosted in Amazon ECR

1. See Getting started in the *Amazon ECR Public User guide* to set up an Amazon ECR Public repository with a Docker image.
2. Sign in to the AWS Management Console and open the Amplify Console.
3. Choose the app that you want to configure a custom build image for.
4. In the navigation pane, choose **App Settings**, **Build settings**.

5. On the **Build settings** page, in the **Build image settings** section, choose **Edit**.
6. In the **Edit build image settings** dialog box, expand the **Build image** menu, and choose **Build image**.
7. Enter the name of the Amazon ECR Public repo that you created in step one. This is where your build image is hosted. For example, if the name of your repo is *ecr-examplerepo*, you would enter **`public.ecr.aws/xxxxxxxx/ecr-examplerepo`**.
8. Choose **Save**.

Custom build image requirements

For a custom build image to work as an Amplify Console build image, it must meet the following requirements:

1. **cURL**: When we launch your custom image, we download our build runner into your container, and therefore we require cURL to be present. If this dependency is missing, the build will instantly fail without any output as our build-runner was unable to produce any output.
2. **Git**: In order to clone your Git repository we require Git to be installed in the image. If this dependency is missing, the 'Cloning repository' step will fail.
3. **OpenSSH**: In order to securely clone your repository we require OpenSSH to set up the SSH key temporarily during the build, the OpenSSH package provides the commands that the build runner requires to do this.
4. **(NPM-based builds) Node.JS+NPM**: Our build runner does not install Node, but instead relies on Node and NPM being installed in the image. This is only required for builds that require NPM packages or Node specific commands.

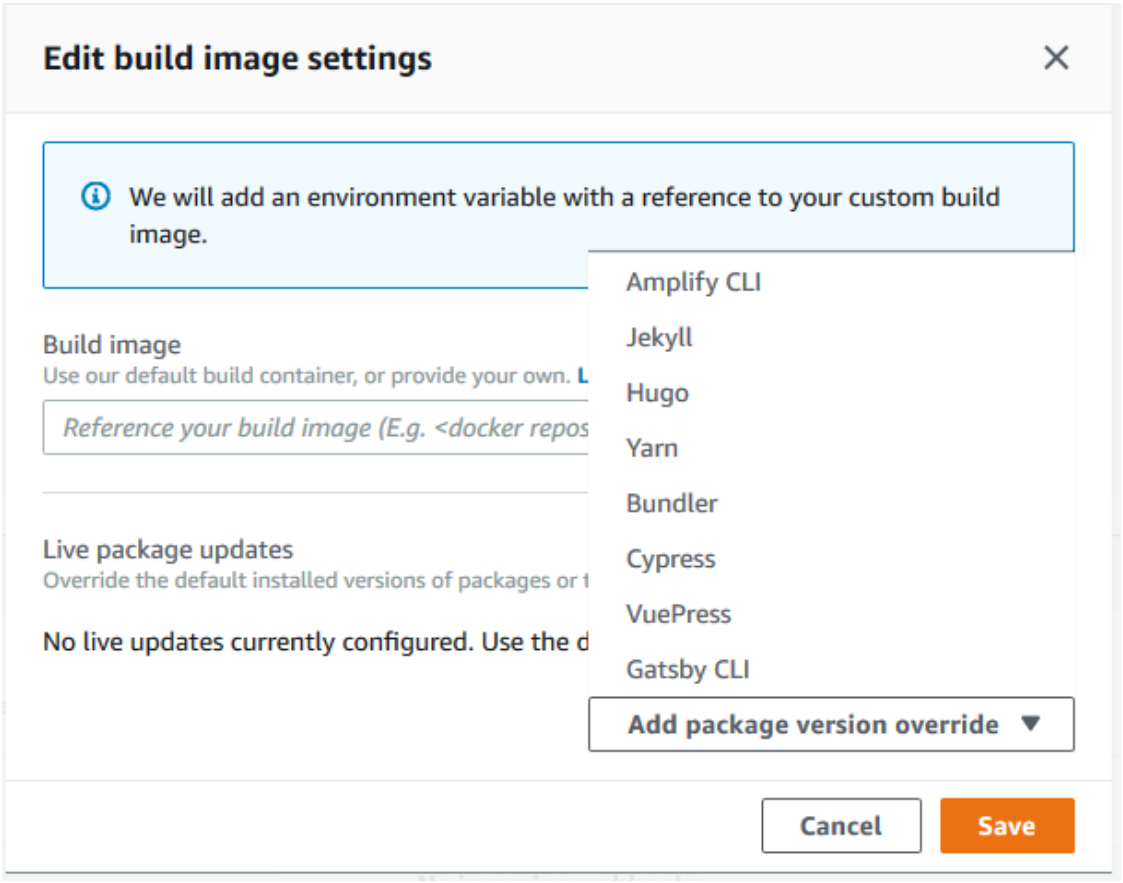
Live package updates

Live package updates enable you to specify versions of packages and dependencies to use in the Amplify Console default build image. The default build image comes with several packages and dependencies pre-installed (e.g. Hugo, Amplify CLI, Yarn, etc). With live package updates you can override the version of these dependencies and specify either a specific version, or ensure that the latest version is always installed. If live package updates is enabled, before your build runs, the build runner first updates (or downgrades) the specified dependencies. This increases the build time proportional to the time it takes to update the dependencies, but the benefit is that you can ensure the same version of a dependency is used to build your app.

Configuring live package updates

To configure live package updates

1. Sign in to the AWS Management Console and open the Amplify Console.
2. Choose the app that you want to configure live package updates for.
3. In the navigation pane, choose **App Settings**, **Build settings**.
4. On the **Build settings** page, in the **Build image settings** section, choose **Edit**.
5. In the **Edit build image settings** dialog box, expand the **Add package version override** list, and choose the package you want to change.



6. For **Version**, either keep the default **latest** or enter a specific version of the dependency. If you use **latest**, the dependency will always be upgraded to the latest version available.
7. Choose **Save**.

Adding a service role to the Amplify Console when you connect an app

The Amplify Console requires permissions to deploy backend resources with your front end. You use a service role to accomplish this. A service role is the AWS Identity and Access Management (IAM) role that Amplify Console assumes when calling other services on your behalf. In this guide, you will create an Amplify service role that has account administrative permissions and explicity allows direct access to resources that Amplify applications require to deploy any Amplify CLI resources, and create and manage backends . For more information, about the Amplify CLI, see Amplify CLI in the *Amplify Framework Documentation*.

Step 1: Sign in to the IAM console

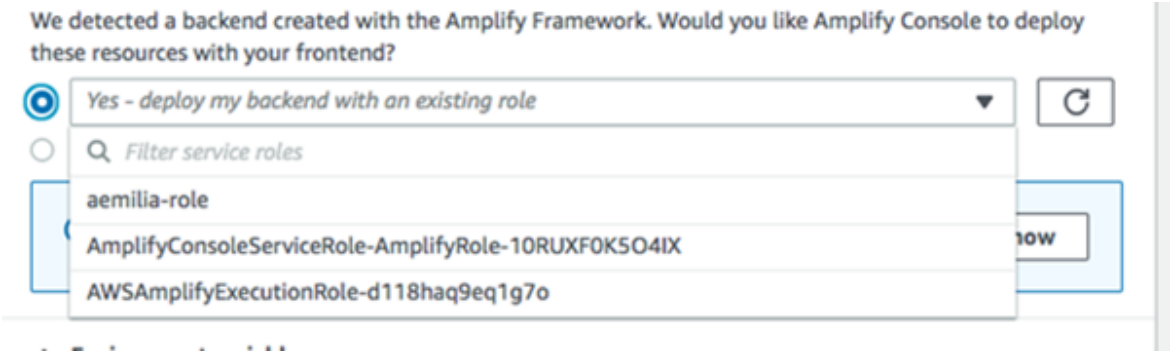
Open the IAM console and choose **Roles** from the left navigation bar, then choose **Create role**.

Step 2: Create Amplify role

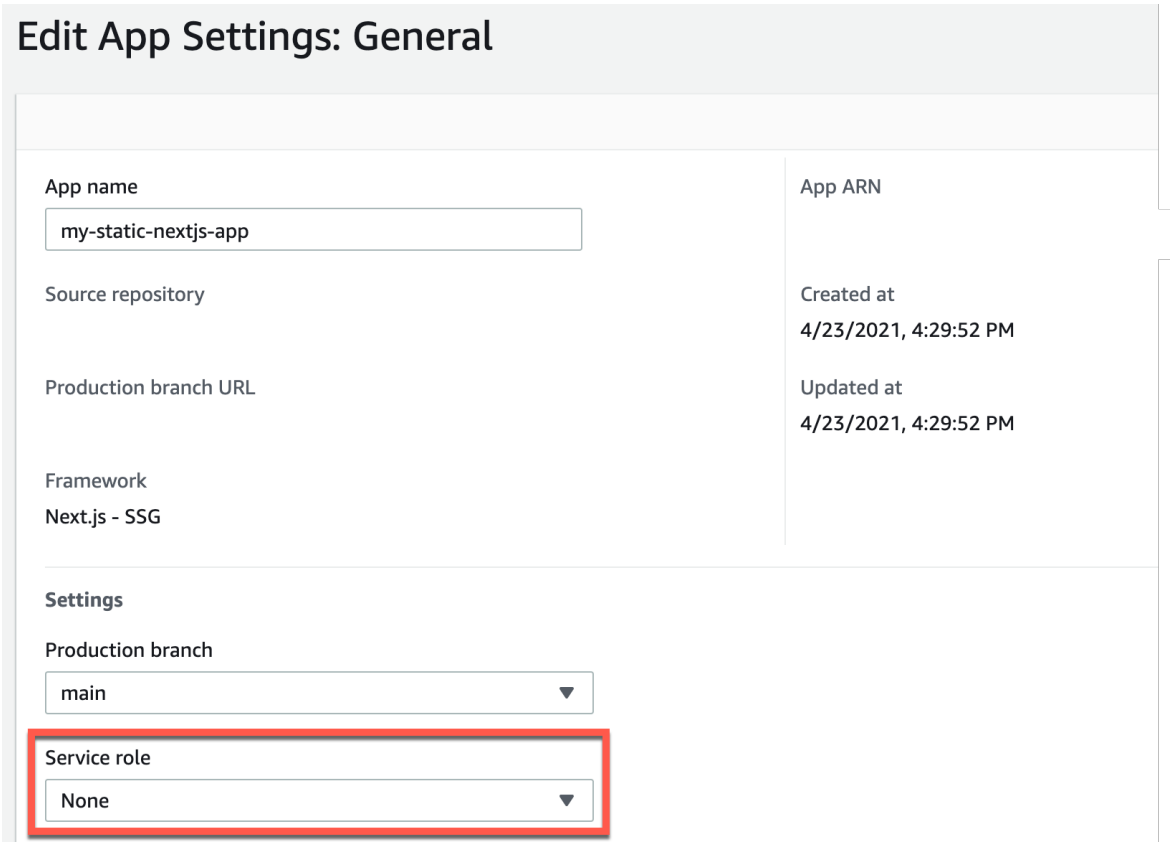
In the role selection screen find **Amplify** and choose the **Amplify-Backend Deployment** role. Accept all the defaults and choose a name for your role, such as **AmplifyConsoleServiceRole-AmplifyRole**.

Step 3: Return to the Amplify Console

Open the Amplify Console. If you are in the process of deploying a new app, choose **refresh**, and then choose the role you just created. It should look like **AmplifyConsoleServiceRole-AmplifyRole**.



If you already have an existing app, you can find the service role setting in **App settings > General** and then choose **Edit** from the top right corner of the box. Pick the service role you just created from the dropdown and choose **Save**.



The Amplify Console now has permissions to deploy backend resources.

Managing app performance

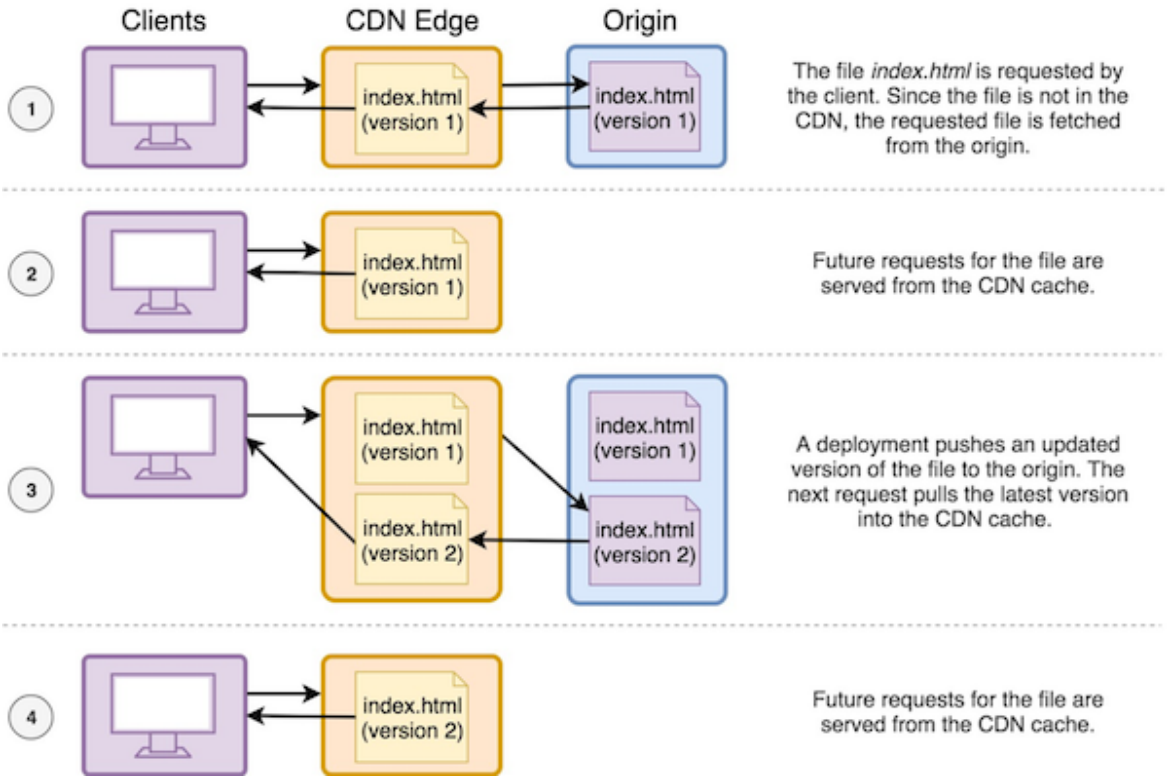
Amplify Console's default hosting architecture optimizes the balance between hosting performance and deployment availability. For more information, see the section called “Instant cache invalidation with instant deploys” (p. 102).

For advanced users that require finer control over an app's performance, Amplify Console supports *performance mode*. Performance mode optimizes for faster hosting performance by keeping content cached at the content delivery network (CDN) edge for a longer interval. For more information, see the section called “Performance mode” (p. 102).

Instant cache invalidation with instant deploys

Amplify Console supports instant cache invalidation of the CDN on every code commit. This enables you to deploy updates to your single page or static app instantly, without giving up the performance benefits of CDN caching.

For more information about how the Amplify Console handles cache invalidations, see the blog post AWS Amplify Console supports instant cache invalidation and delta deployments on every code commit.



Performance mode

Amplify Console performance mode optimizes for faster hosting performance by keeping content cached at the edge of the CDN for a longer interval. When performance mode is enabled, hosting configuration or code changes can take up to 10 minutes to be deployed and available.

Performance mode is intended for advanced customers that require finer control over an app's performance. To optimize the balance between hosting performance and deployment availability, the default the section called “Instant cache invalidation with instant deploys” (p. 102) hosting architecture is recommended.

To enable performance mode for an app

1. Sign in to the AWS Management Console and open the Amplify Console.
2. Choose the app to enable performance mode for.
3. In the navigation pane, choose **App settings**, **General**.
4. In the **General** pane, scroll down to the **Branches** section. Select the branch that you want to to enable performance mode for.
5. Choose **Action**, **Enable performance mode**.
6. In the **Enable performance mode** dialog box, choose **Enable performance mode**.

Using headers to control cache duration

HTTP `Cache-Control` header `max-age` and `s-maxage` directives affect the content caching duration for your app. The `max-age` directive tells the browser how long (in seconds) that you want content to remain in the cache before it is refreshed from the origin server. The `s-maxage` directive overrides `max-age` and lets you specify how long (in seconds) that you want content to remain at the CDN edge before it is refreshed from the origin server. Note that apps hosted with Amplify Console honor and reuse the `Cache-Control` request headers sent by clients, unless they are overridden by a custom header that you define. Continue reading for a description of how to configure a custom header.

You can manually adjust the `s-maxage` directive to have more control over the performance and deployment availability of your app. For example, to increase the length of time that your content stays cached at the edge, you can manually increase the time to live (TTL) by updating `s-maxage` to a value longer than the default 600 seconds (10 minutes).

> **Note**
> When performance mode is enabled for an app, Amplify increases the maximum TTL, that you can set for the app using a custom header, from 10 minutes (600 seconds) to one day (86,400 seconds). Amplify caps the `s-maxage` that you can set using a custom header at one day. For example, if you set `s-maxage` to one week (604,800 seconds), Amplify uses the maximum TTL of one day.

You can define custom headers for an app in the **Custom headers** section of the Amplify Console. For more information, see Setting custom headers (p. 87). To specify a custom value for `s-maxage`, use the following YAML format. This example keeps the associated content cached at the edge for 3600 seconds (one hour).

```
customHeaders:
  - pattern: '/img/*'
    headers:
      - key: 'Cache-Control'
        value: 's-maxage=3600'
```

Logging Amplify API calls using AWS CloudTrail

AWS Amplify is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Amplify. CloudTrail captures all API calls for Amplify as events. The calls captured include calls from the Amplify console and code calls to the Amplify API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Amplify. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information that CloudTrail collects, you can determine the request that was made to Amplify, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

Amplify information in CloudTrail

CloudTrail is enabled on your AWS account by default. When activity occurs in Amplify, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing events with CloudTrail Event history in the *AWS CloudTrail User Guide*.

For an ongoing record of events in your AWS account, including events for Amplify, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following in the *AWS CloudTrail User Guide*:

- Creating a trail for your AWS account
- CloudTrail supported services and integrations
- Configuring Amazon SNS notifications for CloudTrail
- Receiving CloudTrail log files from multiple regions and Receiving CloudTrail log files from multiple accounts

All Amplify operations are logged by CloudTrail and are documented in the AWS Amplify Console API Reference, and the AWS Amplify Admin UI API Reference. For example, calls to the `CreateApp`, `DeleteApp` and `DeleteBackendEnvironment` operations generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Was the request made with root or AWS Identity and Access Management (IAM) user credentials.
- Was the request made with temporary security credentials for a role or federated user.
- Was the request made by another AWS service.

For more information, see the CloudTrail userIdentity element in the *AWS CloudTrail User Guide*.

Understanding Amplify log file entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the Amplify Console API Reference `DeleteBackendEnvironment` operation.


```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::444455556666:user/Mary_Major",
        "accountId": "444455556666",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AIDACKCEVSQ6C2EXAMPLE",
                "arn": "arn:aws:iam::444455556666:user/Mary_Major",
                "accountId": "444455556666",
                "userName": "Mary_Major"
            },
            "webIdFederationData": {},
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2021-01-12T00:28:50Z"
            }
        },
        "invokedBy": "apigateway.amazonaws.com"
    },
    "eventTime": "2021-01-12T00:31:08Z",
    "eventSource": "amplify.amazonaws.com",
    "eventName": "DeleteBackendEnvironment",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "apigateway.amazonaws.com",
    "userAgent": "apigateway.amazonaws.com",
    "requestParameters": {
        "environmentName": "staging",
        "appId": "d3lap6vexample"
    },
    "responseElements": {
        "backendEnvironment": {
            "backendEnvironmentArn": "arn:aws:amplify:us-west-2:444455556666:apps/
d3lap6vexample/backendenvironments/staging",
            "createTime": 1610086829.109,
            "deploymentArtifacts": "amplify-amplify9b7cd3example-staging-62027-deployment",
            "environmentName": "staging",
            "stackName": "amplify-amplify9b7cd3example-staging-62027",
            "updateTime": 1610086829.109
        }
    },
    "requestID": "1135382e-f832-45ba-ae53-f7ffbexample",
    "eventID": "cebab152-deb6-42e1-bd1f-d05b6example",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "444455556666"
```

```
}
```

The following example shows a CloudTrail log entry that demonstrates the AWS Amplify Console API Reference `ListApps` operation.

```
{
    "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"
}
```

The following example shows a CloudTrail log entry that demonstrates the AWS Amplify Admin UI API Reference `ListBackendJobs` operation.

```
{
    "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"
}
```

Security in Amplify

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from data centers and network architectures that are built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS Compliance Programs. To learn about the compliance programs that apply to AWS Amplify, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using Amplify. The following topics show you how to configure Amplify to meet your security and compliance objectives. You also learn how to use other AWS services that help you monitor and secure your Amplify resources.

Topics

Identity and Access Management for Amplify

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use Amplify resources. IAM is an AWS service that you can use with no additional charge.

Topics

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in Amplify.

Service user – If you use the Amplify service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more Amplify features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in Amplify, see Troubleshooting Amplify identity and access (p. 125).

Service administrator – If you're in charge of Amplify resources at your company, you probably have full access to Amplify. It's your job to determine which Amplify features and resources your employees should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with Amplify, see How Amplify works with IAM (p. 112).

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to Amplify. To view example Amplify identity-based policies that you can use in IAM, see Identity-based policy examples for Amplify (p. 117).

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. For more information about signing in using the AWS Management Console, see Signing in to the AWS Management Console as an IAM user or root user in the *IAM User Guide*.

You must be *authenticated* (signed in to AWS) as the AWS account root user, an IAM user, or by assuming an IAM role. You can also use your company's single sign-on authentication or even sign in using Google or Facebook. In these cases, your administrator previously set up identity federation using IAM roles. When you access AWS using credentials from another company, you are assuming a role indirectly.

To sign in directly to the AWS Management Console, use your password with your root user email address or your IAM user name. You can access AWS programmatically using your root user or IAM users access keys. AWS provides SDK and command line tools to cryptographically sign your request using your credentials. If you don't use AWS tools, you must sign the request yourself. Do this using *Signature Version 4*, a protocol for authenticating inbound API requests. For more information about authenticating requests, see Signature Version 4 signing process in the *AWS General Reference*.

Regardless of the authentication method that you use, you might also be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.

AWS account root user

When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the best practice of using the root user only to create your first IAM user. Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.

IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. An IAM user can have long-term credentials such as a user name and password or a set of

access keys. To learn how to generate access keys, see Managing access keys for IAM users in the *IAM User Guide*. When you generate access keys for an IAM user, make sure you view and securely save the key pair. You cannot recover the secret access key in the future. Instead, you must generate a new access key pair.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see When to create an IAM user (instead of a role) in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by switching roles. You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Using IAM roles in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Temporary IAM user permissions** – An IAM user can assume an IAM role to temporarily take on different permissions for a specific task.
- **Federated user access** – Instead of creating an IAM user, you can use existing identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as *federated users*. AWS assigns a role to a federated user when access is requested through an identity provider. For more information about federated users, see Federated users and roles in the *IAM User Guide*.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
  - **Principal permissions** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see Actions, resources, and condition keys for AWS Amplify in the *Service Authorization Reference*.
  - **Service role** – A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.
  - **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests.

This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Using an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

To learn whether to use IAM roles or IAM users, see When to create an IAM role (instead of a user) in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to IAM identities or AWS resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. You can sign in as the root user or an IAM user, or you can assume an IAM role. When you then make a request, AWS evaluates the related identity-based or resource-based policies. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

Every IAM entity (user or role) starts with no permissions. In other words, by default, users can do nothing, not even change their own password. To give a user permission to do something, an administrator must attach a permissions policy to a user. Or the administrator can add the user to a group that has the intended permissions. When an administrator gives permissions to a group, all users in that group are granted those permissions.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choosing between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see How SCPs work in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

How Amplify works with IAM

Before you use IAM to manage access to Amplify, learn what IAM features are available to use with Amplify.

IAM features that you can use with Amplify

IAM feature	Amplify support
Identity-based policies (p. 113)	Yes
Resource-based policies (p. 113)	No
Policy actions (p. 114)	Yes

IAM feature	Amplify support
Policy resources (p. 114)	Yes
Policy condition keys (p. 115)	Yes
ACLs (p. 115)	No
ABAC (tags in policies) (p. 115)	Partial
Temporary credentials (p. 116)	Yes
Principal permissions (p. 116)	Yes
Service roles (p. 116)	Yes
Service-linked roles (p. 117)	No

To get a high-level view of how Amplify and other AWS services work with most IAM features, see AWS services that work with IAM in the *IAM User Guide*.

Identity-based policies for Amplify

Supports identity-based policies	Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. You can't specify the principal in an identity-based policy because it applies to the user or role to which it is attached. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Identity-based policy examples for Amplify

To view examples of Amplify identity-based policies, see Identity-based policy examples for Amplify (p. 117).

Resource-based policies within Amplify

Supports resource-based policies	No

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. Adding a cross-account principal to a resource-based policy is

only half of establishing the trust relationship. When the principal and the resource are in different AWS accounts, an IAM administrator in the trusted account must also grant the principal entity (user or role) permission to access the resource. They grant permission by attaching an identity-based policy to the entity. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. For more information, see How IAM roles differ from resource-based policies in the *IAM User Guide*.

Policy actions for Amplify

Supports policy actions	Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

For a list of Amplify actions, see Actions defined by AWS Amplify in the *Service Authorization Reference*.

Policy actions in Amplify use the following prefix before the action:

```
amplify
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
      "amplify:action1",
      "amplify:action2"
         ]
```

To view examples of Amplify identity-based policies, see Identity-based policy examples for Amplify (p. 117).

Policy resources for Amplify

Supports policy resources	Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

For a list of Amplify resource types and their ARNs, see Resource types defined by AWS Amplify in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see Actions defined by AWS Amplify.

To view examples of Amplify identity-based policies, see Identity-based policy examples for Amplify (p. 117).

Policy condition keys for Amplify

Supports policy condition keys	Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element (or `Condition` *block*) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical `AND` operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical `OR` operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM policy elements: variables and tags in the *IAM User Guide*.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

For a list of Amplify condition keys, see Condition keys for AWS Amplify in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see Actions defined by AWS Amplify.

To view examples of Amplify identity-based policies, see Identity-based policy examples for Amplify (p. 117).

Access control lists (ACLs) in Amplify

Supports ACLs	No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Attribute-based access control (ABAC) with Amplify

Supports ABAC (tags in policies)	Partial

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called *tags*. You can attach tags to IAM entities (users or roles) and to many AWS resources. Tagging entities and resources is the first step of ABAC. Then you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

For more information about ABAC, see What is ABAC? in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

Using temporary credentials with Amplify

Supports temporary credentials	Yes

Some AWS services don't work when you sign in using temporary credentials. For additional information, including which AWS services work with temporary credentials, see AWS services that work with IAM in the *IAM User Guide*.

You are using temporary credentials if you sign in to the AWS Management Console using any method except a user name and password. For example, when you access AWS using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see Switching to a role (console) in the *IAM User Guide*.

You can manually create temporary credentials using the AWS CLI or AWS API. You can then use those temporary credentials to access AWS. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM.

Cross-service principal permissions for Amplify

Supports principal permissions	Yes

When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see Actions, resources, and condition keys for AWS Amplify in the *Service Authorization Reference*.

Service roles for Amplify

Supports service roles	Yes

A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.

Warning
Changing the permissions for a service role might break Amplify functionality. Edit service roles only when Amplify provides guidance to do so.

Service-linked roles for Amplify

Supports service-linked roles	No

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For details about creating or managing service-linked roles, see AWS services that work with IAM in the *IAM User Guide*. Find a service in the table that includes a `Yes` in the **Service-linked role** column. Choose the **Yes** link to view the service-linked roles documentation for that service.

Identity-based policy examples for Amplify

By default, IAM users and roles don't have permission to create or modify Amplify resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS API. An IAM administrator must create IAM policies that grant users and roles permission to perform actions on the resources that they need. The administrator must then attach those policies to the IAM users or groups that require those permissions.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see Creating policies on the JSON tab in the *IAM User Guide*.

Topics

Policy best practices

Identity-based policies are very powerful. They determine whether someone can create, access, or delete Amplify resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started using AWS managed policies** – To start using Amplify quickly, use AWS managed policies to give your employees the permissions they need. These policies are already available in your account and are maintained and updated by AWS. For more information, see Get started using permissions with AWS managed policies in the *IAM User Guide*.
- **Grant least privilege** – When you create custom policies, grant only the permissions required to perform a task. Start with a minimum set of permissions and grant additional permissions as necessary. Doing so is more secure than starting with permissions that are too lenient and then trying to tighten them later. For more information, see Grant least privilege in the *IAM User Guide*.
- **Enable MFA for sensitive operations** – For extra security, require IAM users to use multi-factor authentication (MFA) to access sensitive resources or API operations. For more information, see Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.
- **Use policy conditions for extra security** – To the extent that it's practical, define the conditions under which your identity-based policies allow access to a resource. For example, you can write conditions to specify a range of allowable IP addresses that a request must come from. You can also write conditions

to allow requests only within a specified date or time range, or to require the use of SSL or MFA. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.

Using the Amplify console

To access the AWS Amplify console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the Amplify resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (IAM users or roles) with that policy.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that you're trying to perform.

With the release of the Admin UI, deleting an app or a backend requires both `amplify` and `amplifybackend` permissions. If an IAM policy provides only `amplify` permissions, a user gets a permissions error when trying to delete an app. If you are an administrator writing policies, use the permissions reference (p.) to determine the correct permissions to give users who need to perform delete actions.

To ensure that users and roles can still use the Amplify console, also attach the Amplify `ConsoleAccess` or `ReadOnly` AWS managed policy to the entities. For more information, see Adding permissions to a user in the *IAM User Guide*.

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "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 managed policies for AWS Amplify

To add permissions to users, groups, and roles, it is easier to use AWS managed policies than to write policies yourself. It takes time and expertise to create IAM customer managed policies that provide your team with only the permissions they need. To get started quickly, you can use our AWS managed policies. These policies cover common use cases and are available in your AWS account. For more information about AWS managed policies, see AWS managed policies in the *IAM User Guide*.

AWS services maintain and update AWS managed policies. You can't change the permissions in AWS managed policies. Services occasionally add additional permissions to an AWS managed policy to support new features. This type of update affects all identities (users, groups, and roles) where the policy is attached. Services are most likely to update an AWS managed policy when a new feature is launched or when new operations become available. Services do not remove permissions from an AWS managed policy, so policy updates won't break your existing permissions.

Additionally, AWS supports managed policies for job functions that span multiple services. For example, the **ReadOnlyAccess** AWS managed policy provides read-only access to all AWS services and resources. When a service launches a new feature, AWS adds read-only permissions for new operations and resources. For a list and descriptions of job function policies, see AWS managed policies for job functions in the *IAM User Guide*.

AWS managed policy: AdministratorAccess-Amplify

You can attach `AdministratorAccess-Amplify` to your IAM entities. Amplify also attaches this policy to a service role that allows Amplify to perform actions on your behalf. When you deploy a backend in the Amplify console, you must create an `Amplify-Backend Deployment` service role that Amplify uses to create and manage AWS resources. IAM attaches the `AdministratorAccess-Amplify` managed policy to the `Amplify-Backend Deployment` service role.

This policy grants account administrative permissions while explicity allowing direct access to resources that Amplify applications require to create and manage backends.

Permissions details

This policy provides access to multiple AWS services, including IAM actions. These actions allow identities with this policy to use AWS Identity and Access Management to create other identities with any permissions. This allows permissions escalation and this policy should be considered as powerful as the `AdministratorAccess` policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Sid": "CLICloudformationPolicy",
            "Effect": "Allow",
            "Action": [
                "cloudformation:CreateChangeSet",
                "cloudformation:CreateStack",
                "cloudformation:DeleteStack",
                "cloudformation:DescribeChangeSet",
                "cloudformation:DescribeStackEvents",
                "cloudformation:DescribeStackResource",
                "cloudformation:DescribeStackResources",
                "cloudformation:DescribeStacks",
                "cloudformation:ExecuteChangeSet",
                "cloudformation:GetTemplate",
                "cloudformation:UpdateStack",
                "cloudformation:ListStackResources"
            ],
            "Resource": [
                "arn:aws:cloudformation:*:*:stack/amplify-*"
            ]
        },
        {
            "Sid": "CLIManageviaCFNPolicy",
            "Effect": "Allow",
            "Action": [
                "iam:ListRoleTags",
                "iam:TagRole",
                "iam:AttachRolePolicy",
                "iam:CreatePolicy",
                "iam:DeletePolicy",
                "iam:DeleteRole",
                "iam:DeleteRolePolicy",
                "iam:DetachRolePolicy",
                "iam:PutRolePolicy",
                "iam:UpdateRole",
                "iam:GetRole",
                "iam:GetPolicy",
                "iam:GetRolePolicy",
                "iam:PassRole",
                "iam:ListPolicyVersions",
                "iam:CreatePolicyVersion",
                "iam:DeletePolicyVersion",
                "iam:CreateRole",
                "iam:ListRolePolicies",
                "iam:PutRolePermissionsBoundary",
                "iam:DeleteRolePermissionsBoundary",
                "appsync:CreateApiKey",
                "appsync:CreateDataSource",
                "appsync:CreateFunction",
                "appsync:CreateResolver",
                "appsync:CreateType",
                "appsync:DeleteApiKey",
                "appsync:DeleteDataSource",
                "appsync:DeleteFunction",
                "appsync:DeleteResolver",
                "appsync:DeleteType",
                "appsync:GetDataSource",
                "appsync:GetFunction",
                "appsync:GetIntrospectionSchema",
                "appsync:GetResolver",
                "appsync:GetSchemaCreationStatus",
                "appsync:GetType",
                "appsync:GraphQL",
                "appsync:ListApiKeys",
                "appsync:ListDataSources",
                "appsync:ListFunctions",
                "appsync:ListGraphqlApis",
```

```
                "appsync:ListResolvers",
                "appsync:ListResolversByFunction",
                "appsync:ListTypes",
                "appsync:StartSchemaCreation",
                "appsync:UpdateApiKey",
                "appsync:UpdateDataSource",
                "appsync:UpdateFunction",
                "appsync:UpdateResolver",
                "appsync:UpdateType",
                "appsync:TagResource",
                "appsync:CreateGraphqlApi",
                "appsync:DeleteGraphqlApi",
                "appsync:GetGraphqlApi",
                "appsync:ListTagsForResource",
                "appsync:UpdateGraphqlApi",
                "apigateway:DELETE",
                "apigateway:GET",
                "apigateway:PATCH",
                "apigateway:POST",
                "apigateway:PUT",
                "cognito-idp:CreateUserPool",
                "cognito-identity:CreateIdentityPool",
                "cognito-identity:DeleteIdentityPool",
                "cognito-identity:DescribeIdentity",
                "cognito-identity:DescribeIdentityPool",
                "cognito-identity:SetIdentityPoolRoles",
                "cognito-identity:GetIdentityPoolRoles",
                "cognito-identity:UpdateIdentityPool",
                "cognito-idp:CreateUserPoolClient",
                "cognito-idp:DeleteGroup",
                "cognito-idp:DeleteUserPool",
                "cognito-idp:DeleteUserPoolClient",
                "cognito-idp:DescribeUserPool",
                "cognito-idp:DescribeUserPoolClient",
                "cognito-idp:ListTagsForResource",
                "cognito-idp:ListUserPoolClients",
                "cognito-idp:UpdateUserPoolClient",
                "cognito-idp:CreateGroup",
                "cognito-idp:DeleteGroup",
                "cognito-identity:TagResource",
                "cognito-idp:TagResource",
                "cognito-idp:UpdateUserPool",
                "lambda:AddPermission",
                "lambda:CreateFunction",
                "lambda:DeleteFunction",
                "lambda:GetFunction",
                "lambda:GetFunctionConfiguration",
                "lambda:InvokeAsync",
                "lambda:InvokeFunction",
                "lambda:RemovePermission",
                "lambda:UpdateFunctionCode",
                "lambda:UpdateFunctionConfiguration",
                "lambda:ListTags",
                "lambda:TagResource",
                "lambda:UntagResource",
                "lambda:DeleteFunction",
                "lambda:AddLayerVersionPermission",
                "lambda:CreateEventSourceMapping",
                "lambda:DeleteEventSourceMapping",
                "lambda:DeleteLayerVersion",
                "lambda:GetEventSourceMapping",
                "lambda:GetLayerVersion",
                "lambda:ListEventSourceMappings",
                "lambda:ListLayerVersions",
                "lambda:PublishLayerVersion",
                "lambda:RemoveLayerVersionPermission",
```

```
                "dynamodb:CreateTable",
                "dynamodb:DeleteItem",
                "dynamodb:DeleteTable",
                "dynamodb:DescribeContinuousBackups",
                "dynamodb:DescribeTable",
                "dynamodb:DescribeTimeToLive",
                "dynamodb:ListStreams",
                "dynamodb:PutItem",
                "dynamodb:TagResource",
                "dynamodb:ListTagsOfResource",
                "dynamodb:UpdateContinuousBackups",
                "dynamodb:UpdateItem",
                "dynamodb:UpdateTable",
                "dynamodb:UpdateTimeToLive",
                "s3:CreateBucket",
                "s3:ListBucket",
                "s3:PutBucketAcl",
                "s3:PutBucketCORS",
                "s3:PutBucketNotification",
                "s3:PutBucketPolicy",
                "s3:PutBucketWebsite",
                "s3:PutObjectAcl",
                "cloudfront:CreateCloudFrontOriginAccessIdentity",
                "cloudfront:CreateDistribution",
                "cloudfront:DeleteCloudFrontOriginAccessIdentity",
                "cloudfront:DeleteDistribution",
                "cloudfront:GetCloudFrontOriginAccessIdentity",
                "cloudfront:GetCloudFrontOriginAccessIdentityConfig",
                "cloudfront:GetDistribution",
                "cloudfront:GetDistributionConfig",
                "cloudfront:TagResource",
                "cloudfront:UntagResource",
                "cloudfront:UpdateCloudFrontOriginAccessIdentity",
                "cloudfront:UpdateDistribution",
                "events:DeleteRule",
                "events:DescribeRule",
                "events:ListRuleNamesByTarget",
                "events:PutRule",
                "events:PutTargets",
                "events:RemoveTargets",
                "mobiletargeting:GetApp",
                "kinesis:AddTagsToStream",
                "kinesis:CreateStream",
                "kinesis:DeleteStream",
                "kinesis:DescribeStream",
                "kinesis:PutRecords",
                "es:AddTags",
                "es:CreateElasticsearchDomain",
                "es:DeleteElasticsearchDomain",
                "es:DescribeElasticsearchDomain",
                "s3:PutEncryptionConfiguration"
            ],
            "Resource": "*",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "aws:CalledVia": [
                        "cloudformation.amazonaws.com"
                    ]
                }
            }
        },
        {
            "Sid": "CLISDKCalls",
            "Effect": "Allow",
            "Action": [
                "appsync:GetIntrospectionSchema",
```

```
                "appsync:GraphQL",
                "appsync:UpdateApiKey",
                "appsync:ListApiKeys",
                "s3:PutObject",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:ListBucketVersions",
                "s3:DeleteBucket",
                "s3:DeleteBucketPolicy",
                "s3:DeleteBucketWebsite",
                "s3:DeleteObject",
                "s3:DeleteObjectVersion",
                "s3:GetBucketLocation",
                "s3:ListAllMyBuckets",
                "amplify:*",
                "amplifybackend:*",
                "sts:AssumeRole",
                "mobiletargeting:*",
                "cognito-idp:AdminAddUserToGroup",
                "cognito-idp:AdminCreateUser",
                "cognito-idp:CreateGroup",
                "cognito-idp:DeleteGroup",
                "cognito-idp:DeleteUser",
                "cognito-idp:ListUsers",
                "cognito-idp:AdminGetUser",
                "cognito-idp:ListUsersInGroup",
                "cognito-idp:AdminDisableUser",
                "cognito-idp:AdminRemoveUserFromGroup",
                "cognito-idp:AdminResetUserPassword",
                "cognito-idp:AdminListGroupsForUser",
                "cognito-idp:ListGroups",
                "cognito-idp:AdminDeleteUser",
                "cognito-idp:AdminListUserAuthEvents",
                "cognito-idp:AdminDeleteUser",
                "cognito-idp:AdminConfirmSignUp",
                "cognito-idp:AdminEnableUser",
                "cognito-idp:AdminUpdateUserAttributes",
                "cognito-idp:DescribeIdentityProvider",
                "cognito-idp:DescribeUserPool",
                "cognito-idp:DeleteUserPool",
                "cognito-idp:DescribeUserPoolClient",
                "cognito-idp:CreateUserPool",
                "cognito-idp:CreateUserPoolClient",
                "cognito-idp:UpdateUserPool",
                "cognito-idp:AdminSetUserPassword",
                "cognito-identity:GetIdentityPoolRoles",
                "cognito-identity:SetIdentityPoolRoles",
                "cognito-identity:CreateIdentityPool",
                "cognito-identity:DeleteIdentityPool",
                "cognito-idp:ListUserPools",
                "cognito-idp:ListUserPoolClients",
                "cognito-identity:ListIdentityPools",
                "cognito-identity:DescribeIdentityPool",
                "dynamodb:DescribeTable",
                "lambda:GetFunction",
                "lambda:CreateFunction",
                "lambda:AddPermission",
                "lambda:DeleteFunction",
                "s3:DeleteObjectVersion",
                "s3:PutEncryptionConfiguration",
                "iam:PutRolePolicy",
                "iam:CreatePolicy",
                "iam:AttachRolePolicy",
                "iam:ListPolicyVersions",
                "iam:ListAttachedRolePolicies",
                "iam:CreateRole",
```

```
                "iam:PassRole",
                "iam:ListRolePolicies",
                "iam:DeleteRolePolicy",
                "iam:CreatePolicyVersion",
                "iam:DeletePolicyVersion",
                "iam:DeleteRole",
                "cloudformation:ListStacks",
                "sns:CreateSMSSandboxPhoneNumber",
                "sns:GetSMSSandboxAccountStatus",
                "sns:VerifySMSSandboxPhoneNumber",
                "sns:DeleteSMSSandboxPhoneNumber",
                "sns:ListSMSSandboxPhoneNumbers",
                "sns:ListOriginationNumbers",
            ],
            "Resource": "*"
        },
        {
            "Sid": "AmplifySSMCalls",
            "Effect": "Allow",
            "Action": [
                "ssm:PutParameter",
                "ssm:DeleteParameter",
                "ssm:GetParametersByPath",
                "ssm:GetParameters",
                "ssm:GetParameter",
                "ssm:DeleteParameters"
            ],
            "Resource": "arn:aws:ssm:*:*:parameter/amplify/*"
        }
    ]
}
```

Amplify updates to AWS managed policies

View details about updates to AWS managed policies for Amplify since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the Document history for AWS Amplify (p. 137) page.

Change	Description	Date
AdministratorAccess-Amplify (p. 119) – Update to an existing policy	Consolidate all Amplify actions into a single `amplify:*` action. Add an Amazon S3 action to support encrypting customer Amazon S3 buckets. Add IAM permission boundary actions to support Amplify apps that have permission boundaries enabled. Add Amazon SNS actions to support viewing origination phone numbers, and viewing, creating, verifying, and deleting destination phone numbers.	July 28, 2021

Change	Description	Date
	Amplify Admin UI: Add Amazon Cognito, AWS Lambda, IAM, and AWS CloudFormation policy actions to enable managing backends in the Amplify console and Amplify Admin UI . Add an AWS Systems Manager (SSM) policy statement to manage Amplify environment secrets. Add an AWS CloudFormation `ListResources` action to support Lambda layers for Amplify apps.	
Amplify started tracking changes	Amplify started tracking changes for its AWS managed policies.	July 28, 2021

Troubleshooting Amplify identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with Amplify and IAM.

Topics

I am not authorized to perform an action in Amplify

If the AWS Management Console tells you that you're not authorized to perform an action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password.

With the release of the Admin UI, deleting an app or a backend requires both `amplify` and `amplifybackend` permissions. If an administrator has written an IAM policy that provides only `amplify` permissions, a user will get a permissions error when trying to delete an app.

The following example error occurs when the `mateojackson` IAM user tries to use the console to delete a fictional *`example-amplify-app`* resource but does not have the `amplifybackend:`*`RemoveAllBackends`* permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 amplifybackend;:RemoveAllBackends on resource: example-amplify-app
```

In this case, Mateo asks his administrator to update his policies to allow him to access the *`example-amplify-app`* resource using the `amplifybackend:`*`RemoveAllBackends`* action.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password. Ask that person to update your policies to allow you to pass a role to Amplify.

Some AWS services allow you to pass an existing role to that service, instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in Amplify. However, the action requires the service to have permissions granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform: iam:PassRole
```

In this case, Mary asks her administrator to update her policies to allow her to perform the `iam:PassRole` action.

I want to view my access keys

After you create your IAM user access keys, you can view your access key ID at any time. However, you can't view your secret access key again. If you lose your secret key, you must create a new access key pair.

Access keys consist of two parts: an access key ID (for example, `AKIAIOSFODNN7EXAMPLE`) and a secret access key (for example, `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). Like a user name and password, you must use both the access key ID and secret access key together to authenticate your requests. Manage your access keys as securely as you do your user name and password.

> **Important**
> Do not provide your access keys to a third party, even to help find your canonical user ID. By doing this, you might give someone permanent access to your account.

When you create an access key pair, you are prompted to save the access key ID and secret access key in a secure location. The secret access key is available only at the time you create it. If you lose your secret access key, you must add new access keys to your IAM user. You can have a maximum of two access keys. If you already have two, you must delete one key pair before creating a new one. To view instructions, see Managing access keys in the *IAM User Guide*.

I'm an administrator and want to allow others to access Amplify

To allow others to access Amplify, you must create an IAM entity (user or role) for the person or application that needs access. They will use the credentials for that entity to access AWS. You must then attach a policy to the entity that grants them the correct permissions in Amplify.

To get started right away, see Creating your first IAM delegated user and group in the *IAM User Guide*.

I want to allow people outside of my AWS account to access my Amplify resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether Amplify supports these features, see How Amplify works with IAM (p. 112).

- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.

Amplify permissions reference

The following table lists each AWS Amplify Console API operation, the corresponding permissions required to perform the operation, and the AWS resource for which you can grant the permissions. Refer to this table when setting up access control and writing permissions policies that you can attach to an IAM identity (identity-based policies).

Amplify console API operations	Required permissions	Resources
CreateApp	`amplify:CreateApp`	arn:aws:amplify:*region*:*account-id*:apps/*app-id*
CreateBackendEnvironment	`amplify:CreateBackendEnvironment`	arn:aws:amplify:*region*:*account-id*:apps/*app-id*
CreateBranch	`amplify:CreateBranch`	arn:aws:amplify:*region*:*account-id*:apps/*app-id*
CreateDeployment	`amplify:CreateDeployment`	arn:aws:amplify:*region*:*account-id*:apps/*app-id*/branches/*branch-name*
CreateDomainAssociation	`amplify:CreateDomainAssociation`	arn:aws:amplify:*region*:*account-id*:apps/*app-id*
CreateWebhook	`amplify:CreateWebhook`	arn:aws:amplify:*region*:*account-id*:apps/*app-id*/branches/*branch-name*
DeleteApp	`amplify:DeleteApp`	arn:aws:amplify:*region*:*account-id*:apps/*app-id*
DeleteBackendEnvironment	`amplify:DeleteBackendEnvironment`	arn:aws:amplify:*region*:*account-id*:apps/*app-id*
DeleteBranch	`amplify:DeleteBranch`	arn:aws:amplify:*region*:*account-id*:apps/*app-id*/branches/*branch-name*
DeleteDomainAssociation	`amplify:DeleteDomainAssociation`	arn:aws:amplify:*region*:*account-id*:apps/*app-id*/domains/*domain-name*
DeleteJob	`amplify:DeleteJob`	arn:aws:amplify:*region*:*account-id*:apps/*app-id*/branches/*branch-name*/jobs/*job-id*

Amplify console API operations	Required permissions	Resources
DeleteWebhook	`amplify:DeleteWebhook`	arn:aws:amplify:*region*:*account-id*:apps/*app-id*
GenerateAccessLogs	`amplify:GenerateAccessLogs`	arn:aws:amplify:*region*:*account-id*:apps/*app-id*
GetApp	`amplify:GetApp`	arn:aws:amplify:*region*:*account-id*:apps/*app-id*
GetArtifactUrl	`amplify:GetArtifactUrl`	arn:aws:amplify:*region*:*account-id*:apps/*app-id*
GetBackendEnvironment	`amplify:GetBackendEnvironment`	arn:aws:amplify:*region*:*account-id*:apps/*app-id*
GetBranch	`amplify:GetBranch`	arn:aws:amplify:*region*:*account-id*:apps/*app-id*/branches/*branch-name*
GetDomainAssociation	`amplify:GetDomainAssociation`	arn:aws:amplify:*region*:*account-id*:apps/*app-id*/domains/*domain-name*
GetJob	`amplify:GetJob`	arn:aws:amplify:*region*:*account-id*:apps/*app-id*/branches/*branch-name*/jobs/*job-id*
GetWebhook	`amplify:GetWebhook`	arn:aws:amplify:*region*:*account-id*:apps/*app-id*
ListApps	`amplify:ListApps`	*No required resource*
ListArtifacts	`amplify:ListArtifacts`	arn:aws:amplify:*region*:*account-id*:apps/*app-id*
ListBackendEnvironments	`amplify:ListBackendEnvironments`	arn:aws:amplify:*region*:*account-id*:apps/*app-id*
ListBranches	`amplify:ListBranches`	arn:aws:amplify:*region*:*account-id*:apps/*app-id*
ListDomainAssociations	`amplify:ListDomainAssociations`	arn:aws:amplify:*region*:*account-id*:apps/*app-id*
ListJobs	`amplify:ListJobs`	arn:aws:amplify:*region*:*account-id*:apps/*app-id*/branches/*branch-name*
ListWebhooks	`amplify:ListWebhooks`	arn:aws:amplify:*region*:*account-id*:apps/*app-id*
StartDeployment	`amplify:StartDeployment`	arn:aws:amplify:*region*:*account-id*:apps/*app-id*/branches/*branch-name*

Amplify console API operations	Required permissions	Resources
StartJob	`amplify:StartJob`	arn:aws:amplify:*region*:*account-id*:apps/*app-id*/branches/*branch-name*/jobs/*job-id*
StopJob	`amplify:StopJob`	arn:aws:amplify:*region*:*account-id*:apps/*app-id*/branches/*branch-name*/jobs/*job-id*
TagResource	`amplify:TagResource`	arn:aws:amplify:*region*:*account-id*:apps/*app-id* or arn:aws:amplify:*region*:*account-id*:apps/*app-id*/branches/*branch-name* or arn:aws:amplify:*region*:*account-id*:apps/*app-id*/branches/*branch-name*/jobs/*job-id*
UntagResource	`amplify:UntagResource`	arn:aws:amplify:*region*:*account-id*:apps/*app-id* or arn:aws:amplify:*region*:*account-id*:apps/*app-id*/branches/*branch-name* or arn:aws:amplify:*region*:*account-id*:apps/*app-id*/branches/*branch-name*/jobs/*job-id*
UpdateApp	`amplify:UpdateApp`	arn:aws:amplify:*region*:*account-id*:apps/*app-id*
UpdateBranch	`amplify:UpdateBranch`	arn:aws:amplify:*region*:*account-id*:apps/*app-id*/branches/*branch-name*
UpdateDomainAssociation	`amplify:UpdateDomainAssociation`	arn:aws:amplify:*region*:*account-id*:apps/*app-id*/domains/*domain-name*
UpdateWebhook	`amplify:UpdateWebhook`	arn:aws:amplify:*region*:*account-id*:apps/*app-id*

The following table lists each Amplify Admin UI API operation, the corresponding permissions required to perform the operation, and the AWS resource for which you can grant the permissions.

Admin UI API operations	Required permissions	Resources
CloneBackend	`amplifybackend:CloneBackend`	arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id*
CreateBackend	`amplifybackend:CreateBackend`	arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id*
CreateBackendAPI	`amplifybackend:CreateBackendAPI`	arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id* arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id*/environments arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id*/api
CreateBackendAuth	`amplifybackend:CreateBackendAuth`	arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id* arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id*/environments arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id*/auth
CreateBackendConfig	`amplifybackend:CreateBackendConfig`	arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id*
CreateToken	`amplifybackend:CreateToken`	arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id*
DeleteBackend	`amplifybackend:DeleteBackend`	arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id* arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id*/environments
DeleteBackendAPI	`amplifybackend:DeleteBackendAPI`	arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id* arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id*/environments arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id*/api
DeleteBackendAuth	`amplifybackend:DeleteBackendAuth`	arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id* arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id*/environments

Admin UI API operations	Required permissions	Resources
		arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id*/auth
DeleteToken	`amplifybackend:DeleteToken`	arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id*
GenerateBackendAPIModels	`amplifybackend:GenerateBackendAPIModels`	arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id* arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id*/environments arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id*/api
GetBackend	`amplifybackend:GetBackend`	arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id* arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id*/environments
GetBackendAPI	`amplifybackend:GetBackendAPI`	arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id* arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id*/environments arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id*/api
GetBackendAPIModels	`amplifybackend:GetBackendAPIModels`	arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id* arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id*/environments arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id*/api
GetBackendAuth	`amplifybackend:GetBackendAuth`	arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id* arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id*/environments arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id*/auth
GetBackendJob	`amplifybackend:GetBackendJob`	arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id* arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id*/job

Admin UI API operations	Required permissions	Resources
GetToken	`amplifybackend:GetToken`	arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id*
ListBackendJobs	`amplifybackend:ListBackendJobs`	arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id* arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id*/job
RemoveAllBackends	`amplifybackend:RemoveAllBackends`	arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id* arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id*/environments
RemoveBackendConfig	`amplifybackend:RemoveBackendConfig`	arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id*
UpdateBackendAPI	`amplifybackend:UpdateBackendAPI`	arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id* arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id*/environments arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id*/api
UpdateBackendAuth	`amplifybackend:UpdateBackendAuth`	arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id* arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id*/environments arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id*/auth
UpdateBackendConfig	`amplifybackend:UpdateBackendConfig`	arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id*
UpdateBackendJob	`amplifybackend:UpdateBackendJob`	arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id* arn:aws:amplifybackend:*region*:*account-id*:backend/*app-id*/job

Security event logging and monitoring in Amplify

Monitoring is an important part of maintaining the reliability, availability, and performance of Amplify and your other AWS solutions. AWS provides the following monitoring tools to watch Amplify, report when something is wrong, and take automatic actions when appropriate:

- *Amazon CloudWatch* monitors in real time your AWS resources and the applications that you run on AWS. You can collect and track metrics, create customized dashboards, and set alarms that notify you

or take actions when a certain metric reaches a threshold that you specify. For example, you can have CloudWatch track CPU usage or other metrics of your Amazon Elastic Compute Cloud (Amazon EC2) instances and automatically launch new instances when needed. For more information about using CloudWatch metrics and alarms with Amplify, see Monitoring (p. 92).

- *Amazon CloudWatch Logs* enables you to monitor, store, and access your log files from Amazon EC2 instances, AWS CloudTrail, and other sources. CloudWatch Logs can monitor information in the log files and notify you when certain thresholds are met. You can also archive your log data in highly durable storage. For more information, see the Amazon CloudWatch Logs User Guide.
- *AWS CloudTrail* captures API calls and related events made by or on behalf of your AWS account and delivers the log files to an Amazon Simple Storage Service (Amazon S3) bucket that you specify. You can identify which users and accounts called AWS, the source IP address from which the calls were made, and when the calls occurred. For more information, see Logging Amplify API calls using AWS CloudTrail (p. 104).
- *Amazon EventBridge* is a serverless event bus service that makes it easy to connect your applications with data from a variety of sources. EventBridge delivers a stream of real-time data from your own applications, Software-as-a-Service (SaaS) applications, and AWS services, and routes that data to targets such as AWS Lambda. This enables you to monitor events that happen in services and build event-driven architectures. For more information, see the Amazon EventBridge User Guide.

Data Protection in Amplify

AWS Amplify conforms to the AWS shared responsibility model, which includes regulations and guidelines for data protection. AWS is responsible for protecting the global infrastructure that runs all the AWS services. AWS maintains control over data hosted on this infrastructure, including the security configuration controls for handling customer content and personal data. AWS customers and APN partners, acting either as data controllers or data processors, are responsible for any personal data that they put in the AWS Cloud.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management (IAM), so that each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon S3.

We strongly recommend that you never put sensitive identifying information, such as your customers' account numbers, into free-form fields such as a **Name** field. This includes when you work with Amplify or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into Amplify or other services might get picked up for inclusion in diagnostic logs. When you provide a URL to an external server, don't include credentials information in the URL to validate your request to that server.

For more information about data protection, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

Encryption at rest

Encryption at rest refers to protecting your data from unauthorized access by encrypting data while stored. Amplify encrypts an app's build artifacts by default using AWS KMS keys for Amazon S3 that are managed by the AWS Key Management Service.

Amplify uses Amazon CloudFront to serve your app to your customers. CloudFront uses SSDs which are encrypted for edge location points of presence (POPs), and encrypted EBS volumes for Regional Edge Caches (RECs). Function code and configuration in CloudFront Functions is always stored in an encrypted format on the encrypted SSDs on the edge location POPs, and in other storage locations used by CloudFront.

Encryption in transit

Encryption in transit refers to protecting your data from being intercepted while it moves between communication endpoints. Amplify Console provides encryption for data in-transit by default. All communication between customers and Amplify and between Amplify and its downstream dependencies is protected using TLS connections that are signed using the Signature Version 4 signing process. All Amplify Console endpoints use SHA-256 certificates that are managed by AWS Certificate Manager Private Certificate Authority. For more information, see Signature Version 4 signing process and What is ACM PCA.

Encryption key management

AWS Key Management Service (KMS) is a managed service for creating and controlling AWS KMS keys, the encryption keys used to encrypt customer data. AWS Amplify generates and manages cryptographic keys for encrypting data on behalf of customers. There are no encryption keys for you to manage.

Compliance Validation for AWS Amplify

Third-party auditors assess the security and compliance of AWS Amplify as part of multiple AWS compliance programs. These include SOC, PCI, ISO, HIPAA, MTCS, C5, K-ISMS, ENS High, OSPAR, HITRUST CSF, and FINMA.

To learn whether Amplify or other AWS services are in scope of specific compliance programs, see AWS Services in Scope by Compliance Program. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security and Compliance Quick Start Guides – These deployment guides discuss architectural considerations and provide steps for deploying baseline environments on AWS that are security and compliance focused.
- Architecting for HIPAA Security and Compliance Whitepaper – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.

  Note
  Not all services are compliant with HIPAA.
- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.

- Evaluating Resources with Rules in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.
- AWS Audit Manager – This AWS service helps you continuously audit your AWS usage to simplify how you manage risk and compliance with regulations and industry standards.

Infrastructure Security in AWS Amplify

As a managed service, AWS Amplify is protected by the AWS global network security procedures that are described in the Amazon Web Services: Overview of Security Processes whitepaper.

You use AWS published API calls to access Amplify through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

AWS Amplify reference

Use the topics in this section to find detailed reference material for AWS Amplify.

Topics

AWS CloudFormation support

Use AWS CloudFormation templates to provision Amplify Console resources, enabling repeatable and reliable web app deployments. AWS CloudFormation provides a common language for you to describe and provision all the infrastructure resources in your cloud environment and simplifies the roll out across multiple AWS accounts and/or regions with just a couple of clicks.

For more information, see the Amplify Console CloudFormation documentation

AWS Command Line Interface support

Use the AWS Command Line Interface to create Amplify Console apps programmatically from the command line. For information, see the AWS CLI documentation.

Resource tagging support

You can use the AWS Command Line Interface to tag Amplify resources. For more information, see the AWS CLI tag-resource documentation.

Document history for AWS Amplify

The following table describes the important changes to the documentation since the last release of AWS Amplify.

- **Latest documentation update:** September 8, 2021

Change	Description	Date
Updated Server side rendering chapter	Updated the Deploy and host server-side rendered apps with Amplify (p. 14) chapter to describe new support for access control for Next.js SSR apps.	September 8, 2021
New managed policies topic	Added the AWS managed policies for AWS Amplify (p. 119) topic to describe the AWS managed policies for Amplify and recent changes to those policies.	July 28, 2021
Updated Server side rendering chapter	Updated the Deploy and host server-side rendered apps with Amplify (p. 14) chapter to describe new support for Next.js version 10.*x.x* and Next.js version 11.	July 22, 2021
Updated Configuring build settings chapter	Added the Monorepo build settings (p. 41) topic to describe how to configure the build settings and the new `AMPLIFY_MONOREPO_APP_ROOT` environment variable when deploying a monorepo app with Amplify.	July 20, 2021
Updated Feature branch deployments chapter	Added the Automatic build-time generation of Amplify config (p. 57) topic to describe how to autogenerate the `aws-exports.js` file at build-time. Added the Conditional backend builds (p. 58) topic to describe how to enable conditional backend builds. Added the Use Amplify backends across apps (p. 58) topic to describe how to reuse existing backends when you create a new app, connect a new branch to an existing app, or update an	June 30, 2021

Change	Description	Date
	existing frontend to point to a different backend environment.	
Updated Security chapter	Added the Data Protection in Amplify (p. 133) topic to describe how to apply the shared responsibility model and how Amplify uses encryption to protect your data at rest and in transit.	June 3, 2021
New support for SSR feature launch	Added the Deploy and host server-side rendered apps with Amplify (p. 14) chapter to describe Amplify support for web apps that use server-side rendering (SSR) and are created with Next.js.	May 18, 2021
New security chapter	Added the Security in Amplify (p. 108) chapter to describe how to apply the shared responsibility model when using Amplify and how to configure Amplify to meet your security and compliance objectives.	March 26, 2021
Updated custom builds topic	Updated the Custom build images and live package updates (p.) topic to describe how to configure a custom build image hosted in Amazon Elastic Container Registry Public.	March 12, 2021
Updated monitoring topic	Updated the Monitoring (p.) topic to describe how to access Amazon CloudWatch metrics data and set alarms.	February 2, 2021
New CloudTrail logging topic	Added the Logging Amplify API calls using AWS CloudTrail (p.) topic to describe how AWS CloudTrail captures and logs all of the API actions for the AWS Amplify Console API Reference and the AWS Amplify Admin UI API Reference.	February 2, 2021

Change	Description	Date
New Admin UI feature launch	Updated the Welcome to the AWS Amplify Console (p. 1) topic to describe the new Admin UI that provides a visual interface for frontend web and mobile developers to create and manage app backends outside the AWS Management Console.	December 1, 2020
New performance mode feature launch	Updated the Managing app performance (p.) topic to describe how to enable performance mode to optimize for faster hosting performance.	November 4, 2020
Updated the custom headers topic	Updated the Custom headers (p.) topic to describe how to define custom headers for an Amplify app using the console or by editing a YML file.	October 28, 2020
New auto subdomains feature launch	Added the Set up automatic subdomains for a Route 53 custom domain (p.) topic to describe how to use pattern-based feature branch deployments for an app connected to an Amazon Route 53 custom domain. Added the Web preview access with subdomains (p. 68) topic to describe how to set up web previews from pull requests to be accessible with subdomains.	June 20, 2020
New notifications topic	Added the Notifications (p.) topic to describe how to set up email notifications for an Amplify app to alert stakeholders or team members when a build succeeds or fails.	June 20, 2020
Updated the custom domains topic	Updated the Set up custom domains (p. 22) topic to improve the procedures for adding custom domains in Amazon Route 53 , GoDaddy, and Google Domains. This update also includes new troubleshooting information for setting up custom domains.	May 12, 2020
AWS Amplify release	This release introduces the Amplify Console.	November 26, 2018