User Guide

AWS Amplify Hosting

AWS Amplify Hosting: User Guide

Table of Contents

What is AWS Amplify Hosting? 1
Supported frameworks 1
Amplify Hosting features 2
Getting started with Amplify Hosting 2
Building a backend 3
Amplify Hosting pricing 3
Getting started tutorials 4
Deploy a Next.js app 4
Step 1: Connect a repository 4
Step 2: Confirm the build settings 5
Step 3: Deploy the application 6
Step 4: (Optional) clean up resources 7
Add features to your app 7
Deploy a Nuxt.js app 8
Deploy an Astro.js app 8
Deploy a SvelteKit app 11
Deploying SSR applications 14
Next.js 15
Next.js feature support 16
Deploying a Next.js SSR application to Amplify 17
Migrating a Next.js 11 SSR app to Amplify Hosting compute 21
Adding SSR functionality to a static Next.js app 22
Making environment variables accessible to server-side runtimes 25
Deploying a Next.js app in a monorepo 27
Nuxt.js 28
Astro.js 28
SvelteKit 29
Deploying an SSR app to Amplify 29
SSR supported features 30
Node.js version support for Next.js apps 31
Image optimization for SSR apps 31
Amazon CloudWatch Logs for SSR apps 31
Amplify Next.js 11 SSR support 32
Troubleshooting SSR deployments 40

Advanced: Open source adapters ... 40
Deployment specification ... 40
Deploying an Express server ... 64
Image optimization for framework authors ... 70
Using open source adapters for any SSR framework ... 78
Deploying a static website from S3 ... 80
Deploying from the Amplify console ... 81
Creating a bucket policy to deploy using the SDKs ... 81
Updating a static website deployed from an S3 bucket ... 83
Updating an S3 deployment to use a bucket and prefix instead of a .zip file ... 84
Deploying without Git ... 86
Drag and drop manual deployments ... 86
Amazon S3 or URL manual deployment ... 87
Troubleshooting Amazon S3 bucket access for manual deployments ... 88
Build settings and configuration ... 89
Configuring build settings ... 89
Build specification reference ... 90
Editing the build specification ... 93
Monorepo build settings ... 99
Customizing the build image ... 106
Configuring a custom build image for an app ... 107
Using specific package and dependency versions in the build image ... 108
Configuring the build instance ... 108
Understanding build instance types ... 109
Configuring the build instance type in the Amplify console ... 110
Configuring an application's heap memory to utilize large instance types ... 112
Incoming webhooks ... 114
Build notifications ... 114
Setting up email notifications ... 115
Connecting a custom domain ... 116
Understanding DNS terminology and concepts ... 117
DNS terminology ... 117
DNS verification ... 118
Custom domain activation process ... 118
Using SSL/TLS certificates ... 119
Adding a custom domain managed by Amazon Route 53 ... 120

Adding a custom domain managed by a third-party DNS provider 122
Updating DNS records for a domain managed by GoDaddy 127
Updating the SSL/TLS certificate for a domain 131
Managing subdomains 132
To add a subdomain only 132
To add a multilevel subdomain 132
To add or edit a subdomain 133
Setting up wildcard subdomains 133
To add or delete a wildcard subdomain 134
Setting up automatic subdomains for an Amazon Route 53 custom domain 134
Web previews with subdomains 135
Troubleshooting custom domains 135
Firewall support for hosted sites 136
Enable AWS WAF using the console 137
Remove AWS WAF from an app 141
Enable AWS WAF using the CDK 142
How Amplify integrates with AWS WAF 143
Amplify web ACL resource policy 144
Firewall pricing 144
Feature branch deployments 146
Team workflows with fullstack Amplify Gen 2 apps 147
Team workflows with fullstack Amplify Gen 1 apps 147
Feature branch workflow 147
GitFlow workflow 153
Per-developer sandbox 153
Pattern-based feature branch deployments 155
Pattern-based feature branch deployments for an app connected to a custom domain 156
Automatic build-time generation of Amplify config (Gen 1 apps only) 156
Conditional backend builds (Gen 1 apps only) 158
Use Amplify backends across apps (Gen 1 apps only) 158
Reuse backends when creating a new app 159
Reuse backends when connecting a branch to an existing app 160
Edit an existing frontend to point to a different backend 160
Building a backend 162
Create a backend for a Gen 2 app 162
Create a backend for a Gen 1 app 162

Prerequisites ... 162
Step 1: Deploy a frontend ... 163
Step 2: Create a backend ... 164
Step 3: Connect the backend to the frontend ... 165
Next steps ... 167
Advanced deployment features ... 168
Password protect branches ... 168
Pull request previews ... 169
Enable web previews for pull requests ... 170
Web preview access with subdomains ... 172
End-to-end testing ... 172
Adding Cypress tests to an existing Amplify application ... 172
Turning off tests for an Amplify application or branch ... 174
One-click deploy button ... 175
Adding the Deploy to Amplify Hosting button to a repository or blog ... 175
Redirects and rewrites ... 177
Understanding the redirects that Amplify supports ... 177
Understanding the order of redirects ... 178
Understanding how Amplify forwards query parameters ... 179
Creating and editing redirects ... 179
Example redirects and rewrites ... 180
Simple redirects and rewrites ... 181
Redirects for single page web apps (SPA) ... 184
Reverse proxy rewrite ... 184
Trailing slashes and clean URLs ... 185
Placeholders ... 186
Query strings and path parameters ... 186
Region-based redirects ... 188
Using wildcard expressions in redirects and rewrites ... 188
Environment variables ... 190
Amplify environment variable reference ... 190
Frontend framework environment variables ... 196
Setting environment variables ... 196
Create a new backend environment with authentication parameters for social sign-in ... 197
Managing environment secrets ... 198

Using AWS Systems Manager to set environment secrets for an Amplify Gen 1 application ... 199
Accessing environment secrets for a Gen 1 application ... 199
Amplify environment secrets reference ... 200
Custom headers ... 201
YAML reference ... 201
Setting custom headers ... 202
Security custom headers example ... 204
Setting Cache-Control custom headers ... 204
Migrating custom headers ... 205
Monorepo custom headers ... 206
Managing cache configuration ... 208
How Amplify applies cache configuration ... 210
Understanding Amplify's managed cache policies ... 211
Managing cache key cookies ... 213
Including or excluding cookies from the cache key ... 214
Changing the cache key cookie configuration for an app ... 215
Using the Cache-Control header to increase app performance ... 216
Skew protection ... 218
Configuring skew protection ... 219
How skew protection works ... 220
X-Amplify-Dpl header example ... 221
Monitoring applications ... 222
CloudWatch metrics and alarms ... 222
Supported CloudWatch metrics ... 222
Accessing CloudWatch metrics ... 226
Creating CloudWatch alarms ... 226
Accessing CloudWatch Logs for SSR apps ... 228
Access logs ... 228
Retrieving an app's access logs ... 229
Analyzing access logs ... 230
Logging Amplify API calls using AWS CloudTrail ... 230
Amplify information in CloudTrail ... 230
Understanding Amplify log file entries ... 231
Using IAM roles with applications ... 235
Adding a service role to deploy backend resources ... 235

Creating an Amplify service role in the IAM console ... 236
Editing a service role's trust policy to prevent confused deputy ... 236
Adding an SSR Compute role ... 237
Creating an SSR Compute role in the IAM console ... 238
Adding an IAM SSR Compute role to an Amplify app ... 241
Managing IAM SSR Compute role security ... 241
Adding a service role to access CloudWatch Logs ... 242
Unified webhooks for Git repositories ... 244
Getting started with unified webhooks ... 244
Security ... 246
Identity and Access Management ... 246
Audience ... 247
Authenticating with identities ... 248
Managing access using policies ... 251
How Amplify works with IAM ... 253
Identity-based policy examples ... 260
AWS managed policies ... 263
Troubleshooting ... 277
Data Protection ... 279
Encryption at rest ... 280
Encryption in transit ... 280
Encryption key management ... 280
Compliance Validation ... 281
Infrastructure Security ... 282
Logging and monitoring ... 282
Cross-service confused deputy prevention ... 283
Security best practices ... 285
Using cookies with the Amplify default domain ... 286
Quotas ... 287
Troubleshooting ... 290
General issues ... 290
HTTP 429 status code (Too many requests) ... 290
The Amplify console doesn't display the build status and last update time for my app ... 291
Web previews are not being created for new pull requests ... 292
My manual deployment is stuck with a pending status in the Amplify console ... 292
I need to update my application's Node.js version ... 293

AL2023 build image 295
I want to run Amplify functions with the Python runtime 295
I want to run commands that require superuser or root privileges 296
Build issues 296
New commits to my repository aren't triggering Amplify builds 296
My repository name isn't listed in the Amplify console when creating a new application ... 296
My build fails with the `Cannot find module aws-exports` error (Gen 1 apps only) 297
I want to override a build timeout 297
Custom domains 297
I need to verify that my CNAME resolves 298
My domain hosted with a third-party is stuck in the Pending Verification state 299
My domain hosted with Amazon Route 53 is stuck in the Pending Verification state 299
My app with multi-level subdomains is stuck in the Pending Verification state 300
My DNS provider doesn't support A records with fully qualified domain names 301
I get a CNAMEAlreadyExistsException error 301
I get an Additional Verification Required error 303
I get a 404 error on the CloudFront URL 303
I get SSL certificate or HTTPS errors when visiting my domain 303
Server-side rendering (SSR) 304
I need help using a framework adapter 305
Edge API routes cause my Next.js build to fail 305
On-Demand Incremental Static Regeneration isn't working for my app 305
My application's build output exceeds the maximum allowed size 305
My build fails with an out of memory error 38
My application's HTTP response size is too large 308
How do I measure my compute app's start up time locally? 38
Redirects and rewrites 309
Access is denied for certain routes even with the SPA redirect rule. 309
I want to set up a reverse proxy to an API 310
Caching 310
I want to reduce the size of the cache for an app 310
I want to disable reading from the cache for an app 311
Setting up GitHub access 311
Installing and authorizing the Amplify GitHub App for a new deployment 312
Migrating an existing OAuth app to the Amplify GitHub App 313

Setting up the Amplify GitHub App for AWS CloudFormation, CLI, and SDK deployments 314
Setting up web previews with the Amplify GitHub App 315
AWS Amplify Hosting reference 316
AWS CloudFormation support 316
AWS Command Line Interface support 316
Resource tagging support 316
Amplify Hosting API 316
Document history 317

Welcome to AWS Amplify Hosting

Amplify Hosting provides a Git-based workflow for hosting full-stack serverless web applications with continuous deployment. Amplify deploys your app to the AWS global content delivery network (CDN). This user guide provides the information you need to get started with Amplify Hosting.

Supported frameworks

Amplify Hosting supports many common SSR frameworks, single-page application (SPA) frameworks, and static site generators, including the following.

SSR frameworks

- Next.js
- Nuxt
- Astro with a community adapter
- SvelteKit with a community adapter
- Any SSR framework with a custom adapter

SPA frameworks

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

Static site generators

- Eleventy
- Gatsby
- Hugo
- Jekyll

- VuePress

Amplify Hosting features

Feature branches

Manage production and staging environments for your frontend and backend by connecting new branches.

Custom domains

Connect your application to a custom domain.

Pull request previews

Preview changes during code reviews.

End-to-end testing

Improve your app quality with end-to-end tests.

Password protected branches

Password protect your web app so you can work on new features without making them publicly accessible.

Redirects and rewrites

Set up rewrites and redirects to maintain SEO rankings and route traffic based on your client app requirements.

Atomic deployments

Atomic deployments eliminate maintenance windows by ensuring that your web app is updated only after the entire deployment finishes. This eliminates scenarios where files fail to upload properly.

Getting started with Amplify Hosting

To get started with Amplify Hosting, see the Getting started with deploying an app to Amplify Hosting tutorial. After completing the tutorial, you will know how to connect a web app in a Git repository (GitHub, BitBucket, GitLab, or AWS CodeCommit) and deploy it to Amplify Hosting with continuous deployment.

Building a backend

AWS Amplify Gen 2 introduces a TypeScript-based, code-first developer experience for defining backends. To learn how to use Amplify Gen 2 to build and connect a backend to your app, see Build & connect backend in the *Amplify docs*.

To better understand Amplify Gen 2's code-first approach, see the Amplify Gen 2 Workshop on the *AWS Workshop Studio* website. In this comprehensive tutorial, you build a serverless application with React and Next.js and learn how to use Amplify Gen 2 Data and Auth libraries and the Amplify UI library to add functionality to the application.

If you are looking for the documentation for building backends for a Gen 1 app, using the CLI and Amplify Studio, see Build & connect backend in the *Gen 1 Amplify docs*.

Amplify Hosting pricing

AWS Amplify only charges you for what you use. For more information, see AWS Amplify Pricing.

Getting started with deploying an app to Amplify Hosting

To help you understand how Amplify Hosting works, the following tutorials walk you through building and deploying applications created using common SSR frameworks that Amplify supports.

Tutorials

- Deploy a Next.js app to Amplify Hosting
- Deploy a Nuxt.js app to Amplify Hosting
- Deploy an Astro.js app to Amplify Hosting
- Deploy a SvelteKit app to Amplify Hosting

Deploy a Next.js app to Amplify Hosting

This tutorial walks you through building and deploying a Next.js application from a Git repository.

Before you begin this tutorial, complete the following prerequisites.

Sign up for an AWS account

If you are not already an AWS customer, you need to create an AWS account by following the online instructions. Signing up enables you to access Amplify and other AWS services that you can use with your application.

Create an application

Create a basic Next.js application to use for this tutorial, using the create-next-app instructions in the *Next.js documentation*.

Create a Git repository

Amplify supports GitHub, Bitbucket, GitLab, and AWS CodeCommit. Push your `create-next-app` application to your Git repository.

Step 1: Connect a Git repository

In this step, you connect your Next.js application in a Git repository to Amplify Hosting.

To connect an app in a Git repository

1. Open the Amplify console.
2. If you are deploying your first app in the current Region, by default you will start from the **AWS Amplify** service page.

   Choose **Deploy an app** at the top of the page.
3. On the **Start building with Amplify** page, choose your Git repository provider, then choose **Next**.

   For GitHub repositories, Amplify uses the GitHub Apps feature to authorize Amplify access. For more information about installing and authorizing the GitHub App, see Setting up Amplify access to GitHub repositories.

   > **Note**
   >
   > After you authorize the Amplify console with Bitbucket, GitLab, or AWS CodeCommit, 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.
4. On the **Add repository branch** page do the following:

   a. Select the name of the repository to connect.

   b. Select the name of the repository branch to connect.

   c. Choose **Next**.

Step 2: Confirm the build settings

Amplify automatically detects the sequence of build commands to run for the branch you are deploying. In this step you review and confirm your build settings.

To confirm the build settings for an app

1. On the **App settings** page, locate the **Build settings** section.

   Verify that the **Frontend build command** and **Build output directory** are correct. For this Next.js example app, the **Build output directory** is set to `.next`.

2. The procedure for adding a service role varies depending on whether you want to create a new role or use an existing one.
   - To create a new role:
     - Choose **Create and use a new service role**.
   - To use an existing role:
     a. Choose **Use an existing role**.
     b. In the service role list, select the role to use.
3. Choose **Next**.

Step 3: Deploy the application

In this step you deploy your app to the AWS global content delivery network (CDN).

To save and deploy an app

1. On the **Review** page, confirm that your repository details and app settings are correct.
2. Choose **Save and deploy**. Your front end build typically takes 1 to 2 minutes, but can vary based on the size of the app.
3. When the deployment is complete, you can view your app using the link to the `amplifyapp.com` default domain.

> **Note**
>
> To augment the security of your Amplify applications, the *amplifyapp.com* domain is registered in the Public Suffix List (PSL). For further security, we recommend that you use cookies with a `__Host-` prefix if you ever need to set sensitive cookies in the default domain name for your Amplify applications. This practice will help to defend your domain against cross-site request forgery attempts (CSRF). For more information see the Set-Cookie page in the Mozilla Developer Network.

Step 4: (Optional) clean up resources

If you no longer need the app you deployed for the tutorial, you can delete it. This step helps ensure that you aren't charged for resources that you aren't using.

To delete an app

1. From the **App settings** menu in the navigation pane, choose **General settings**.
2. On the **General settings** page, choose **Delete app**.
3. In the confirmation window, enter **`delete`**. Then choose **Delete app**.

Add features to your app

Now that you have an app deployed to Amplify, you can explore some of the following features that are available to your hosted application.

Environment variables

Applications often need configuration information at runtime. These configurations can be database connection details, API keys, or parameters. Environment variables provide a way to expose these configurations at build time. For more information, see Environment variables.

Custom domains

In this tutorial, Amplify hosts your app for you on the default `amplifyapp.com` domain with a URL such as `https://branch-name.d1m7bkiki6tdw1.amplifyapp.com`. When you connect your app to a custom domain, users see that your app is hosted on a custom URL, such as `https://www.example.com`. For more information, see Setting up custom domains.

Pull request previews

Web pull request previews offer teams a way to preview changes from pull requests (PRs) before merging code to a production or integration branch. For more information, see Web previews for pull requests.

Manage multiple environments

To learn how Amplify works with feature branches and GitFlow workflows to support multiple deployments, see Feature branch deployments and team workflows.

Deploy a Nuxt.js app to Amplify Hosting

Use the following instructions to deploy a Nuxt.js application to Amplify Hosting. Nuxt has implemented a preset adapter using the Nitro server. This enables you to deploy a Nuxt project without any additional configuration.

To deploy a Nuxt app to Amplify Hosting

1. Sign in to the AWS Management Console and open the Amplify console.
2. On the **All apps** page, choose **Create new app**.
3. On the **Start building with Amplify** page, choose your Git repository provider, then choose **Next**.
4. On the **Add repository branch** page, do the following:
   a. Select the name of the repository to connect.
   b. Select the name of the repository branch to connect.
   c. Choose **Next**.
5. If you want Amplify to be able to deliver app logs to Amazon CloudWatch Logs, you must explicitly enable this in the console. Open the **Advanced settings** section, then choose **Enable SSR app logs** in the **Server-Side Rendering (SSR) deployment** section.
6. Choose **Next**.
7. On the **Review** page, choose **Save and deploy**.

Deploy an Astro.js app to Amplify Hosting

Use the following instructions to deploy an Astro.js application to Amplify Hosting. You can use an existing application, or create a starter application using one of the official examples that Astro provides. To create a starter application, see Use a theme or starter template in the *Astro documentation*.

To deploy an Astro site with SSR to Amplify Hosting, you must add an adapter to your application. We do not maintain an Amplify owned adapter for the Astro framework. This tutorial uses the `astro-aws-amplify` adapter that was created by a member of the community. This adapter is available at github.com/alexnguyennz/astro-aws-amplify on the GitHub website. AWS does not maintain this adapter.

To deploy an Astro app to Amplify Hosting

1. On your local computer, navigate to the Astro application to deploy.
2. To install the adapter, open a terminal window and run the following command. This example uses the community adapter available at github.com/alexnguyennz/astro-aws-amplify. You can replace *astro-aws-amplify* with the name of the adapter that you are using.

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

3. In the project folder for your Astro app, open the `astro.config.mjs` file. Update the file to add the adapter. The file should look like the following.

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

import sitemap from '@astrojs/sitemap';

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

4. Commit the change and push the project to your Git repository.

   Now you are ready to deploy your Astro app to Amplify.
5. Sign in to the AWS Management Console and open the Amplify console.
6. On the **All apps** page, choose **Create new app**.
7. On the **Start building with Amplify** page, choose your Git repository provider, then choose **Next**.
8. On the **Add repository branch** page, do the following:
   a. Select the name of the repository to connect.
   b. Select the name of the repository branch to connect.
   c. Choose **Next**.

9. On the **App settings** page, locate the **Build settings** section. For **Build output directory** enter **`.amplify-hosting`**.
10. You must also update the app's frontend build commands in the build specification. To open the build specification, choose **Edit YML file**.
11. In the `amplify.yml` file, locate the frontend build commands section. Enter **`mv node_modules ./.amplify-hosting/compute/default`**

    Your build settings file should look like the following.

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

12. Choose **Save**.
13. If you want Amplify to be able to deliver app logs to Amazon CloudWatch Logs, you must explicitly enable this in the console. Open the **Advanced settings** section, then choose **Enable SSR app logs** in the **Server-Side Rendering (SSR) deployment** section.
14. Choose **Next**.
15. On the **Review** page, choose **Save and deploy**.

Deploy a SvelteKit app to Amplify Hosting

Use the following instructions to deploy a SvelteKit application to Amplify Hosting. You can use your own application, or create a starter app. For more information, see Creating a project in the *SvelteKit documentation*.

To deploy a SvelteKit app with SSR to Amplify Hosting, you must add an adapter to your project. We do not maintain an Amplify owned adapter for the SvelteKit framework. In this example, we are using the `amplify-adapter` created by a member of the community. The adapter is available at github.com/gzimbron/amplify-adapter on the GitHub website. AWS does not maintain this adapter.

To deploy a SvelteKit app to Amplify Hosting

1. On your local computer, navigate to the SvelteKit application to deploy.
2. To install the adapter, open a terminal window and run the following command. This example uses the community adapter available at github.com/gzimbron/amplify-adapter. If you are using a different community adapter, replace *`amplify-adapter`* with the name of your adapter.

   ```
   npm install amplify-adapter
   ```

3. In the project folder for your SvelteKit app, open the `svelte.config.js` file. Edit the file to use the `amplify-adapter` or replace *`'amplify-adapter'`* with the name of your adapter. The file should look like the following.

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


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

           kit: {
                   // adapter-auto only supports some environments, see https://
   kit.svelte.dev/docs/adapter-auto for a list.
   ```

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

export default config;
```

4. Commit the change and push the application to your Git repository.
5. Now you are ready to deploy your SvelteKit app to Amplify.

   Sign in to the AWS Management Console and open the Amplify console.
6. On the **All apps** page, choose **Create new app**.
7. On the **Start building with Amplify** page, choose your Git repository provider, then choose **Next**.
8. On the **Add repository branch** page, do the following:

   a. Select the name of the repository to connect.

   b. Select the name of the repository branch to connect.

   c. Choose **Next**.
9. On the **App settings** page, locate the **Build settings** section. For **Build output directory** enter **`build`**.
10. You must also update the app's frontend build commands in the build specification. To open the build specification, choose **Edit YML file**.
11. In the `amplify.yml` file, locate the frontend build commands section. Enter **`- cd build/compute/default/`** and **`- npm i --production`**.

    Your build settings file should look like the following.

```
version: 1
frontend:
    phases:
        preBuild:
            commands:
                - 'npm ci --cache .npm --prefer-offline'
        build:
            commands:
```

```
                - 'npm run build'
                - 'cd build/compute/default/'
                - 'npm i --production'

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

12. Choose **Save**.
13. If you want Amplify to be able to deliver app logs to Amazon CloudWatch Logs, you must explicitly enable this in the console. Open the **Advanced settings** section, then choose **Enable SSR app logs** in the **Server-Side Rendering (SSR) deployment** section.
14. Choose **Next**.
15. On the **Review** page, choose **Save and deploy**.

Deploying server-side rendered applications with Amplify Hosting

You can use AWS Amplify to deploy and host web apps that use server-side rendering (SSR). Amplify Hosting automatically detects applications created using the Next.js framework and you don't have to perform any manual configuration in the AWS Management Console.

Amplify also supports any Javascript based SSR framework with an open source build adapter that transforms an application's build output into the directory structure that Amplify Hosting expects. For example, you can deploy app's created with the Nuxt, Astro, and SvelteKit frameworks by installing the available adapters.

Advanced users can use the deployment specification to create a build adapter or configure a post-build script.

You can deploy the following frameworks to Amplify Hosting with minimal configuration.

Next.js

- Amplify supports Next.js 15 applications without the need for an adapter. To get started, see Amplify support for Next.js.

Nuxt.js

- Amplify supports Nuxt.js application deployments with a preset adapter. To get started, see Amplify support for Nuxt.js.

Astro.js

- Amplify supports Astro.js application deployments with a community adapter. To get started, see Amplify support for Astro.js.

SvelteKit

- Amplify supports SvelteKit application deployments with a community adapter. To get started, see Amplify support for SvelteKit.

Open source adapters

- **Use an open source adapter -** For instructions on using any adapter that isn't in the preceding list, see Using open source adapters for any SSR framework .
- **Build a framework adapter -** Framework authors that want to integrate features that a framework provides, can use the Amplify Hosting deployment specification to configure your

build output to conform to the structure that Amplify expects. For more information, see Using the Amplify Hosting deployment specification to configure build output.

- **Configure a post-build script -** You can use the Amplify Hosting deployment specification to manipulate your build output as needed for specific scenarios. For more information, see Using the Amplify Hosting deployment specification to configure build output. For an example, see Deploying an Express server using the deployment manifest.

Topics

- Amplify support for Next.js
- Amplify support for Nuxt.js
- Amplify support for Astro.js
- Amplify support for SvelteKit
- Deploying an SSR app to Amplify
- SSR supported features
- Troubleshooting SSR deployments
- Advanced: Open source adapters

Amplify support for Next.js

Amplify supports deployment and hosting for server-side rendered (SSR) web apps created using Next.js. Next.js is a React framework for developing SPAs with JavaScript. You can deploy apps built with Next.js versions up through Next.js 15, with features such as image optimization and middleware.

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.

Next.js feature support

Amplify Hosting compute fully manages server-side rendering (SSR) for apps built with Next.js versions 12 through 15.

If you deployed a Next.js app to Amplify prior to the release of Amplify Hosting compute in November 2022, your app is using Amplify's previous SSR provider, Classic (Next.js 11 only). Amplify Hosting compute doesn't support apps created using Next.js version 11 or earlier. We strongly recommend that you migrate your Next.js 11 apps to the Amplify Hosting compute managed SSR provider.

The following list describes the specific features that the Amplify Hosting compute SSR provider supports.

Supported features

- Server-side rendered pages (SSR)
- Static pages
- API routes
- Dynamic routes
- Catch all routes
- SSG (Static generation)
- Incremental Static Regeneration (ISR)
- Internationalized (i18n) sub-path routing
- Internationalized (i18n) domain routing
- Internationalized (i18n) automatic locale detection
- Middleware
- Environment variables
- Image optimization
- Next.js 13 app directory

Unsupported features

- Edge API Routes (*Edge middleware is not supported*)
- *On-Demand* Incremental Static Regeneration (ISR)
- Next.js streaming
- Running middleware on static assets and optimized images
- Executing code after a response with `unstable_after` (Experimental feature released with Next.js 15)

Next.js images

The maximum output size of an image can't exceed 4.3 MB. You can have a larger image file stored somewhere and use the Next.js Image component to resize and optimize it into a Webp or AVIF format and then serve it as a smaller size.

Note that the Next.js documentation advises you to install the Sharp image processing module to enable image optimization to work correctly in production. However, this isn't necessary for Amplify deployments. Amplify automatically deploys Sharp for you.

Deploying a Next.js SSR application to Amplify

By default, Amplify deploys new SSR apps using Amplify Hosting's compute service with support for Next.js versions 12 through 15. Amplify Hosting compute fully manages the resources required to deploy an SSR app. SSR apps in your Amplify account that you deployed before November 17, 2022 are using the Classic (Next.js 11 only) SSR provider.

We strongly recommend that you migrate apps using Classic (Next.js 11 only) SSR to the Amplify Hosting compute SSR provider. Amplify doesn't perform automatic migrations for you. You must manually migrate your app and then initiate a new build to complete the update. For instructions, see Migrating a Next.js 11 SSR app to Amplify Hosting compute.

Use the following instructions to deploy a new Next.js SSR app.

To deploy an SSR app to Amplify using the Amplify Hosting compute SSR provider

1. Sign in to the AWS Management Console and open the Amplify console.
2. On the **All apps** page, choose **Create new app**.

3. On the **Start building with Amplify** page, choose your Git repository provider, then choose **Next**.
4. On the **Add repository branch** page, do the following:

   a. In the **Recently updated repositories** list, select the name of the repository to connect.

   b. In the **Branch** list, select the name of the repository branch to connect.

   c. Choose **Next**.
5. The app requires an IAM service role that Amplify assumes when calling other services on your behalf. You can either allow Amplify Hosting compute to automatically create a service role for you or you can specify a role that you have created.
   - To allow Amplify to automatically create a role and attach it to your app:
     - Choose **Create and use a new service role**.
   - To attach a service role that you previously created:

     a. Choose **Use an existing service role**.

     b. Select the role to use from the list.
6. Choose **Next**.
7. On the **Review** page, choose **Save and deploy**.

Package.json file settings

When you deploy a Next.js app, Amplify inspects the app's build script in the `package.json` file to determine the application type.

The following is an example of the build script for a Next.js app. The build script `"next build"` indicates that the app supports both SSG and SSR pages. This build script is also used for Next.js 14 or later SSG only apps.

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

The following is an example of the build script for a Next.js 13 or earlier 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 for a Next.js SSR application

After inspecting your app's `package.json` file, 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 the build settings for an Amplify application.

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/**/*
```

Amplify build settings for a Next.js 13 or earlier SSG application

If Amplify detects that you are deploying a Next.js 13 or earlier SSG app, it generates a build specification 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 `out` directory is the default folder that Next.js creates to store exported static assets. When you configure your app's build specification settings, change the name of the `baseDirectory` folder to match your app's configuration.

The following is an example of the build settings for an app where `baseDirectory` is set to `out` to indicate that the build artifacts are for a Next.js 13 or earlier 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/**/*
```

Amplify build settings for a Next.js 14 or later SSG application

In Next.js version 14, the `next export` command was deprecated and replaced with `output: 'export'` in the `next.config.js` file to enable static exports. If you are deploying a Next.js 14 SSG only application in the console, Amplify generates a buildspec for the app and sets `baseDirectory` to `.next`. If you are deploying an app where an `amplify.yml` file is present, you must manually set the `baseDirectory` to `.next` in the file. This is the same `baseDirectory` setting that Amplify uses for Next.js `WEB_COMPUTE` applications that support both SSG and SSR pages.

The following is an example of the build settings for a Next.js 14 SSG only app with the `baseDirectory` set to `.next`.

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

Migrating a Next.js 11 SSR app to Amplify Hosting compute

When you deploy a new Next.js app, by default Amplify uses the most recent supported version of Next.js. Currently, the Amplify Hosting compute SSR provider supports Next.js version 15.

The Amplify console detects apps in your account that were deployed prior to the November 2022 release of the Amplify Hosting compute service with full support for Next.js versions 12 through 15. The console displays an information banner identifying apps with branches that are deployed using Amplify's previous SSR provider, Classic (Next.js 11 only). We strongly recommend that you migrate your apps to the Amplify Hosting compute SSR provider.

If you are updating your hosted Next.js 11 application to Next.js 12 or later, you might get a `"target" property is no longer supported` error when a deployment is triggered. In this case, You must migrate to Amplify Hosting compute.

You must manually migrate the app and all of its production branches at the same time. An app can't contain both Classic (Next.js 11 only) and Next.js 12 or later branches.

Use the following instructions to migrate an app to the Amplify Hosting compute SSR provider.

To migrate an app to the Amplify Hosting compute SSR provider

1. Sign in to the AWS Management Console and open the Amplify console.
2. Choose the Next.js app that you want to migrate.

> **Note**
>
> Before you migrate an app in the Amplify console, you must first update the app's package.json file to use Next.js version 12 or later.

3. In the navigation pane, choose **App settings**, **General**.
4. On the app homepage, the console displays a banner if the app has branches deployed using the *Classic (Next.js 11 only)* **SSR provider**. On the banner, choose **Migrate**.
5. In the migration confirmation window, select the three statements and choose **Migrate**.
6. Amplify will build and redeploy your app to complete the migration.

Reverting an SSR migration

When you deploy a Next.js app, Amplify Hosting detects the settings in your app and sets the internal platform value for the app. There are three valid platform values. An SSG app is set to the platform value `WEB`. An SSR app using Next.js version 11 is set to the platform value `WEB_DYNAMIC`. A Next.js 12 or later SSR app is set to the platform value `WEB_COMPUTE`.

When you migrate an app using the instructions in the previous section, Amplify changes the platform value of your app from `WEB_DYNAMIC` to `WEB_COMPUTE`. After the migration to Amplify Hosting compute is complete, you can't revert the migration in the console. To revert the migration, you must use the AWS Command Line Interface to change the app's platform back to `WEB_DYNAMIC`. Open a terminal window and enter the following command, updating the app ID and Region with your unique information.

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

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. Before you start the process of converting your SSG app to SSR, update the app to use Next.js version 12 or later and add SSR functionality. Then you will need to perform the following steps.

1. Use the AWS Command Line Interface to change the app's platform type.
2. Add a service role to the app.
3. Update the output directory in the app's build settings.

4. Update the app's `package.json` file to indicate that the app uses SSR.

Updating the platform

There are three valid values for platform type. An SSG app is set to platform type `WEB`. An SSR app using Next.js version 11 is set to platform type `WEB_DYNAMIC`. For apps deployed to Next.js 12 or later using SSR managed by Amplify Hosting compute, the platform type is set to `WEB_COMPUTE`.

When you deployed your app as an SSG app, Amplify set the platform type to `WEB`. Use the AWS CLI to change the platform for your app to `WEB_COMPUTE`. Open a terminal window and enter the following command, updating the text in red with your unique app id and Region.

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

Adding a service role

A service role is the AWS Identity and Access Management (IAM) role that Amplify 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.

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 Adding a service role 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**.

Updating the 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 the build settings for an Amplify application.

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/**/*
```

Updating 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.

Making environment variables accessible to server-side runtimes

Amplify Hosting supports adding environment variables to your application's builds by setting them in the project's configuration in the Amplify console.

However, a Next.js server component doesn't have access to those environment variables by default. This behavior is intentional to protect any secrets stored in environment variables that your application uses during the build phase.

To make specific environment variables accessible to Next.js, you can modify the Amplify build specification file to set them in the environment files that Next.js recognizes. This enables Amplify to load these environment variables before it builds the application.

> **Important**
>
> We strongly recommend that you don't store any credentials, secrets, or sensitive information in your environment variables as any user with access to deployment artifacts can read them.
> To give your SSR compute function access to AWS resources, we recommend using IAM roles.

The following build specification example demonstrates how to add environment variables in the build commands section.

```
version: 1
frontend:
  phases:
    preBuild:
      commands:
        - npm ci
    build:
      commands:
        - env | grep -e API_BASE_URL >> .env.production
        - env | grep -e NEXT_PUBLIC_ >> .env.production
        - npm run build
  artifacts:
    baseDirectory: .next
    files:
      - '**/*'
  cache:
```

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

In this example, the build commands section includes two commands that write environment variables to the `.env.production` file before the application build runs. Amplify Hosting allows your application to access these variables when the application receives traffic.

The following line from the build commands section in the preceding example demonstrates how to take a specific variable from the build environment and add it to the `.env.production` file.

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

If the variables exist in your build environment, the `.env.production` file will contain the following environment variables.

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

The following line from the build commands section in the preceding example demonstrates how to add an environment variable with a specific prefix to the `.env.production` file. In this example, all variables with the prefix `NEXT_PUBLIC_` are added.

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

If multiple variables with the `NEXT_PUBLIC_` prefix exist in the build environment, the `.env.production` file will look similar to the following.

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

SSR environment variables for monorepos

If you are deploying an SSR app in a monorepo and want to make specific environment variables accessible to Next.js, you must prefix the `.env.production` file with your application root. The following example build specification for a Next.js app within a Nx monorepo demonstrates how to add environment variables in the build commands section.

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

The following lines from the build commands section in the preceding example demonstrate how to take specific variables from the build environment and add them to the `.env.production` file for an app in a monorepo with the application root `apps/app`.

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

Deploying a Next.js app in a monorepo

Amplify supports apps in generic monorepos as well as apps in monorepos created using npm workspace, pnpm workspace, Yarn workspace, Nx, and Turborepo. When you deploy your app, Amplify automatically detects the monorepo build framework that you are using. Amplify automatically applies build settings for apps in an npm workspace, Yarn workspace or Nx. Turborepo and pnpm apps require additional configuration. For more information, see Configuring monorepo build settings.

For a detailed Nx example, see the Share code between Next.js apps with Nx on AWS Amplify Hosting blog post.

Amplify support for Nuxt.js

Nuxt is a framework for creating full stack web applications with Vue.js.

Adapter

You can deploy a Nuxt.js application to Amplify using a preset adapter with zero configuration. For more information about the adapter, see the Nuxt documentation.

Tutorial

To learn how to deploy a Nuxt.js app to Amplify, see Deploy a Nuxt.js app to Amplify Hosting.

Demo

For a video demonstration, see *Nuxt Hosting With ZERO Configuration In Minutes (With AWS)* on YouTube.

Amplify support for Astro.js

Astro is a web framework for creating content-driven web applications.

Adapter

You can deploy an Astro.js application to Amplify using a community adapter. We do not maintain an Amplify owned adapter for the Astro framework. However, an adapter is available at github.com/alexnguyennz/astro-aws-amplify on the GitHub website. This adapter was created by a member of the community and is not maintained by AWS.

Tutorial

To learn how to deploy an Astro app to Amplify, see Deploy an Astro.js app to Amplify Hosting.

Demo

For a video demonstration, see *How to deploy an Astro Website to AWS* on the Amazon Web Services YouTube channel.

Amplify support for SvelteKit

SvelteKit is a framework for creating full stack web applications with Svelte.

Adapter

You can deploy a SvelteKit application to Amplify using a community adapter. We do not maintain an Amplify owned adapter for the SvelteKit framework. However, an adapter is available at github.com/gzimbron/amplify-adapter on the GitHub website. This adapter was created by a member of the community and is not maintained by AWS.

Tutorial

To learn how to deploy a SvelteKit app to Amplify, see Deploy a SvelteKit app to Amplify Hosting.

Demo

For a video demonstration, see *How to deploy a SvelteKit website (with API) to AWS* on the Amazon Web Services YouTube channel.

Deploying an SSR app to Amplify

You can use these instructions to deploy an app created with any framework with a deployment bundle that conforms to the build output that Amplify expects. If you're deploying a Next.js application, no adapter is needed.

If you're deploying an SSR app that uses a framework adapter, you must first install and configure the adapter. For instructions, see Using open source adapters for any SSR framework.

To deploy an SSR app to Amplify Hosting

1. Sign in to the AWS Management Console and open the Amplify console.
2. On the **All apps** page, choose **Create new app**.
3. On the **Start building with Amplify** page, choose your Git repository provider, then choose **Next**.
4. On the **Add repository branch** page do the following:

   a. Select the name of the repository to connect.

b. Select the name of the repository branch to connect.

c. Choose **Next**.

5. On the **App settings** page, Amplify automatically detects Next.js SSR apps.

   If you are deploying an SSR app that uses an adapter for another framework, you must explicitly enable Amazon CloudWatch Logs. Open the **Advanced settings** section, then choose **Enable SSR app logs** in the **Server-Side Rendering (SSR) deployment** section.

6. The app requires an IAM service role that Amplify assumes to deliver logs to your AWS account.

   The procedure for adding a service role varies depending on whether you want to create a new role or use an existing one.

   - To create a new role:
     - Choose **Create and use a new service role**.
   - To use an existing role:
     a. Choose **Use an existing role**.
     b. In the service role list, select the role to use.

7. Choose **Next**.

8. On the **Review** page, choose **Save and deploy**.

SSR supported features

This section provides information about Amplify's support for SSR features.

Amplify provides Node.js version support to match the version of Node.js that was used to build your app.

Amplify provides a built-in image optimization feature that supports all SSR apps. If you don't want to use the default image optimization feature, you can implement a custom image optimization loader.

Topics

- Node.js version support for Next.js apps
- Image optimization for SSR apps

- Amazon CloudWatch Logs for SSR apps
- Amplify Next.js 11 SSR support

Node.js version support for Next.js apps

When Amplify builds and deploys a Next.js compute app, it uses the Node.js runtime version that matches the major version of Node.js that was used to build the app.

You can specify the Node.js version to use in the **Live package override** feature in the Amplify console. For more information about configuring live package updates, see Using specific package and dependency versions in the build image. You can also specify the Node.js version using other mechanisms, such as nvm commands. If you don't specify a version, Amplify defaults to use the current version used by the Amplify build container.

Image optimization for SSR apps

Amplify Hosting provides a built-in image optimization feature that supports all SSR apps. With Amplify's image optimization, you can deliver high-quality images in the right format, dimension, and resolution for the device that is accessing them, while maintaining the smallest possible file size.

Currently, you can either use the Next.js Image component to optimize images on-demand or you can implement a custom image loader. If you are using Next.js 13 or later, you don't need to take any further action to use Amplify's image optimization feature. If you are implementing a custom loader, see the following *Using a custom image loader* topic.

Using a custom image loader

If you use a custom image loader, Amplify detects the loader in your application's `next.config.js` file and doesn't utilize the built-in image optimization feature. For more information about the custom loaders that Next.js supports, see the Next.js images documentation.

Amazon CloudWatch Logs for SSR apps

Amplify sends information about your SSR runtime to Amazon CloudWatch Logs in your AWS account. When you deploy an SSR app, the app requires an IAM service role that Amplify assumes when calling other services on your behalf. You can either allow Amplify Hosting compute to automatically create a service role for you or you can specify a role that you have created.

If you choose to allow Amplify to create an IAM role for you, the role will already have the permissions to create CloudWatch Logs. If you create your own IAM role, you will need to add the following permissions to your policy to allow Amplify to access Amazon CloudWatch Logs.

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

For more information about service roles, see Adding a service role with permissions to deploy backend resources.

Amplify Next.js 11 SSR support

If you deployed a Next.js app to Amplify prior to the release of Amplify Hosting compute on November 17, 2022, your app is using Amplify's previous SSR provider, Classic (Next.js 11 only). The documentation in this section applies only to apps deployed using the Classic (Next.js 11 only) SSR provider.

> **Note**
>
> We strongly recommend that you migrate your Next.js 11 apps to the Amplify Hosting compute managed SSR provider. For more information, see Migrating a Next.js 11 SSR app to Amplify Hosting compute.

The following list describes the specific features that the Amplify Classic (Next.js 11 only) SSR provider supports.

Supported features

- Server-side rendered pages (SSR)
- Static pages
- API routes
- Dynamic routes
- Catch all routes
- SSG (Static generation)

- Incremental Static Regeneration (ISR)
- Internationalized (i18n) sub-path routing
- Environment variables

Unsupported features

- Image optimization
- *On-Demand* Incremental Static Regeneration (ISR)
- Internationalized (i18n) domain routing
- Internationalized (i18n) automatic locale detection
- Middleware
- Edge Middleware
- Edge API Routes

Pricing for Next.js 11 SSR apps

When deploying your Next.js 11 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.
- Four Lambda@Edge functions to customize the content that CloudFront delivers.

AWS Identity and Access Management permissions for Next.js 11 SSR apps

Amplify requires AWS Identity and Access Management (IAM) permissions to deploy an SSR app. For SSR apps, Amplify deploys resources such as an Amazon S3 bucket, a CloudFront distribution, Lambda@Edge functions, an Amazon SQS queue (if using ISR) and IAM roles. Without the required minimum permissions, you will get an `Access Denied` error when you try to deploy your SSR app. To provide Amplify with the required permissions, you must specify a service role.

To create an IAM service role that Amplify assumes when calling other services on your behalf, see Adding a service role with permissions to deploy backend resources. These instructions

demonstrate how to create a role that attaches the `AdministratorAccess-Amplify` managed policy.

The `AdministratorAccess-Amplify` managed policy provides access to multiple AWS services, including IAM actions. and should be considered as powerful as the `AdministratorAccess` policy. This policy provides more permissions than required to deploy your SSR app.

It is recommended that you follow the best practice of granting least privilege and reduce the permissions granted to the service role. Instead of granting administrator access permissions to your service role, you can create your own customer managed IAM policy that grants only the permissions required to deploy your SSR app. See, Creating IAM policies in the *IAM User Guide* for instructions on creating a customer managed policy.

If you create your own policy, refer to the following list of the minimum permissions required to deploy an SSR app.

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

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

Troubleshooting Next.js 11 SSR deployments

If you experience unexpected issues when deploying a Classic (Next.js 11 only) SSR app with Amplify, review the following troubleshooting topics.

Topics

- My application's output directory is overridden
- I get a 404 error after deploying my SSR site
- My application is missing the rewrite rule for CloudFront SSR distributions
- My application is too large to deploy
- My build fails with an out of memory error
- My application has both SSR and SSG branches
- My application stores static files in a folder with a reserved path
- My application has reached a CloudFront limit
- Lambda@Edge functions are created in the US East (N. Virginia) Region
- My Next.js application uses unsupported features
- Images in my Next.js application aren't loading
- Unsupported Regions

My application's 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 the build settings for an Amplify application.

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/**/*
```

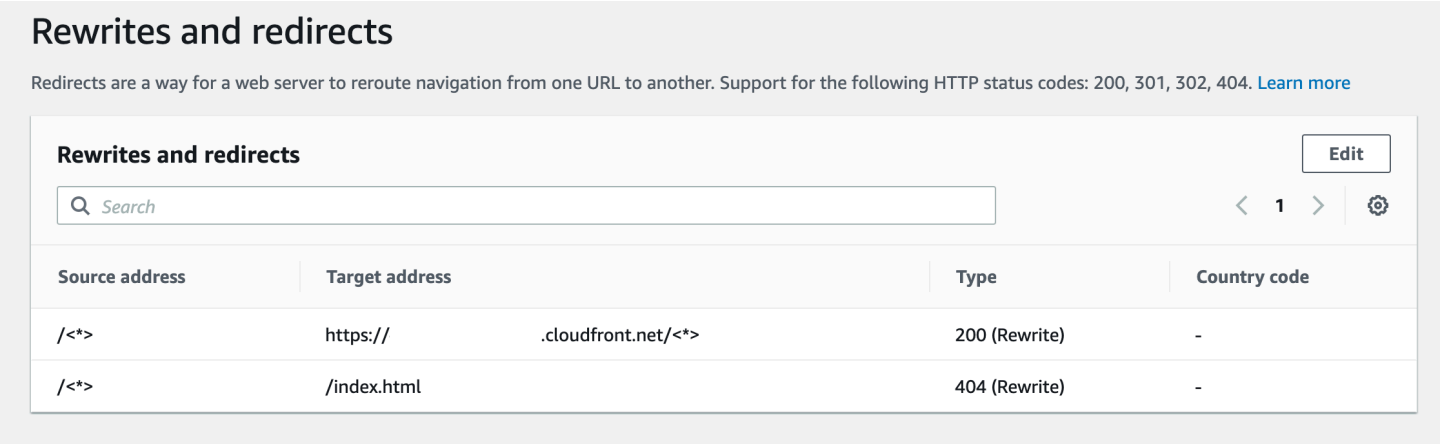
I get a 404 error after deploying my SSR site

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, My application's output directory is overridden.

My application 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 for your app in the Amplify console. 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. Notice that in this example, a CloudFront rewrite rule exists.



My application is too large to deploy

Amplify limits the size of an SSR deployment to 50 MB. 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
      },
}
```

My build fails with an out of memory error

Next.js enables you to cache build artifacts to improve performance on subsequent builds. In addition, Amplify's AWS CodeBuild container compresses and uploads this cache to Amazon S3, on your behalf, to improve subsequent build performance. This could cause your build to fail with an out of memory error.

Perform the following actions to prevent your app from exceeding the memory limit during the build phase. First, remove `.next/cache/**/*` from the cache.paths section of your build settings. Next, remove the `NODE_OPTIONS` environment variable from your build settings file. Instead, set the `NODE_OPTIONS` environment variable in the Amplify console to define the Node maximum memory limit. For more information about setting environment variables using the Amplify console, see Setting environment variables.

After making these changes, try your build again. If it succeeds, add `.next/cache/**/*` back to the cache.paths section of your build settings file.

For more information about Next.js cache configuration to improve build performance, see AWS CodeBuild on the Next.js website.

My application 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.

My application 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.

My application 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*.

Lambda@Edge functions are created in the US East (N. Virginia) Region

When you deploy a Next.js app, Amplify creates Lambda@Edge functions to customize the content that CloudFront delivers. Lambda@Edge functions are created in the US East (N. Virginia) Region, not the Region where your app is deployed. This is a Lambda@Edge restriction. For more information about Lambda@Edge functions, see Restrictions on edge functions in the *Amazon CloudFront Developer Guide.*

My Next.js application uses unsupported features

Apps deployed with Amplify support the Next.js major versions up through version 11. For a detailed list of the Next.js features that are supported and unsupported by Amplify, see supported features.

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 migrate the app to the Amplify Hosting compute SSR provider. For instructions, see Migrating a Next.js 11 SSR app to Amplify Hosting compute.

Images in my Next.js application aren't loading

When you add images to your Next.js app using the `next/image` component, the size of the image can't exceed 1 MB. When you deploy the app to Amplify, images that are larger than 1 MB will return a 503 error. This is caused by a Lambda@Edge limit that restricts the size of a response that is generated by a Lambda function, including headers and body, to 1 MB.

The 1 MB limit applies to other artifacts in your app, such as PDF and document files.

Unsupported Regions

Amplify doesn't support Classic (Next.js 11 only) SSR app deployment in every AWS region where Amplify is available. Classic (Next.js 11 only) 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.

Troubleshooting SSR deployments

If you experience unexpected issues when deploying an SSR app with Amplify Hosting compute, see Troubleshooting server-side rendered applications in the Amplify troubleshooting chapter.

Advanced: Open source adapters

Framework authors can use the file system based deployment specification to develop open source build adapters customized for their specific frameworks. These adapters will transform an app's build output into a deployment bundle that conforms to Amplify Hosting's expected directory structure. This deployment bundle will include all the necessary files and assets to host an app, including runtime configuration, such as routing rules.

If you aren't using a framework, you can develop your own solution to generate a build output that Amplify expects.

Topics

- Using the Amplify Hosting deployment specification to configure build output
- Deploying an Express server using the deployment manifest
- Image optimization integration for framework authors
- Using open source adapters for any SSR framework

Using the Amplify Hosting deployment specification to configure build output

The Amplify Hosting deployment specification is a file system based specification that defines the directory structure that facilitates deployments to Amplify Hosting. A framework can generate this expected directory structure as the output of its build command, enabling the framework to take

advantage of Amplify Hosting's service primitives. Amplify Hosting understands the structure of the deployment bundle and deploys it accordingly.

For a video demonstration that explains how to use the deployment specification, see *How to host any website using AWS Amplify* on the Amazon Web Services YouTube channel.

The following is an example of the folder structure that Amplify expects for the deployment bundle. At a high level, it has a folder named `static`, a folder named `compute` and a deployment manifest file named `deploy-manifest.json`.

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

Amplify SSR primitive support

The Amplify Hosting deployment specification defines a contract that closely maps to the following primitives.

Static assets

Provides frameworks with the ability to host static files.

Compute

Provides frameworks with the ability to run a Node.js HTTP server on port 3000.

Image optimization

Provides frameworks with a service to optimize images at runtime.

Routing rules

Provides frameworks with a mechanism to map incoming request paths to specific targets.

The .amplify-hosting/static directory

You must place all publicly accessible static files that are meant to be served from the application URL in the `.amplify-hosting/static` directory. The files inside this directory are served via the static assets primitive.

Static files are accessible at the root (/) of the application URL without any changes to their content, file name, or extension. Additionally, subdirectories are preserved in the URL structure and appear before the file name. As an example, `.amplify-hosting/static/favicon.ico` will be served from `https://myAppId.amplify-hostingapp.com/favicon.ico` and `.amplify-hosting/static/_nuxt/main.js` will be served from `https://myAppId.amplify-hostingapp.com/_nuxt/main.js`

If a framework supports the ability to modify the base path of the application, it must prepend the base path to the static assets inside the `.amplify-hosting/static` directory. For example, if the base path is `/folder1/folder2`, then the build output for a static asset called `main.css` will be `.amplify-hosting/static/folder1/folder2/main.css`.

The .amplify-hosting/compute directory

A single compute resource is represented by a single subdirectory named `default` contained within the `.amplify-hosting/compute` directory. The path is `.amplify-hosting/compute/default`. This compute resource maps to Amplify Hosting's compute primitive.

The contents of the `default` subdirectory must conform to the following rules.

- A file must exist in the root of the `default` subdirectory, to serve as the entry point to the compute resource.
- The entry point file must be a Node.js module and it must start an HTTP server that listens on port 3000.
- You can place other files in the `default` subdirectory and reference them from code in the entry point file.
- The contents of the subdirectory must be self-contained. Code in the entry point module can't reference any modules outside of the subdirectory. Note that frameworks can bundle their HTTP server in any way that they want. If the compute process can be initiated with the `node server.js` command, where `server.js is` is the name of the entry file, from within the subdirectory, Amplify considers the directory structure to conform to the deployment specification.

Amplify Hosting bundles and deploys all files inside the `default` subdirectory to a provisioned compute resource. Each compute resource is allocated 512 MB of ephemeral storage. This storage isn't shared between execution instances, but is shared among subsequent invocations within the same execution instance. Execution instances are limited to a maximum execution time of 15 minutes, and the only writable path within the execution instance is the `/tmp` directory. The compressed size of each compute resource bundle can't exceed 220 MB. For example, the `.amplify/compute/default` subdirectory can't exceed 220 MB when compressed.

The .amplify-hosting/deploy-manifest.json file

Use the `deploy-manifest.json` file to store the configuration details and metadata for a deployment. At a minimum, a `deploy-manifest.json` file must include a `version` attribute, the `routes` attribute with a catch-all route specified, and the `framework` attribute with framework metadata specified.

The following object definition demonstrates the configuration for a deployment manifest.

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

The following topics describe the details and usage for each attribute in the deployment manifest.

Using the version attribute

The `version` attribute defines the version of the deployment specification that you are implementing. Currently, the only version for the Amplify Hosting deployment specification is version 1. The following JSON example demonstrates the usage for the `version` attribute.

```
"version": 1
```

Using the routes attribute

The `routes` attribute enables frameworks to leverage the Amplify Hosting routing rules primitive. Routing rules provide a mechanism for routing incoming request paths to a specific target in the deployment bundle. Routing rules only dictate the destination of an incoming request and are applied after the request has been transformed by rewrite and redirect rules. For more information about how Amplify Hosting handles rewrites and redirects, see Setting up redirects and rewrites for an Amplify application.

Routing rules don't rewrite or transform the request. If an incoming request matches the path pattern for a route, the request is routed as-is to the route's target.

The routing rules specified in the `routes` array must conform to the following rules.

- A catch-all route must be specified. A catch-all route has the `/*` pattern that matches all incoming requests.
- The `routes` array can contain a maximum of 25 items.
- You must specify either a `Static` route or a `Compute` route.
- If you specify a `Static` route, the `.amplify-hosting/static` directory must exist.
- If you specify a `Compute` route, the `.amplify-hosting/compute` directory must exist.
- If you specify an `ImageOptimization` route, you must also specify a `Compute` route. This is required because image optimization is not yet supported for purely static applications.

The following object definition demonstrates the configuration for the `Route` object.

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

```
}
```

The following table describes the `Route` object's properties.

Key	Type	Required	Description
path	String	Yes	Defines a pattern that matches incoming request paths (excluding querystring). The maximum path length is 255 characters. A path must start with the forward slash `/`. A path can contain any of the following characters: [A-Z], [a-z], [0-9],[_-.*$/~"'@:+]. For pattern matching, only the following wildcard characters are supported: • `*` (matches 0 or more characters) • The `/*` pattern is called a catch-all pattern and will match all incoming requests.

Key	Type	Required	Description
target	Target	Yes	An object that defines the target to route the matched request to. If a `Compute` route is specified, a corresponding `ComputeResource` must exist. If an `ImageOptimization` route is specified, `imageSettings` must also be specified.

Key	Type	Required	Description
fallback	Target	No	An object that defines the target to fallback to if the original target returns a 404 error. The `target` kind and the `fallback` kind can't be the same for a specified route. For example, fallback from `Static` to `Static` is not allowed. Fallbacks are only supported for GET requests that don't have a body. If a body is present in the request, it will be dropped during the fallback.

The following object definition demonstrates the configuration for the `Target` object.

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

The following table describes the `Target` object's properties.

Key	Type	Required	Description
kind	Targetkind	Yes	An `enum` that defines the target type. Valid values are `Static`, `Compute`, and `ImageOptimization` .
src	String	Yes for `Compute` No for other primitives	A string that specifies the name of the subdirectory in the deployment bundle that contains the primitive's executable code. Valid and required only for the Compute primitive. The value must point to one of the compute resources present in the deployment bundle. Currently, the only supported value for this field is `default`.
cacheControl	String	No	A string that specifies the value of the Cache-Control header to apply to the response. Valid only for the Static and the ImageOptimization primitives.

Key	Type	Required	Description
			The specified value is overriden by custom headers. For more informati on about Amplify Hosting customer headers, see Setting custom headers for an Amplify app. **Note** This Cache-Control header is only applied to successful responses with a status code set to 200 (OK).

The following object definition demonstrates the usage for the `TargetKind` enumeration.

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

The following list specifies the valid values for the `TargetKind` enum.

Static

Routes requests to the static assets primitive.

Compute

Routes requests to the compute primitive.

ImageOptimization

Routes requests to the image optimization primitive.

The following JSON example demonstrates the usage for the `routes` attribute with multiple routing rules specified.

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

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

For more information about specifying routing rules in your deployment manifest, see Best practices for configuring routing rules

Using the computeResources attribute

The `computeResources` attribute enables frameworks to provide metadata about the provisioned compute resources. Every compute resource must have a corresponding route associated with it.

The following object definition demonstrates the usage for the `ComputeResource` object.

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

type ComputeRuntime = 'nodejs16.x' | 'nodejs18.x' | 'nodejs20.x';
```

The following table describes the `ComputeResource` object's properties.

Key	Type	Required	Description
name	String	Yes	Specifies the name of the compute resource. The

Key	Type	Required	Description
			name must match the name of the subdirectory inside the `.amplify-hosting/compute` directory . For version 1 of the deployment specification, the only valid value is `default`.
runtime	ComputeRuntime	Yes	Defines the runtime for the provisioned compute resource. Valid values are `nodejs16.x` , `nodejs18.x` , and `nodejs20.x` .
entrypoint	String	Yes	Specifies the name of the starting file that code will run from for the specified compute resource. The file must exist inside the subdirectory that represents a compute resource.

If you have a directory structure that looks like the following.

```
.amplify-hosting
|---compute
```

```
|   |---default
|       |---index.js
```

The JSON for the `computeResource` attribute will look like the following.

```
"computeResources": [
    {
      "name": "default",
      "runtime": "nodejs16.x",
      "entrypoint": "index.js",
    }
  ]
```

Using the imageSettings attribute

The `imageSettings` attribute enables frameworks to customize the behavior of the image optimization primitive, that provides on-demand optimization of images at runtime.

The following object definition demonstrates the usage for the `ImageSettings` object.

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

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

The following table describes the `ImageSettings` object's properties.

Key	Type	Required	Description
sizes	Number[]	Yes	An array of supported image widths.
domains	String[]	Yes	An array of allowed external domains

Key	Type	Required	Description
			that can use image optimization. Leave the array empty to allow only the deployment domain to use image optimization.
remotePatterns	RemotePattern[]	Yes	An array of allowed external patterns that can use image optimization. Similar to domains, but provides more control with regular expressions (regex).
formats	ImageFormat[]	Yes	An array of allowed output image formats.
minimumCacheTTL	Number	Yes	The cache duration in seconds for the optimized images.
dangerouslyAllowSVG	Boolean	Yes	Allows SVG input image URLs. This is disabled by default for security purposes.

The following object definition demonstrates the usage for the `RemotePattern` object.

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

```
  pathname?: string;
}
```

The following table describes the `RemotePattern` object's properties.

Key	Type	Required	Description
protocol	String	No	The protocol of the allowed remote pattern. The only valid value is `https`.
hostname	String	Yes	The hostname of the allowed remote pattern. You can specify a literal or wildcard. A single `*` matches a single subdomain. A double `**` matches any number of subdomains. Amplify doesn't allow blanket wildcards where only `**` is specified.
port	String	No	The port of the allowed remote pattern.
pathname	String	No	The path name of the allowed remote pattern.

The following example demonstrates the `imageSettings` attribute.

```
"imageSettings": {
```

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

Using the framework attribute

Use the `framework` attribute to specify framework metadata.

The following object definition demonstrates the configuration for the `FrameworkMetadata` object.

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

The following table describes the `FrameworkMetadata` object's properties.

Key	Type	Required	Description
name	String	Yes	The name of the framework.

Key	Type	Required	Description
version	String	Yes	The version of the framework. It must be a valid semantic versioning (semver) string.

Best practices for configuring routing rules

Routing rules provide a mechanism for routing incoming request paths to specific targets in the deployment bundle. In a deployment bundle, framework authors can emit files to the build output that are deployed to either of the following targets:

- **Static assets primitive** – Files are contained in the `.amplify-hosting/static` directory.
- **Compute primitive** – Files are contained in the `.amplify-hosting/compute/default` directory.

Framework authors also provide an array of routing rules in the deploy manifest file. Each rule in the array is matched against the incoming request in sequential traversal order, until there's a match. When there's a matching rule, the request is routed to the target specified in the matching rule. Optionally, a fallback target can be specified for each rule. If the original target returns a 404 error, the request is routed to the fallback target.

The deployment specification *requires* the last rule in the traversal order to be a catch-all rule. A catch-all rule is specified with the `/*` path. If the incoming request doesn't match with any of the previous routes in the routing rules array, the request is routed to the catch-all rule target.

For SSR frameworks like Nuxt.js, the catch-all rule target has to be the compute primitive. This is because SSR applications have server-side rendered pages with routes that aren't predictable at build time. For example, if a Nuxt.js application has a page at `/blog/[slug]` where `[slug]` is a dynamic route parameter. The catch-all rule target is the only way to route requests to these pages.

In contrast, specific path patterns can be used to target routes that are known at build time. For example, Nuxt.js serves static assets from the `/_nuxt` path. This means that the `/_nuxt/*` path can be targeted by a specific routing rule that routes requests to the static assets primitive.

Public folder routing

Most SSR frameworks provide the ability to serve mutable static assets from a `public` folder. Files like `favicon.ico` and `robots.txt` are typically kept inside the `public` folder and are served from the application's root URL. For example, the `favicon.ico` file is served from `https://example.com/favicon.ico`. Note that there is no predictable path pattern for these files. They are almost entirely dictated by the file name. The only way to target files inside the `public` folder is to use the catch-all route. However, the catch-all route target has to be the compute primitive.

We recommend one of the following approaches for managing your `public` folder.

1. Use a path pattern to target request paths that contain file extensions. For example, you can use `/*.*` to target all request paths that contain a file extension.

   Note that this approach can be unreliable. For example, if there are files without file extensions inside the `public` folder, they are not targeted by this rule. Another issue to be aware of with this approach is that the application could have pages with periods in their names. For example, a page at `/blog/2021/01/01/hello.world` will be targeted by the `/*.*` rule. This is not ideal since the page is not a static asset. However, you can add a fallback target to this rule to ensure that when there is a 404 error from the static primitive, the request falls back to the compute primitive.

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

2. Identify the files in the `public` folder at build time and emit a routing rule for each file. This approach is not scalable since there is a limit of 25 rules imposed by the deployment specification.

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

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

3. Recommend that your framework users store all mutable static assets inside a sub-folder inside the `public` folder.

   In the following example, the user can store all mutable static assets inside the `public/assets` folder. Then, a routing rule with the path pattern `/assets/*` can be used to target all mutable static assets inside the `public/assets` folder.

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

4. Specify a static fallback for the catch-all route. This approach has drawbacks that are described in more detail in the next Catch-all fallback routing section.

Catch-all fallback routing

For SSR frameworks such as Nuxt.js, where a catch-all route is specified for the compute primitive target, framework authors might consider specifying a static fallback for the catch-all route to solve the `public` folder routing problem. However, this type of routing rule breaks server-side rendered 404 pages. For example, if the end user visits a page that doesn't exist, the application renders a 404 page with a status code of 404. However, if the catch-all route has a static fallback, the 404 page isn't be rendered. Instead, the request falls back to the static primitive and still ends up with a 404 status code, but the 404 page isn't be rendered.

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

```
    },
    "fallback": {
        "kind": "Static"
    }
}
```

Base path routing

Frameworks that offer the ability to modify the base path of the application are expected to prepend the base path to the static assets inside the `.amplify-hosting/static` directory. For example, if the base path is `/folder1/folder2`, then the build output for a static asset called main.css will be `.amplify-hosting/static/folder1/folder2/main.css`.

This means that the routing rules also need to be updated to reflect the base path. For example, if the base path is `/folder1/folder2`, then the routing rule for the static assets in the `public` folder will look like the following.

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

Similarly, server-side routes also need to have the base path prepended to them. For example, if the base path is `/folder1/folder2`, then the routing rule for the `/api` route will look like the following.

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

However, the base path should not be prepended to the catch-all route. For example, if the base path is `/folder1/folder2`, then the catch-all route will remain like the following.

```
{
    "path": "/*",
```

```
    "target": {
        "kind": "Compute",
        "src": "default"
    }
}
```

Nuxt.js routes examples

The following is an example `deploy-manifest.json` file for a Nuxt application that demonstrates how to specify routing rules.

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

```
      "path": "/*.*",
      "target": {
        "kind": "Static"
      },
      "fallback": {
        "kind": "Compute",
        "src": "default"
      }
    },
    {
      "path": "/*",
      "target": {
        "kind": "Compute",
        "src": "default"
      }
    }
  ],
  "computeResources": [
    {
      "name": "default",
      "entrypoint": "server.js",
      "runtime": "nodejs18.x"
    }
  ],
  "framework": {
    "name": "nuxt",
    "version": "3.8.1"
  }
}
```

The following is an example `deploy-manifest.json` file for Nuxt that demonstrates how to specify routing rules including base paths.

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

```
    {
      "path": "/base-path/_nuxt/builds/meta/*",
      "target": {
        "cacheControl": "public, max-age=31536000, immutable",
        "kind": "Static"
      }
    },
    {
      "path": "/base-path/_nuxt/builds/*",
      "target": {
        "cacheControl": "public, max-age=1, immutable",
        "kind": "Static"
      }
    },
    {
      "path": "/base-path/_nuxt/*",
      "target": {
        "cacheControl": "public, max-age=31536000, immutable",
        "kind": "Static"
      }
    },
    {
      "path": "/base-path/*.*",
      "target": {
        "kind": "Static"
      },
      "fallback": {
        "kind": "Compute",
        "src": "default"
      }
    },
    {
      "path": "/*",
      "target": {
        "kind": "Compute",
        "src": "default"
      }
    }
  ],
  "computeResources": [
    {
      "name": "default",
      "entrypoint": "server.js",
      "runtime": "nodejs18.x"
```

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

For more information about using the `routes` attribute, see Using the routes attribute.

Deploying an Express server using the deployment manifest

This example explains how to deploy a basic Express server using the Amplify Hosting deployment specification. You can leverage the provided deployment manifest to specify routing, compute resources, and other configurations.

Set up an Express server locally before deploying to Amplify Hosting

1. Create a new directory for your project and install Express and Typescript.

```
mkdir express-app
cd express-app

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

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

2. Add a `tsconfig.json` file to the root of your project with the following contents.

```
{
  "compilerOptions": {
    "target": "es6",
    "module": "commonjs",
    "outDir": "./dist",
    "strict": true,
    "esModuleInterop": true,
    "skipLibCheck": true,
    "forceConsistentCasingInFileNames": true
```

```
  },
  "include": ["src/**/*.ts"],
  "exclude": ["node_modules"]
}
```

3. Create a directory named `src` in your project root.
4. Create an `index.ts` file in the `src` directory. This will be the entry point to the application that starts an Express server. The server should be configured to listen on port 3000.

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

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

app.use(express.text());

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

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

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

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

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

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

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

5. Add the following scripts to your `package.json` file.

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

6. Create a directory named `public` in the root of your project. Then create a file named `hello-world.txt` with the following contents.

```
Hello world!
```

7. Add a `.gitignore` file to your project root with the following contents.

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

Set up the Amplify deployment manifest

1. Create a file named `deploy-manifest.json` in your project's root directory.
2. Copy and paste the following manifest into your `deploy-manifest.json` file.

```
{
  "version": 1,
  "framework": { "name": "express", "version": "4.18.2" },
  "imageSettings": {
    "sizes": [
```

```
      100,
      200,
      1920
    ],
    "domains": [],
    "remotePatterns": [],
    "formats": [],
    "minimumCacheTTL": 60,
    "dangerouslyAllowSVG": false
  },
  "routes": [
    {
      "path": "/_amplify/image",
      "target": {
        "kind": "ImageOptimization",
        "cacheControl": "public, max-age=3600, immutable"
      }
    },
    {
      "path": "/*.*",
      "target": {
        "kind": "Static",
        "cacheControl": "public, max-age=2"
      },
      "fallback": {
        "kind": "Compute",
        "src": "default"
      }
    },
    {
      "path": "/*",
      "target": {
        "kind": "Compute",
        "src": "default"
      }
    }
  ],
  "computeResources": [
    {
      "name": "default",
      "runtime": "nodejs18.x",
      "entrypoint": "index.js"
    }
  ]
```

```
}
```

The manifest describes how Amplify Hosting should handle the deployment of your application. The primary settings are the following.

- **version** – Indicates the version of the deployment specification that you're using.
- **framework** – Adjust this to specify your Express server setup.
- **imageSettings** – This section is optional for an Express server unless you're handling image optimization.
- **routes** – These are critical for directing traffic to the right parts of your app. The `"kind": "Compute"` route directs traffic to your server logic.
- **computeResources** – Use this section to specify your Express server's runtime and entry point.

Next, set up a post-build script that moves the built application artifacts into the `.amplify-hosting` deployment bundle. The directory structure aligns with the Amplify Hosting deployment specification.

Set up the post-build script

1. Create a directory named `bin` in your project root.
2. Create a file named `postbuild.sh` in the `bin` directory. Add the following contents to the `postbuild.sh` file.

```
#!/bin/bash

rm -rf ./.amplify-hosting

mkdir -p ./.amplify-hosting/compute

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

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

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

3. Add a `postbuild` script to your `package.json` file. The file should look like the following.

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

4. Run the following command to build your application.

```
npm run build
```

5. (Optional) Adjust your routes for Express. You can modify the routes in your deployment manifest to fit your Express server. For example, if you don't have any static assets in the `public` directory, you might only need the catch-all route `"path": "/*"` directing to Compute. This will depend on your server's setup.

Your final directory structure should look like the following.

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

```
### package.json
### package-lock.json
### tsconfig.json
```

Deploy your server

1. Push your code to your Git repository and then deploy your app to Amplify Hosting.
2. Update your build settings to point `baseDirectory` to `.amplify-hosting` as follows. During the build, Amplify will detect the manifest file in the `.amplify-hosting` directory and deploy your Express server as configured.

   ```
   version: 1
   frontend:
     phases:
       preBuild:
         commands:
           - nvm use 18
           - npm install
       build:
         commands:
           - npm run build
     artifacts:
       baseDirectory: .amplify-hosting
       files:
         - '**/*'
   ```

3. To verify that your deployment was successful and that your server is running correctly, visit your app at the default URL provided by Amplify Hosting.

Image optimization integration for framework authors

Framework authors can integrate Amplify's image optimization feature by using the Amplify Hosting deployment specification. To enable image optimization, your deployment manifest must contain a routing rule that targets the image optimization service. The following example demonstrates how to configure the routing rule.

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

{
  "routes": [
```

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

For more information about configuring image optimization settings using the deployment specification, see Using the Amplify Hosting deployment specification to configure build output .

Understanding the Image optimization API

Image optimization can be invoked at runtime via an Amplify app's domain URL, at the path defined by the routing rule.

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

Image optimization imposes the following rules on images.

- Amplify can't optimize GIF, APNG and SVG formats or convert them to another format.
- SVG images aren't served unless the `dangerouslyAllowSVG` setting is enabled.
- The width or height of a source images can't exceed 11 MB or 9,000 pixels.
- The size limit of an optimized image is 4 MB.
- HTTPS is the only protocol supported for sourcing images with remote URLs.

HTTP headers

The **Accept** request HTTP header is used to specify the image formats, expressed as MIME types, allowed by the client (usually a web browser). The image optimization service will attempt to convert the image to the specified format. The value specified for this header will have a higher priority than the format query parameter. For example, a valid value for the **Accept** header is `image/png, image/webp, */*` . The formats setting specified in the Amplify deployment manifest will restrict the formats to those in the list. Even if the **Accept** header asks for a specific format, it will be ignored if the format isn't in the allow list.

URI request parameters

The following table describes the URI request parameters for Image optimization.

Query parameter	Type	Required	Description	Example
url	String	Yes	A relative path or absolute URL to the source image. For a remote URL, only the https protocol is supported. Value must be URL encoded.	`?url=https%3A%2F%2Fwww.example.com%2Fbuffalo.png`
width	Number	Yes	The width in pixels of the optimized image.	`?width=800`
height	Number	No	The height in pixels of the optimized image. If not specified, the image will be auto scaled to match the width.	`?height=600`
fit	Enum values: `cover`, `contain`, `fill`, `inside`, `outside`	No	How the image is resized to fit the specified width and height.	`?width=800&height=600&fit=cover`

Query parameter	Type	Required	Description	Example
position	Enum values: `center`, `top`, `right`, `bottom`, `left`	No	A position to be used when fit is `cover` or `contain`.	`?fit=contain&position=centre`
trim	Number	No	Trims pixels from all edges that contain values similar to the specified background color of the top-left pixel.	`?trim=50`
extend	Object	No	Adds pixels to the edges of the image using the color derived from the nearest edge pixels. The format is `{top}_{right}_{bottom}_{left}` where each value is the number of pixels to add.	`?extend=10_0_5_0`

Query parameter	Type	Required	Description	Example
extract	Object	No	Crops the image to the specified rectangle delimited by top, left, width and height. The format is {left}_{top}_{width}_{right} where each value is the number of pixels to crop.	`?extract=10_0_5_0`
format	String	No	The desired output format for the optimized image.	`?format=webp`
quality	Number	No	The quality of the image, from 1 to 100. Only used when converting the format of the image.	`?quality=50`
rotate	Number	No	Rotates the image by the specified angle in number of degrees.	`?rotate=45`

Query parameter	Type	Required	Description	Example
flip	Boolean	No	Mirrors the image vertically (up-down) on the x-axis. This always occurs before rotation, if any.	`?flip`
flop	Boolean	No	Mirrors the image horizontally (left-right) on the y-axis. This always occurs before rotation, if any.	`?flop`
sharpen	Number	No	Sharpening enhances the definition of edges in the image. Valid values are between 0.000001 and 10.	`?sharpen=1`
median	Number	No	Applies a median filter. This removes noise or smoothes the edges of an image.	`?sharpen=3`

Query parameter	Type	Required	Description	Example
blur	Number	No	Applies a Gaussian blur of the specified sigma. Valid values are from 0.3 to 1,000.	`?blur=20`
gamma	Number	No	Applies a gamma correctio n to improve the perceived brightness of a resized image. Value must be between 1.0 and 3.0.	`?gamma=1`
negate	Boolean	No	Inverts the colors of the image.	`?negate`
normalize	Boolean	No	Enhances image contrast by stretching its luminance to cover a full dynamic range.	`?normalize`

Query parameter	Type	Required	Description	Example
threshold	Number	No	Replaces any pixel in the image with a black pixel, if its intensity is less than the specified threshold. Or with a white pixel if it's greater than the threshold. Valid values are between 0 and 255.	`?threshold=155`
tint	String	No	Tints the image using the provided RGB while preserving the image luminance.	`?tint=#7743CE`
grayscale	Boolean	No	Turns the image into grayscale (black and white).	`?grayscale`

Response status codes

The following list describes the response status codes for image optimization.

Success - HTTP status code 200

The request was fullfilled successfully.

BadRequest - HTTP status code 400

- An input query parameter was specified incorrectly.
- The remote URL is not listed as allowed in the `remotePatterns` setting.
- The remote URL doesn't resolve to an image.
- The requested width or height are not listed as allowed in the `sizes` setting.
- The image requested is SVG but the `dangerouslyAllowSvg` setting is disabled.

Not Found - HTTP status code 404

The source image was not found.

Content too large - HTTP status code 413

Either the source image or the optimized image exceed the maximum allowed size in bytes.

Understanding optimized image caching

Amplify Hosting caches optimized images on our CDN so that subsequent requests to the same image, with the same query parameters, are served from the cache. The cache Time to live (TTL) is controlled by the `Cache-Control` header. The following list describes your options for specifying the `Cache-Control` header.

- Using the `Cache-Control` key within the routing rule that targets image optimization.
- Using custom headers defined in the Amplify app.
- For remote images, the `Cache-Control` header returned by the remote image is honored.

The `minimumCacheTTL` specified in the image optimization settings defines the lower bound of the Cache-Control max-age directive. For example, if a remote image URL responds with a `Cache-Control s-max-age=10`, but the value of `minimumCacheTTL` is 60, then 60 is used.

Using open source adapters for any SSR framework

You can use any SSR framework build adapter that has been created for integration with Amplify Hosting. Each framework that offers an adapter determines how the adapter is configured and connected to their build process. Typically, you will install the adapter as an npm development dependency.

After you create an app with a framework, use the framework's documentation to learn how to install the Amplify Hosting adapter and configure it in your application's configuration file.

Next, create an `amplify.yml` file in your project's root directory. In the `amplify.yml` file, set the `baseDirectory` to your application's build output directory. The framework runs the adapter during the build process to transform the output into the Amplify Hosting deployment bundle.

The name of the build output directory can be anything, but the `.amplify-hosting` filename has significance. Amplify first looks for a directory defined as the `baseDirectory`. If it exists, Amplify looks for the build output there. If the directory doesn't exist, Amplify looks for the build output inside `.amplify-hosting`, even if it has not been defined by the customer.

The following is an example of the build settings for an app. The `baseDirectory` is set to `.amplify-hosting` to indicate that the build output is in the `.amplify-hosting` folder. As long as the contents of the `.amplify-hosting` folder match the Amplify Hosting deployment specification, the app will deploy successfully.

```
version: 1
frontend:
  preBuild:
    commands:
      - npm install
  build:
    commands:
      - npm run build
  artifacts:
    baseDirectory: .amplify-hosting
```

After your app is configured to use a framework adapter, you can deploy it to Amplify Hosting. For detailed instructions, see Deploying an SSR app to Amplify

Deploying a static website to Amplify from an Amazon S3 bucket

You can use the integration between Amplify Hosting and Amazon S3 to host static website content stored on S3 with just a few clicks. Deploying to Amplify Hosting provides you with the following benefits and features.

- Automatic deployment to the globally available AWS content delivery network (CDN) powered by CloudFront
- HTTPS support
- Easily connect your website to a custom domain using the Amplify console
- Bring your own Custom SSL certificates
- Monitor your website with built in access logs and CloudWatch metrics
- Set up password protection for your website
- Create redirect and rewrites rules in the Amplify console

You can start the deployment process from the Amplify console, the AWS CLI, or the AWS SDKs. You can only deploy to Amplify from an Amazon S3 general purpose bucket located in your own account. Amplify doesn't support cross-account S3 bucket access.

When you deploy your application from an Amazon S3 general purpose bucket to Amplify Hosting, AWS charges are based on Amplify's pricing model. For more information, see AWS Amplify Pricing.

Important

Amplify Hosting is not available in all of the AWS Regions where Amazon S3 is available. To deploy a static website to Amplify Hosting, the Amazon S3 general purpose bucket containing your website must be located in a region where Amplify is available. For the list of regions where Amplify is available, see Amplify endpoints in the *Amazon Web Services General Reference*.

See the following topics to learn how to deploy and update a static website from Amazon S3 to Amplify Hosting.

Topics

- Deploying a static website from S3 using the Amplify console
- Creating a bucket policy to deploy a static website from S3 using the AWS SDKs
- Updating a static website deployed to Amplify from an S3 bucket
- Updating an S3 deployment to use a bucket and prefix instead of a .zip file

Deploying a static website from S3 using the Amplify console

Use the following instructions to deploy a new static website from an Amazon S3 general purpose bucket using the Amplify console.

To deploy a static website from an Amazon S3 general purpose bucket using the Amplify console

1. Sign in to the AWS Management Console and open the Amplify console at https://console.aws.amazon.com/amplify/.
2. On the **All apps** page, choose **Create new app**.
3. On the **Start building with Amplify** page, choose **Deploy without Git**.
4. Choose **Next**.
5. On the **Start a manual deployment** page, do the following.
    a. For **App name**, enter the name of your app.
    b. For **Branch name**, enter the name of the branch to deploy.
6. For **Method**, choose **Amazon S3**.
7. For the **S3 location of objects to host**, choose **Browse**. Select the Amazon S3 general purpose bucket to use, then select **Choose prefix**.
8. Choose **Save and deploy**.

Creating a bucket policy to deploy a static website from S3 using the AWS SDKs

You can use the AWS SDKs to deploy a static website from Amazon S3 to Amplify Hosting. If you deploy your website using an SDK, you must create your own bucket policy that grants Amplify Hosting permission to retrieve the objects in your S3 bucket.

To learn more about creating bucket policies, see Bucket policies for Amazon S3 in the *Amazon Simple Storage Service User Guide*.

The following example bucket policy grants Amplify Hosting permissions to list buckets and retrieve bucket objects for the specified AWS account, Amplify application id, and branch.

To use this example:

- Replace *amzn-s3-demo-website-bucket/prefix* with the name of your website's bucket and prefix.
- Replace *111122223333* with your AWS account id.
- Replace *region-id* with the AWS Region that your Amplify application is located in, such as **us-east-1**.
- Replace *app_id* with you Amplify application id. This information is available in the Amplify console.
- Replace *branch_name* with your branch name.

> **Note**
>
> In your bucket policy, the `aws:SourceArn` must be a URL-encoded (percent-encoding) branch ARN.

```
{
    "Version": "2008-10-17",
    "Statement": [
        {
            "Sid": "AllowAmplifyToListPrefix_appid_branch_prefix_",
            "Effect": "Allow",
            "Principal": {
                "Service": "amplify.amazonaws.com"
            },
            "Action": "s3:ListBucket",
            "Resource": "arn:aws:s3:::amzn-s3-demo-website-bucket/prefix/*",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333",
                    "aws:SourceArn": "arn%3Aaws%3Aamplify%3Aregion-
id%3A111122223333%3Aapps%2Fapp_id%2Fbranches%2Fbranch_name",
```

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

Updating a static website deployed to Amplify from an S3 bucket

If you update any of the objects for a static website in general purpose S3 bucket hosted on Amplify, you must redeploy the application to Amplify Hosting to cause the changes to take effect. Amplify Hosting doesn't automatically detect changes to the S3 bucket. We recommend that you use the AWS Command Line Interface (CLI) to update your website.

Sync updates to S3

After you make changes to your website's project files, use the following s3 sync command to synchronize the changes that you made to your local source directory with your target Amazon S3 general purpose bucket. To use this example, replace *<source>* with the name of your local directory and *<target>* with the name of your Amazon S3 bucket.

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

Redeploy the website to Amplify Hosting

Use the following amplify start-deployment command to redeploy your updated application in an Amazon S3 bucket to Amplify Hosting. To use this example, replace *<app_id>* with the id of your Amplify application, *<branch_name>* with the name of your branch, and *s3://amzn-s3-demo-website-bucket/prefix* with your S3 bucket and prefix. .

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

Updating an S3 deployment to use a bucket and prefix instead of a .zip file

If you already have an existing static website deployed to Amplify Hosting from a .zip file in an Amazon S3 general purpose bucket, you can update the application deployment to use the bucket name and prefix that contain the objects to host. This type of deployment eliminates the need to upload a separate file to your bucket that contains the zipped contents of the build output.

To migrate a static website from a .zip file to the bucket contents

1. Sign in to the AWS Management Console and open the Amplify console at https://console.aws.amazon.com/amplify/.
2. On the **All apps** page, choose the name of the manually deployed app that you want to migrate from using a .zip file to using the application files directly.
3. On the application's **Overview** page, choose **Deploy updates**.
4. On the **Deploy updates** page, for **Method**, choose **Amazon S3**.

5. For the **S3 location of objects to host**, choose **Browse**. Select the bucket to use, then select **Choose prefix**.
6. Choose **Save and deploy**.

Deploying an application to Amplify without a Git repository

Manual deployments enable you to publish your web app with Amplify Hosting without connecting a Git provider. You can drag and drop a zipped folder from your desktop and host your site in seconds. Alternatively, you can reference assets in an Amazon S3 bucket or specify a public URL to the location where your files are stored.

> **Note**
>
> Manual deployments have a maximum .zip file size limit of 5GB due to Amazon S3 copy operation constraints. If any of your build artifacts exceed this size, consider breaking them into smaller archives or using an alternative deployment method.

For Amazon S3, you can also set up AWS Lambda triggers to update your site each time new assets are uploaded. See the Deploy files stored on Amazon S3, Dropbox, or your Desktop to the AWS Amplify console blog post for more details about setting up this scenario.

Amplify Hosting does not support manual deploys for server-side rendered (SSR) apps. For more information, see Deploying server-side rendered applications with Amplify Hosting.

Drag and drop manual deployments

To manually deploy an app using drag and drop

1. Sign in to the AWS Management Console and open the Amplify console.
2. In the upper right corner, choose **Create new app**.
3. On the **Start building with Amplify** page, choose **Deploy without Git**. Then, choose **Next**.
4. On the **Start a manual deployment** page, for **App name**, enter the name of your app.
5. For **Branch name**, enter a meaningful name, such as **`development`** or **`production`**.
6. For **Method**, choose **Drag and drop**.
7. Either drag and drop a folder from your desktop onto the drop zone or use **Choose .zip folder** to select the file from your computer. The file that you drag and drop or select must be a a zipped folder that contains the contents of your build output.

8. Choose **Save and deploy**.

Amazon S3 or URL manual deployment

> **Note**
>
> If you are deploying a static website from S3, the following procedure requires that you upload a zipped folder with the contents of your build output to your S3 bucket. We recommend that you deploy a static website directly from S3 using the bucket name and prefix. For more information about this simplified process, see Deploying a static website to Amplify from an Amazon S3 bucket.

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. In the upper right corner, choose **Create new app**.
3. On the **Start building with Amplify** page, choose **Deploy without Git**. Then, choose **Next**.
4. On the **Start a manual deployment** page, for **App name**, enter the name of your app.
5. For **Branch name**, enter a meaningful name, such as **`development`** or **`production`**.
6. For **Method**, choose either **Amazon S3** or **Any URL**.
7. The procedure for uploading your files depends on the upload method.
   - Amazon S3
     a. For **S3 location of objects to host**, choose **Browse S3**. Then, select the name of the Amazon S3 bucket from the list. Access control lists (ACLs) must be enabled for the bucket you select. For more information, see Troubleshooting Amazon S3 bucket access for manual deployments.
     b. Select the name of the .zip file to deploy.
     c. Choose **Choose prefix**.
   - Any URL
     - For **Resource URL**, enter the URL to the .zip file to deploy.
8. Choose **Save and deploy**.

> **Note**
>
> When you create the zipped 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.

Troubleshooting Amazon S3 bucket access for manual deployments

When you create an Amazon S3 bucket, you use its Amazon S3 Object Ownership setting to control whether access control lists (ACLs) are enabled or disabled for the bucket. To manually deploy an app to Amplify from an Amazon S3 bucket, ACLs must be enabled on the bucket.

If you get an `AccessControlList` error when you deploy from an Amazon S3 bucket, the bucket was created with ACLs disabled and you must enable them in the Amazon S3 console. For instructions, see Setting Object Ownership on an existing bucket in the *Amazon Simple Storage Service User Guide*.

Managing the build configuration for an Amplify application

You can customize the build settings and configuration for your Amplify deployments. When you deploy an application, Amplify automatically detects the frontend framework and the associated build settings. You can customize the build settings in the application's build specification (buildspec) to add environment variables, run build commands, and specify build dependencies.

Amplify's default build image comes with several packages and dependencies pre-installed, but you can also use the live package updates feature to specify either a specific version, or ensure that the latest version is always installed. If you have specific dependencies that take a long time to install during a build using Amplify's default container, you can create your own custom build image. You can also customize the build instance size to provide your application deployment with the CPU, memory, and disk space resources it needs.

Builds are initiated automatically with each commit to your Git repository and with each new deployment. You can set up the incoming webhooks feature to initiate a build without a commit to your Git repository.

The build notifications feature allows you to share information with team members about build successes and failures.

Topics

- Configuring the build settings for an Amplify application
- Customizing the build image
- Configuring the build instance for an Amplify application
- Creating an incoming webhook to start a build
- Setting up email notifications for builds

Configuring the build settings for an Amplify application

When you deploy an application, Amplify automatically detects the frontend framework and associated build settings by inspecting the app's `package.json` file in your Git 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 by 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.

> **Note**
>
> **Build settings** is visible in the Amplify console's **Hosting** 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.

Build specification reference

The build specification (buildspec) for an Amplify application is a collection of YAML settings and build commands that Amplify uses to run your build. The following list describes these settings and how they are used.

version

The Amplify 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.

frontend

Run frontend build commands.

test

Run commands during a test phase. Learn how to add tests to your app.

build phases

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 Amplify installs dependencies.
- **build** - Your build commands.
- **postBuild** - The post-build script runs after the build has finished and Amplify has copied all the necessary artifacts to the output directory.

buildpath

The path to use to run the build. Amplify uses this path to locate your build artifacts. If you don't specify a path, Amplify uses the monorepo app root, for example `apps/app`.

artifacts>base-directory

The directory in which your build artifacts exist.

artifacts>files

Specify files from your artifacts you want to deploy. Enter `**/*` to include all files.

cache

Specifies build-time dependencies such as the *node_modules* folder. During the first build, paths provided here are cached. On subsequent builds, Amplify restores the cache to the same paths before it runs your commands.

Amplify considers all provided cache paths to be relative to your project root. However, Amplify doesn't allow traversing outside of the project root. For example, if you specify an absolute path, the build will succeed without an error, but the path won't be cached.

Build specification YAML syntax reference

The following example of a build specification demonstrates the basic YAML syntax.

```
version: 1
env:
  variables:
```

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

```
    files:
        - location
        - location
    configFilePath: *location*
    baseDirectory: *location*
```

Editing the build specification

You can customize an application's build settings by editing the build specification (buildspec) in the Amplify console. 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 Git repository.

To edit build settings 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 edit the build settings for.
3. In the navigation pane, choose **Hosting**, then choose **Build settings**.
4. On the **Build settings** page, in the **App build specification** section, choose **Edit**.
5. In the **Edit build spec** window, enter your updates.
6. Choose **Save**.

You can use the examples described in the following topics to update your build settings for specific scenarios.

Topics

- Setting branch-specific build settings with scripting
- Setting a command to navigate to a subfolder
- Deploying the backend with the front end for a Gen 1 app
- Setting the output folder
- Installing packages as part of a build
- Using a private npm registry
- Installing OS packages
- Setting key-value storage for every build
- Skipping the build for a commit
- Turning off automatic builds on every commit

- Configuring diff based frontend build and deploy
- Configuring diff based backend builds for a Gen 1 app

Setting branch-specific build settings with scripting

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 run 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
```

Setting a command to navigate 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 for a Gen 1 app

> **Note**
>
> This section applies to Amplify Gen 1 applications only. A Gen 1 backend is created using Amplify Studio and the Amplify command line interface (CLI).

The `amplifyPush` command 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 the `npm` or `yarn` commands to install packages during the build.

```
frontend:
```

```
  phases:
    build:
      commands:
        - npm install -g <package>
        - <package> 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

Amplify's AL2023 image runs your code with a non-privileged user named `amplify`. Amplify grants this user privileges to run OS commands using the Linux `sudo` command. If you want to install OS packages for missing dependencies, you can use commands such as `yum` and `rpm` with `sudo`.

The following example build section demonstrates the syntax for installing an OS package using the `sudo` command.

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

Setting 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>
```

Skipping the 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.

Turning off automatic builds on every commit

You can configure Amplify to turn off automatic builds on every code commit. To set up, choose **App settings**, **Branch settings**, and then locate the **Branches** section that lists the connected branches. Select a branch, and then choose **Actions**, **Disable auto build**. New commits to that branch will no longer start a new build.

Configuring diff based frontend build and deploy

You can configure Amplify to use diff based frontend builds. If enabled, at the start of each build Amplify 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 **Hosting**, **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.
6. Choose **Save**.

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`.

Configuring diff based backend builds for a Gen 1 app

> **Note**
>
> This section applies to Amplify Gen 1 applications only. A Gen 1 backend is created using Amplify Studio and the Amplify command line interface (CLI).

You can configure Amplify Hosting 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.

If you currently have custom commands specified in the build settings of your backend phase, conditional backend builds won't work. If you want those custom commands to run, you must move them to the frontend phase of your build settings in your app's `amplify.yml` file.

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 **Hosting**, **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.
6. Choose **Save**.

Configuring monorepo build settings

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

Amplify supports apps in generic monorepos as well as apps in monorepos created using npm workspace, pnpm workspace, Yarn workspace, Nx, and Turborepo. When you deploy your app, Amplify automatically detects the monorepo build tool that you are using. Amplify automatically applies build settings for apps in an npm workspace, Yarn workspace or Nx. Turborepo and pnpm apps require additional configuration. For more information, see Configuring Turborepo and pnpm monorepo apps.

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 reference

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 `appRoot` key 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.

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.

The `buildpath` attribute is set to `/` to run and build the app from the monorepo project root. The `baseDirectory` attribute is the relative path of `buildpath`.

Monorepo build specification YAML syntax

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

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

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

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

An app using the following example build specification, will be built under the project root and the build artifacts will be located at `/packages/nextjs-app/.next`.

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

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 the Amplify console

1. Sign in to the AWS Management Console and open the Amplify console.
2. Choose **Create new app** in the upper right corner.
3. On the **Start building with Amplify** page, choose your Git provider, then choose **Next**.
4. On the **Add repository branch** page, do the following:
    a. Choose the name of your repository from the list.
    b. Choose the name of the branch to use.
    c. Select **My app is a monorepo**
    d. Enter the path to your app in your monorepo, for example, **`apps/app1`**.
    e. Choose **Next**.
5. On the **App settings** page, you can use the default settings or customize the build settings for your app. In the **Environment variables** section, Amplify sets `AMPLIFY_MONOREPO_APP_ROOT` to 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 **Hosting**, 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 new**.
   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**.

Configuring Turborepo and pnpm monorepo apps

The Turborepo and pnpm workspace monorepo build tools get configuration information from `.npmrc` files. When you deploy a monorepo app created with one of these tools, you must have an `.npmrc` file in your project root directory.

In the `.npmrc` file, set the linker for installing Node packages to `hoisted`. You can copy the following line to your file.

```
node-linker=hoisted
```

For more information about `.npmrc` files and settings, see pnpm .npmrc in the *pnpm documentation*.

Pnpm is not included in the Amplify default build container. For pnpm workspace and Turborepo apps, you must add a command to install pnpm in the `preBuild` phase of your app's build settings.

The following example excerpt from a build specification shows a `preBuild` phase with a command to install pnpm.

```
version: 1
applications:
  - frontend:
      phases:
        preBuild:
          commands:
            - npm install -g pnpm
```

Customizing the build image

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's default container, you can create your own Docker image and reference it during a build. Images can be hosted on Amazon Elastic Container Registry Public.

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

Custom build image requirements

1. A Linux distribution that supports the GNU C Library (glibc), such as Amazon Linux, compiled for the x86-64 architecture.
2. **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 instantly fails without any output as our build-runner is unable to produce any output.
3. **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.
4. **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.
5. **Bash and The Bourne Shell**: These two utilities are used to run commands at build time. If they aren't installed, your builds might fail prior to starting.
6. **Node.JS+NPM**: Our build runner doesn't install Node. Instead, it relies on Node and NPM being installed in the image. This is only required for builds that require NPM packages or Node

specific commands. However, we strongly recommend installing them because when they are present, the Amplify build runner can use these tools to improve the build execution. Amplify's package override feature uses NPM to install the Hugo-extended package when you set an override for Hugo.

The following packages aren't required, but we strongly recommend that you install them.

1. **NVM (Node Version Manager)**: We recommend that you install this version manager if you need to handle different versions of Node. When you set an override, Amplify's package override feature uses NVM to change Node.js versions before each build.
2. **Wget**: Amplify can use the Wget utility to download files during the build process. We recommend that you install it in your custom image.
3. **Tar**: Amplify can use the Tar utility to uncompress downloaded files during the build process. We recommend that you install it in your custom image.

Configuring a custom build image for an app

Use the following procedure to configure a custom build image for an application in the Amplify console.

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 **Hosting**, **Build settings**.
5. On the **Build settings** page, in the **Build image settings** section, choose **Edit**.
6. On the **Edit build image settings** page, expand the **Build image** menu, and choose **Custom 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**.

Using specific package and dependency versions in the build image

Live package updates enable you to specify the versions of packages and dependencies to use in the Amplify 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.

> **Warning**
>
> Setting the Node.js version to **latest** causes builds to fail. Instead, you must specify an exact Node.js version, such as `18`, `21.5`, or `v0.1.2`.

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 **Hosting**, **Build settings**.
4. On the **Build settings** page, in the **Build image settings** section, choose **Edit**.
5. On the **Edit build image settings** page, **Live package updates** list, choose **Add new**.
6. For **Package**, select the dependency to override.
7. 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.
8. Choose **Save**.

Configuring the build instance for an Amplify application

Amplify Hosting offers configurable build instance sizes that enable you to provide your application's build instance with the CPU, memory, and disk space resources that it requires. Prior

to the release of this feature, Amplify provided a fixed size build instance configuration of 8 GiB of memory and 4 vCPUs.

Amplify supports three build instance types: Standard, Large, and XLarge. If you don't specify an instance type, Amplify uses the default Standard instance. You can configure the build instance type for an application using the Amplify console, the AWS CLI, or the SDKs.

The cost for each build instance type is calculated per build minute. For pricing details, see AWS Amplify Pricing.

The following table describes the compute specifications for each build instance type:

Build instance type	vCPUs	Memory	Disk space
Standard	4 vCPUs	8 GiB	128 GB
Large	8 vCPUs	16 GiB	128 GB
XLarge	36 vCPUs	72 GiB	256 GB

Topics

- Understanding build instance types
- Configuring the build instance type in the Amplify console
- Configuring an application's heap memory to utilize large instance types

Understanding build instance types

The build instance type setting is configured at the application level and extends to all of the application's branches. The following key details apply to build instance types:

- The build instance type that you configure for an application automatically applies to auto-created branches and pull request previews.
- The *Concurrent jobs* service quota applies across all build instance types in your AWS account. For example, if your *Concurrent jobs* limit is five, you can run up to a maximum of 5 builds across all instance types in your AWS account.

- The cost for each build instance type is calculated per build minute. The build instance allocation process can require additional overhead time before your build starts. For larger instances, especially XLarge, your build might experience latency before the build starts, due to this overhead time. However, you are billed only for the actual build time, not the overhead time.

You can configure the build instance type when you create a new application or you can update the instance type on an existing application. For instructions on configuring this setting in the Amplify console, see Configuring the build instance type in the Amplify console. You can also update this setting using the SDKs. For more information, see the CreateApp, and UpdateApp APIs in the *Amplify API Reference*.

If you have existing applications in your account that were created before the release of the customizable build instance type feature, they are using the default Standard instance type. When you update the build instance type for an existing application, any builds that are queued or in progress before your update will utilize the previously configured build instance type. For example, if you have an existing application with the `main` branch deployed to Amplify and you update its build instance type from **Standard** to **Large**, all new builds that you initiate from the `main` branch will use the **Large** build instance type. However, any builds that are in progress at the time that you update the build instance type will continue to run on the **Standard** instance.

Configuring the build instance type in the Amplify console

Use the following procedure to configure the build instance type when you create a new Amplify application.

To configure the build instance type for a new application

1. Sign in to the AWS Management Console and open the Amplify console.
2. On the **All apps** page, choose **Create new app**.
3. On the **Start building with Amplify** page, choose your Git repository provider, then choose **Next**.
4. On the **Add repository branch** page, do the following:
   a. In the **Recently updated repositories** list, select the name of the repository to connect.
   b. In the **Branch** list, select the name of the repository branch to connect.
   c. Choose **Next**.
5. On the **App settings** page, open the **Advanced settings** section.

6. For **Build instance type**, choose your desired instance type from the list.
7. If you are deploying a Node.js runtime based application, configure the heap memory size to effectively utilize a large instance type. You can do this on the **App settings** page by either setting an environment variable or updating the build settings. For more information, see Configuring an application's heap memory to utilize large instance types .
   - Set an environment variable
     a. In the **Advanced settings**, **Environment variables** section, choose **Add new**.
     b. For **Key** enter **`NODE_OPTIONS`**.
     c. For **Value**, enter `--max-old-space-size=`*`memory_size_in_mb`*. Replace *`memory_size_in_mb`* with your desired heap memory size in megabytes.
   - Update the build settings
     a. In the **Build settings** section, choose **Edit YML file**.
     b. Add the following command to the `preBuild` phase. Replace *`memory_size_in_mb`* with your desired heap memory size in megabytes.

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

     c. Choose **Save**.
8. Choose, **Next**.
9. On the **Review** page, choose **Save and deploy**.

Use the following procedure to configure the build instance type for an existing Amplify application.

To configure the build instance type for an existing application

1. Sign in to the AWS Management Console and open the Amplify console.
2. Choose the app that you want to configure the build instance type for.
3. In the navigation pane, choose **Hosting**, the choose **Build settings**.
4. On the **Build settings** page, in the **Advanced settings** section, choose **Edit**.
5. On the **Edit settings** page, for **Build instance type**, choose your desired instance type from the list.
6. Choose **Save**. This change will take effect the next time you deploy the application.

7. (Optional) To deploy the updated application immediately, do the following:

    a. In the navigation pane, choose **Overview**.

    b. On your application's overview page, choose the branch to redeploy.

    c. On the **Deployment** page, choose a deployment, such as the most recent deployment. Then, choose **Redeploy this version**. A new deployment will begin.

    d. When the deployment completes, the application's build settings will show that the branch is using the updated build instance type.

Configuring an application's heap memory to utilize large instance types

If you are building memory intensive applications, use this section to understand how to configure your application to utilize large instance types. Programming languages and frameworks often rely on allocating dynamic memory, also known as heap memory, during runtime to manage application memory requirements. Heap memory is requested by the runtime environment and allocated by the host operating system. By default, runtime environments enforce a maximum heap size limit available to the application. This means that no additional memory will be available for the application beyond the heap size, even though the host operating system or container has a larger amount of memory available.

As an example, the JavaScript Node.js v8 runtime environment enforces a default heap size limit which depends on several factors, including the host memory size. As a result, Standard and Large build instances have a default Node.js heap size of 2096 MB and the XLarge instance has a default heap size of 4144 MB. Therefore, building an application with a 6000 MB memory requirement using the default Node.js heap size on any Amplify build instance type will result in a failed build due to an out-of-memory error.

To work around the Node.js default heap size memory limits, you can do one of the following:

- Set the `NODE_OPTIONS` environment variable in your Amplify application to the value `--max-old-space-size=`*`memory_size_in_mb`*. For `memory_size_in_mb`, specify your desired heap memory size in megabytes.

  For instructions, see Setting environment variables.
- Add the following command to the `preBuild` phase in your Amplify application's build specification.

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

You can update the build specification in the Amplify console or in your application's `amplify.yml` file in your project repository. For instructions, see Configuring the build settings for an Amplify application.

The following example Amplify build specification sets a Node.js heap memory size to 7000 MB for building a React frontend application:

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

To effectively utilize large instance types, it is important to have a sufficient heap memory size configured. Configuring a small heap size for a memory intensive application will likely result in a build failure. The application's build logs might not directly indicate an out-of-memory error as the application runtime can crash unexpectedly. Configuring a heap size as large as the host memory might result in the host operating system swapping or terminating other processes, and potentially disrupting your build process. As a reference, Node.js recommends setting a maximum heap size of 1536 MB on a machine with approximately 2000 MB of memory to leave some memory for other uses.

The optimal heap size depends on your application's needs and resource usage. If you encounter out-of-memory errors, start with a moderate heap size and then gradually increase it as needed. As a guideline, we recommend starting with 6000 MB for a Standard instance type, 12000 MB for a Large instance type, and 60000 MB for an XLarge instance type.

Creating an incoming webhook to start a build

Set up an incoming webhook in the Amplify console to start a build without committing code to your Git repository. You can use webhooks 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 **Hosting**, then **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 **Create webhook**.
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 initiate builds.
   - Run the curl command in a terminal window to start a new build.

Setting up email notifications for builds

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

Setting up 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 **Hosting**, **Build notifications**. On the **Build notifications** page, choose **Manage notifications**.
4. On the **Manage notifications** page, choose **Add new**.
5. Do one of the following:
   - 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**, choose *All Branches*.
6. Choose **Save**.

Connecting a custom domain

You can connect an app that you've deployed with Amplify Hosting to a custom domain. When you use Amplify to deploy your web app, Amplify hosts it for you on the default `amplifyapp.com` domain with a URL such as `https://branch-name.d1m7bkiki6tdw1.amplifyapp.com`. When you connect your app to a custom domain, users see that your app is hosted on a custom URL, such as `https://www.example.com`.

You can purchase a custom domain through an accredited domain registrar such as Amazon Route 53 or GoDaddy. Route 53 is Amazon's Domain Name System (DNS) web service. For more information about using Route 53, see What is Amazon Route 53. For a list of third-party accredited domain registrars, see the Accredited Registrar Directory at the ICANN website.

When you set up your custom domain, you can use the default managed certificate that Amplify provisions for you or you can use your own custom certificate. You can change the certificate in use for the domain at any time. For detailed information about managing certificates, see Using SSL/TLS certificates.

Before you proceed with setting up a custom domain, verify that you have met the following prerequisites.

- You own a registered domain name.
- You have a certificate issued by or imported into AWS Certificate Manager.
- You have deployed your app to Amplify Hosting.

  For more information about completing this step, see Getting started with deploying an app to Amplify Hosting.
- You have a basic knowledge of domains and DNS terminology.

  For more information about domains and DNS, see Understanding DNS terminology and concepts.

Topics

- Understanding DNS terminology and concepts
- Using SSL/TLS certificates
- Adding a custom domain managed by Amazon Route 53

- Adding a custom domain managed by a third-party DNS provider
- Updating DNS records for a domain managed by GoDaddy
- Updating the SSL/TLS certificate for a domain
- Managing subdomains
- Setting up wildcard subdomains
- Setting up automatic subdomains for an Amazon Route 53 custom domain
- Troubleshooting custom domains

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.

Amplify 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, you have to manually update your domain's DNS settings and add a new CNAME record provided by Amplify.

Custom domain activation process

When you connect your Amplify app to a custom domain in the Amplify console, there are several steps that Amplify must complete before you can view your app using your custom domain. The following list describes each step in the domain set up and activation process.

SSL/TLS creation

If you are using a managed certificate, AWS Amplify issues an SSL/TLS certificate for setting up a secure custom domain.

SSL/TLS configuration and verification

Before issuing a managed certificate, Amplify 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 must manually add the DNS verification record provided in the Amplify console into your domain with a third-party DNS provider.

If you are using a custom certificate, you are responsible for validating domain ownership.

Domain activation

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

Using SSL/TLS certificates

An SSL/TLS certificate is a digital document that allows web browsers to identify and establish encrypted network connections to web sites using the secure SSL/TLS protocol. When you set up your custom domain, you can use the default managed certificate that Amplify provisions for you or you can use your own custom certificate.

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

> **Warning**
>
> Amplify can't renew the certificate if the CNAME verification record has been modified or deleted in the DNS settings with your domain provider. You must delete and add the domain again in the Amplify console.

To use a custom certificate, you must first obtain a certificate from the third-party certificate authority of your choice. Amplify Hosting supports two types of certificates: RSA (Rivest-Shamir-Adleman) and ECDSA (Elliptic Curve Digital Signature Algorithm). Each certificate type must conform to the following requirements.

RSA certificates

- Amplify Hosting supports 1024-bit, 2048-bit, 3072-bit, and 4096-bit RSA keys.
- AWS Certificate Manager (ACM) issues RSA certificates with up to 2048-bit keys.
- To use a 3072-bit or 4096-bit RSA certificate, obtain the certificate externally and import it into ACM. It will then be available for use with Amplify Hosting.

ECDSA certificates

- Amplify Hosting supports 256-bit keys.
- Use the prime256v1 elliptic curve to obtain an ECDSA certificate for Amplify Hosting.

After you obtain a certificate, import it into AWS Certificate Manager. ACM is a service that lets you easily provision, manage, and deploy public and private SSL/TLS certificates for use with AWS services and your internal connected resources. Make sure you request or import the certificate in the US East (N. Virginia) (us-east-1) Region.

Ensure that your custom certificate covers all of the subdomains you plan to add. You can use a wildcard at the beginning of your domain name to cover multiple subdomains. For example, if your domain is `example.com`, you can include the wildcard domain `*.example.com`. This will cover subdomains such as `product.example.com` and `api.example.com`.

After your custom certificate is available in ACM, you will be able to select it during the domain set up process. For instructions on importing certificates into AWS Certificate Manager, see Importing certificates into AWS Certificate Manager in the *AWS Certificate Manager User Guide*.

If you renew or reimport your custom certificate in ACM, Amplify refreshes the certificate data associated with your custom domain. In the case of imported certificates, ACM doesn't manage the renewals automatically. You are responsible for renewing your custom certificates and importing them again.

You can change the certificate in use for a domain at any time. For example, you can switch from the default managed certificate to a custom certificate or change from a custom certificate to a managed certificate. In addition, you can change the custom certificate in use to a different custom certificate. For instructions on updating certificates, see Update the SSL/TLS certificate for a domain.

Adding a custom domain managed by Amazon Route 53

Amazon Route 53 is a highly available and scalable DNS service. For more information, see Amazon Route 53 in the *Amazon Route 53 Developer Guide*. If you already have a Route 53 domain, use the following instructions to connect your custom domain to your Amplify app.

To add a custom domain managed by 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 **Hosting**, **Custom domains**.
4. On the **Custom domains** page, choose **Add domain**.
5. Enter the name of your root domain. For example, if the name of your domain is **https://example.com**, enter **`example.com`**.

   As you start typing, any root domains that you already manage in Route 53 appear in the list. You can choose the domain you want to use from the list. If you don't already own the domain and it is available, you can purchase the domain in Amazon Route 53.
6. After you enter your domain name, choose **Configure domain**.
7. By default, Amplify 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, then configure your domain.
8. Choose the SSL/TLS certificate to use. You can either use the default managed certificate that Amplify provisions for you, or a custom third-party certificate that you have imported into AWS Certificate Manager.
   - Use the default Amplify managed certificate.
     - Choose **Amplify managed certificate**.
   - Use a custom third-party certificate.
     a. Choose **Custom SSL certificate**.
     b. Select the certificate to use from the list.
9. Choose **Add domain**.

> **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.

Adding 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 with Amplify.

If you are using GoDaddy, see the section called "Updating DNS records for a domain managed by GoDaddy" for instructions specific to this provider.

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 **Hosting**, **Custom domains**.
4. On the **Custom domains** page, choose **Add domain**.
5. Enter the name of your root domain. For example, if the name of your domain is **https://example.com**, enter **`example.com`**.
6. Amplify detects that you are not using a Route 53 domain and gives you the option to create a hosted zone in Route 53.
   - To create a hosted zone in Route 53
     a. Choose **Create hosted zone on Route 53**.
     b. Choose **Configure domain**.
     c. Hosted zone name servers are displayed in the console. Go to your DNS provider's website and add the name servers to your DNS settings.
     d. Select **I have added the above name servers to my domain registry**.
     e. Proceed to step seven.
   - To continue with manual configuration
     a. Choose **Manual configuration**
     b. Choose **Configure domain**.
     c. Proceed to step seven.
7. By default, Amplify automatically creates two subdomain entries for your domain. For example, if your domain name is **example.com**, you will see the subdomains **https://**

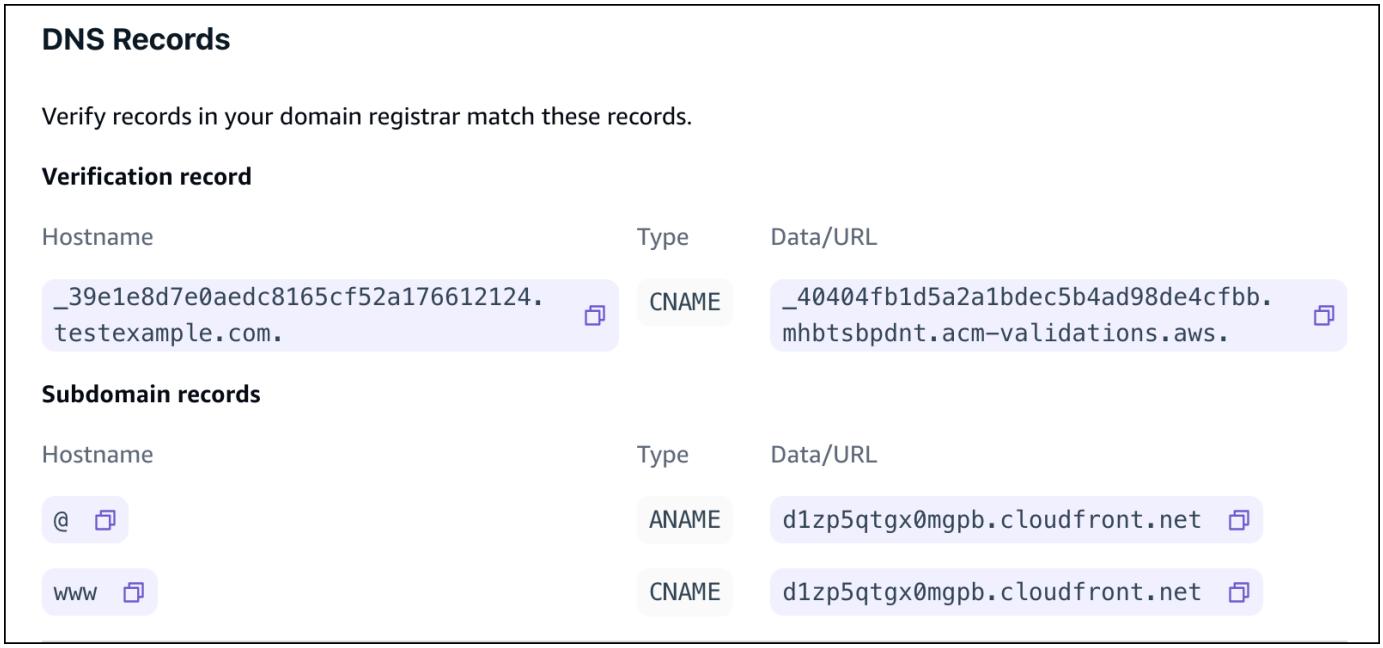
www.example.com and **https://example.com** with a redirect set up from the root domain to the **www** subdomain.

(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 and configure your domain.

8. Choose the SSL/TLS certificate to use. You can either use the default managed certificate that Amplify provisions for you, or a custom third-party certificate that you have imported into AWS Certificate Manager.
   - Use the default Amplify managed certificate.
     - Choose **Amplify managed certificate**.
   - Use a custom third-party certificate.
     a. Choose **Custom SSL certificate**.
     b. Select the certificate to use from the list.
9. Choose **Add domain**.
10. If you chose **Create hosted zone on Route 53** in step six, proceed to step 15.

    If you chose **Manual configuration**, in step six, you must update your DNS records with your third-party domain provider.

    On the **Actions** menu, choose **View DNS records**. The following screenshot shows the DNS records displayed in the console.



11. Do one of the following:

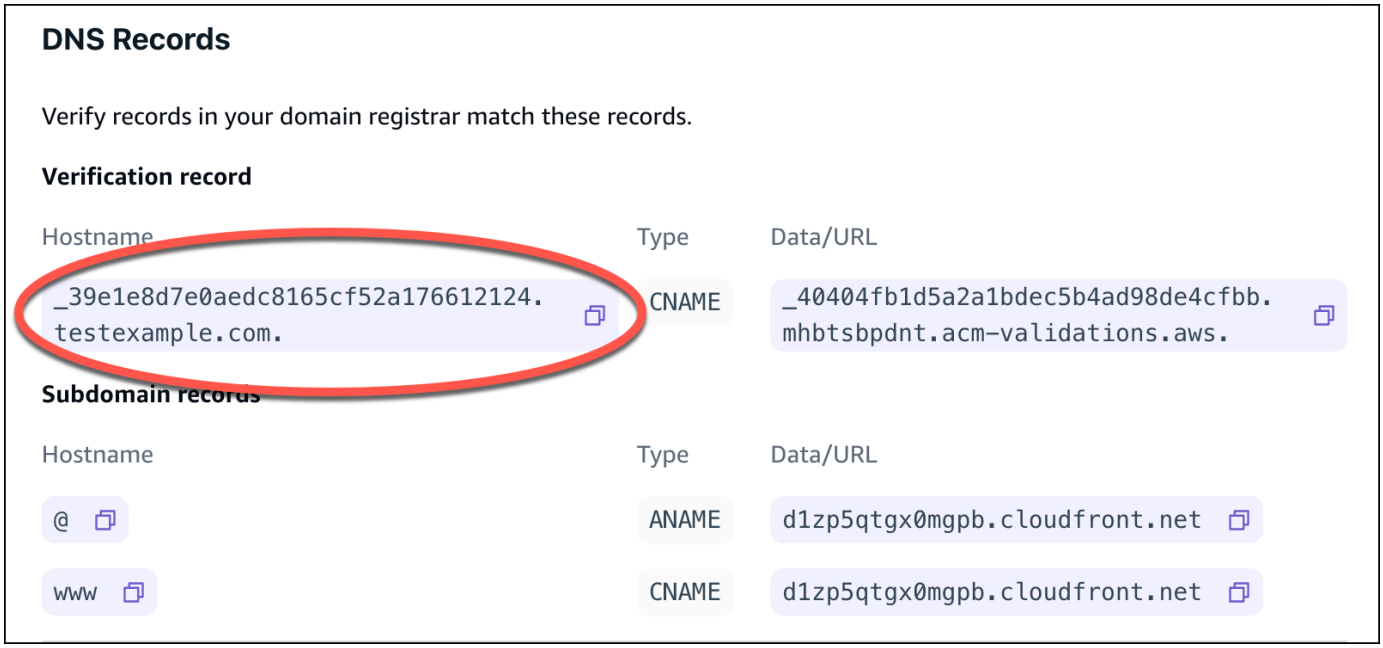
    - If you're using GoDaddy, go to Updating DNS records for a domain managed by GoDaddy.
    - If you're using a different third-party DNS provider, go to the next step in this procedure.

12. Go to your DNS provider's website, log in to your account, and locate the DNS management settings for your domain. You will configure two CNAME records.

13. Configure the first CNAME record to point your subdomain to the AWS validation server.

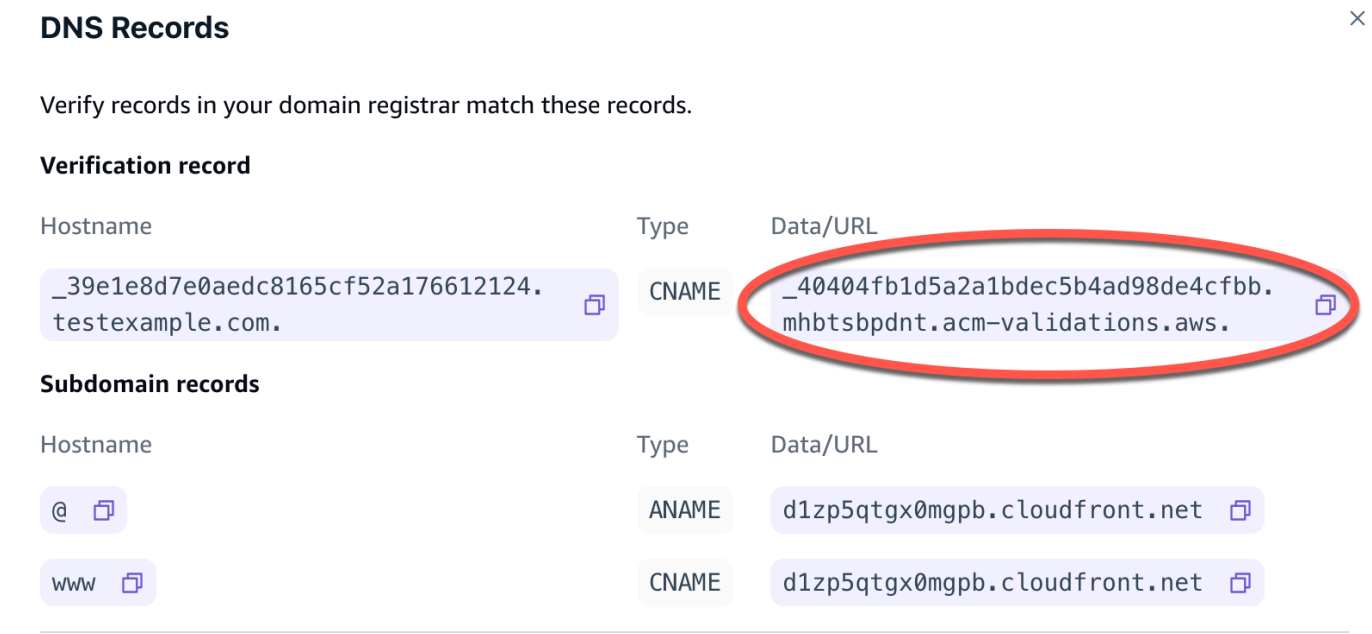
    If the Amplify console displays a DNS record for verifying ownership of your subdomain such as **_c3e2d7eaf1e656b73f46cd6980fdc0e.example.com**, enter only **_c3e2d7eaf1e656b73f46cd6980fdc0e** for the CNAME record subdomain name.

    The following screenshot shows the location of the verification record to use.



If the the Amplify console displays an ACM validation server record such as **_cjhwou20vhu2exampleuw20vuyb2ovb9.j9s73ucn9vy.acm-validations.aws**, enter **`_cjhwou20vhu2exampleuw20vuyb2ovb9.j9s73ucn9vy.acm-validations.aws`** for the CNAME record value.

The following screenshot shows the location of the ACM verification record to use.



Amplify uses this information to verify ownership of your domain and generate an SSL/TLS certificate for your domain. Once Amplify validates ownership of your domain, all traffic will be served using HTTPS/2.

> **Note**
>
> The default Amplify certificate generated by AWS Certificate Manager (ACM) is valid for 13 months and renews automatically as long as your app is hosted with Amplify. Amplify can't renew the certificate if the CNAME verification record has been modified or deleted. You must delete and add the domain again in the Amplify console.

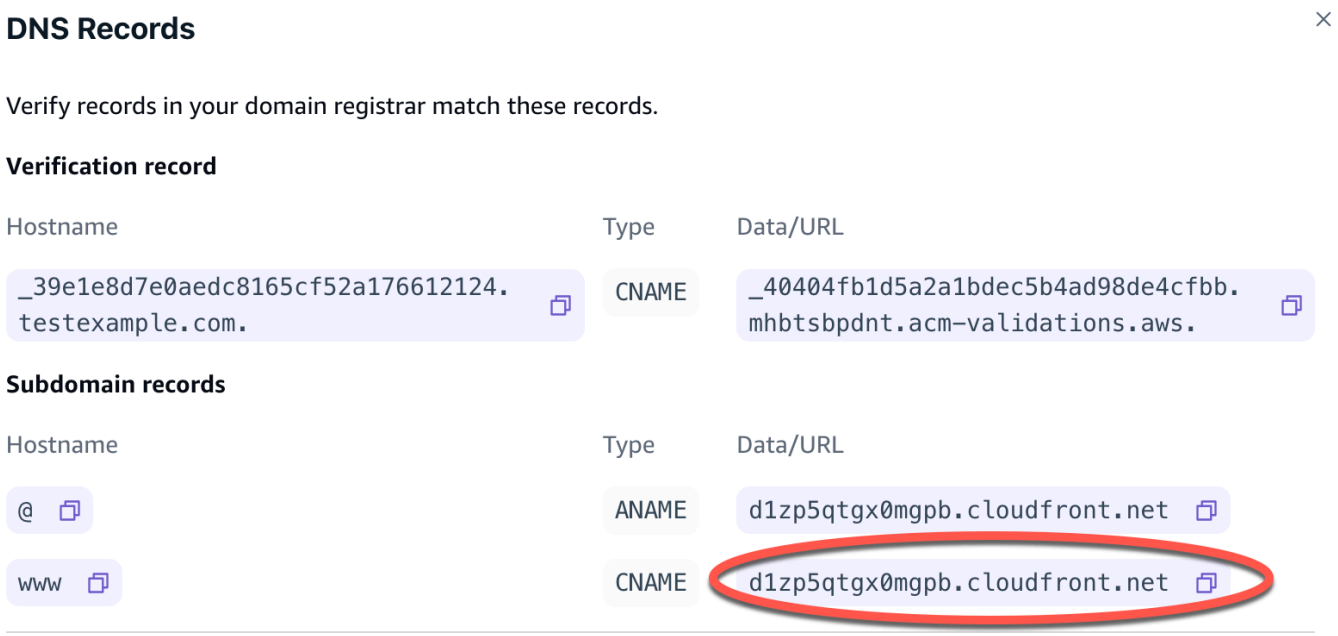
> **Important**
>
> It is important that you perform this step soon after adding your custom domain in the Amplify console. The AWS Certificate Manager (ACM) immediately starts attempting to verify ownership. Over time, the checks become less frequent. If you add or update your CNAME records a few hours after you create your app, this can cause your app to get stuck in the pending verification state.

14. Configure a second CNAME record to point your subdomains to the Amplify domain. For example, if your subdomain is **www.example.com**, enter **www** for the subdomain name.

    If the Amplify console displays the domain for your app as **d111111abcdef8.cloudfront.net**, enter **`d111111abcdef8.cloudfront.net`** for 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.

    The following screenshot shows the location of the domain name record to use.



15. Configure the ANAME/ALIAS record to point to the root domain of your app (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.

Updating DNS records for a domain managed by GoDaddy

If GoDaddy is your DNS provider, use the following instructions to update your DNS records in the GoDaddy UI to finish connecting your Amplify app to your GoDaddy domain.

To add a custom domain managed by GoDaddy

1. Before you can update your DNS records with GoDaddy, complete steps one through nine of the procedure the section called “Adding a custom domain managed by a third-party DNS provider”.

2. Log in to your GoDaddy account.
3. In your list of domains, find the domain to add and choose **Manage DNS**.
4. On the **DNS** page, GoDaddy displays a list of records for your domain in the **DNS Records** section. You need to add two new CNAME records.
5. Create the first CNAME record to point your subdomains to the Amplify domain.

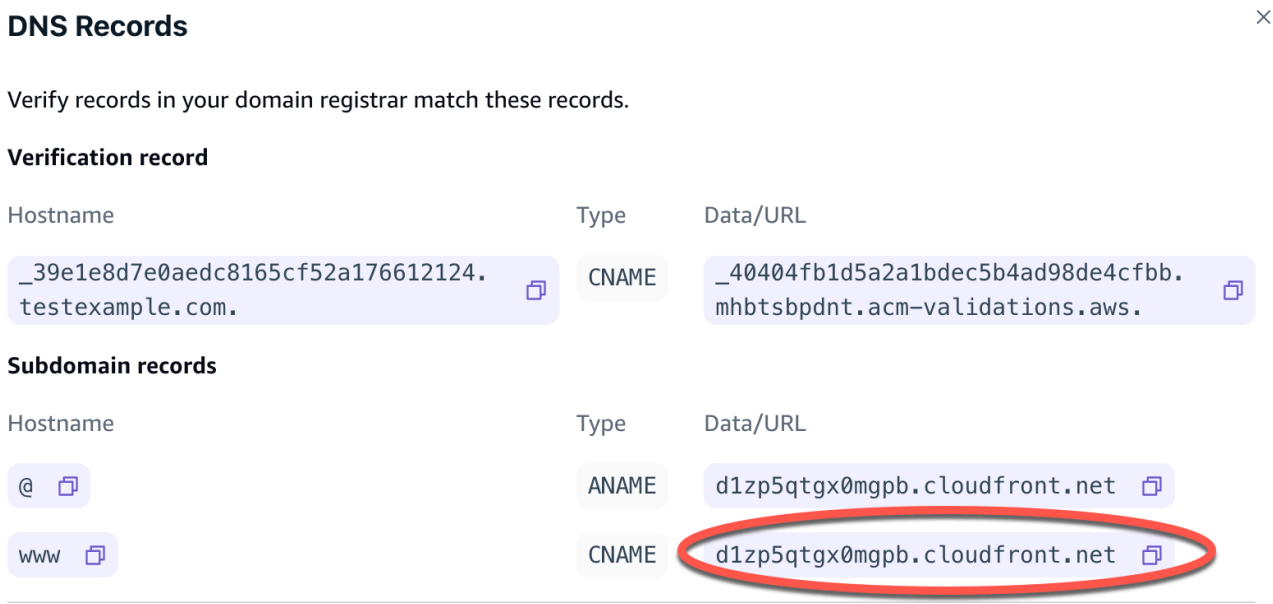
   a. In the **DNS Records** section, choose **Add New Record**.

   b. For **Type**, choose **CNAME**.

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

   d. For **Value**, 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 **d111111abcdef8.cloudfront.net**, enter **`d111111abcdef8.cloudfront.net`** for **Value**.

   The following screenshot shows the location of the domain name record to use.



   e. Choose **Save**.

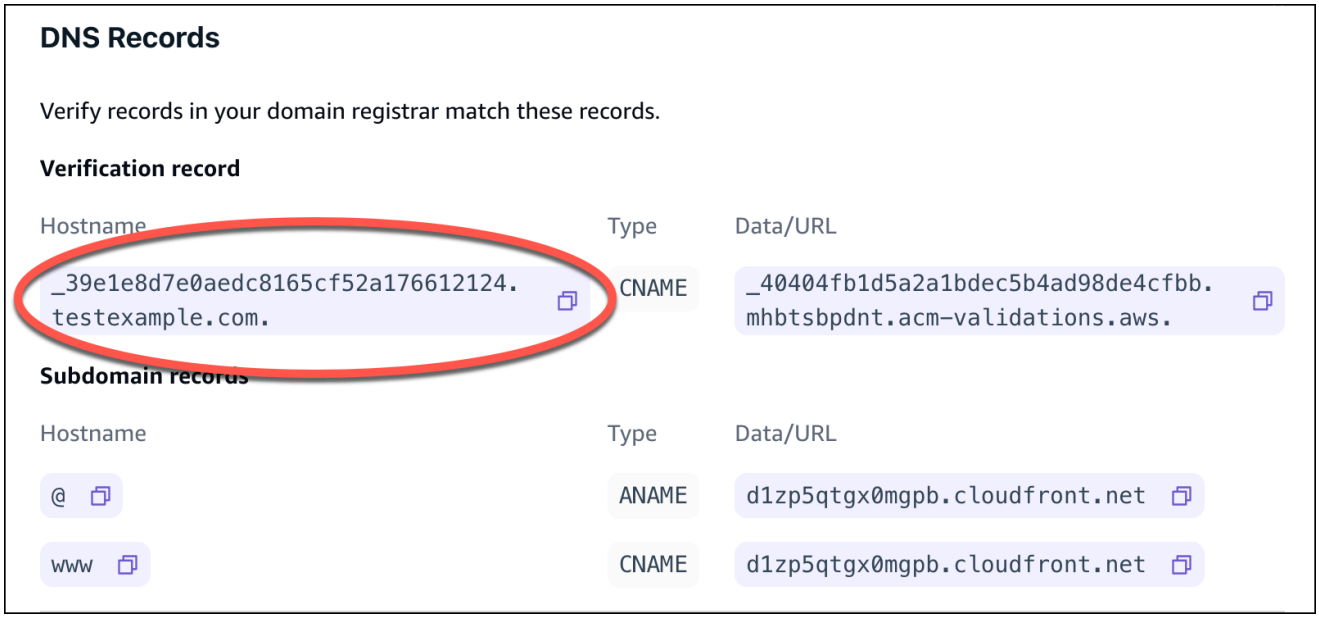
6. Create the second CNAME record to point to the AWS Certificate Manager (ACM) validation server. A single validated ACM generates an SSL/TLS certificate for your domain.

   a. For **Type**, choose **CNAME**.

   b. 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**.

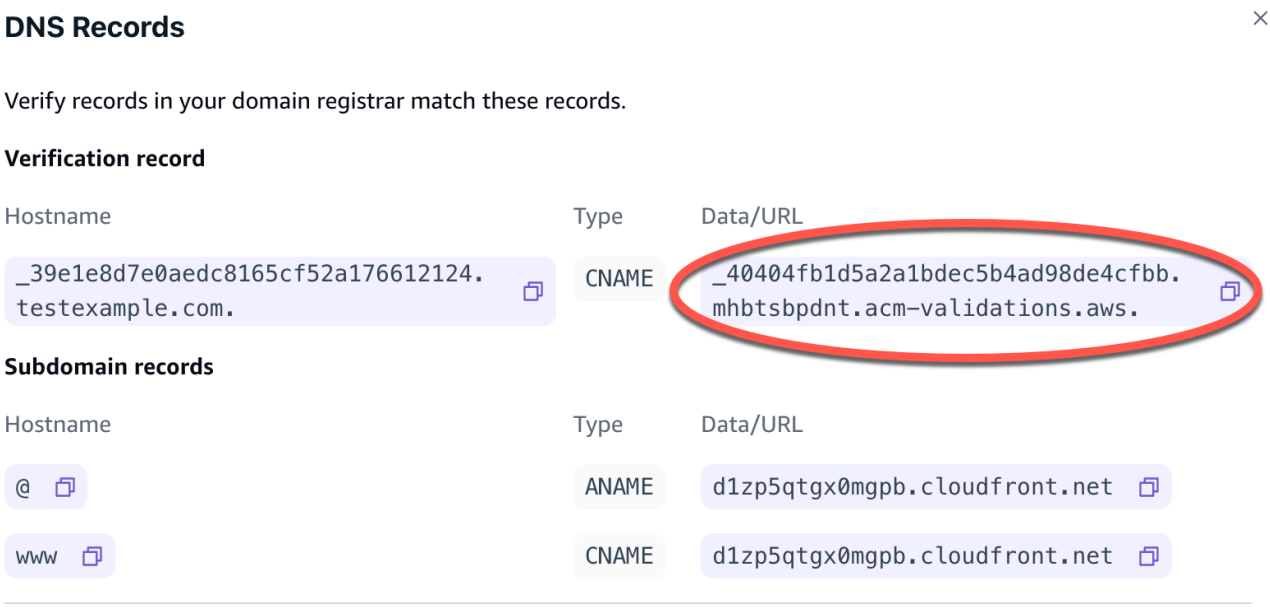
The following screenshot shows the location of the verification record to use.



c. For **Value**, enter the ACM validation certificate.

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

The following screenshot shows the location of the ACM verification record to use.



d. Choose **Save**.

> **Note**
>
> The default Amplify certificate generated by AWS Certificate Manager (ACM) is valid for 13 months and renews automatically as long as your app is hosted with Amplify. Amplify can't renew the certificate if the CNAME verification record has been modified or deleted. You must delete and add the domain again in the Amplify console.

7. 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. On the **DNS** page, locate the menu at the top of the page and choose **Forwarding**.

   b. In the **Domain** section, choose **Add Forwarding**.

   c. Choose **http://**, and then enter the name of your subdomain to forward to (for example, **www.example.com**) for the **Destination URL**.

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

   e. Choose, **Save**.

Updating the SSL/TLS certificate for a domain

You can change the SSL/TLS certificate that is in use for a domain at any time. For example, you can change from using a managed certificate to using a custom certificate. This is helpful if you want to manage the certificate and its expiration notifications. You can also change the custom certificate that is in use for the domain. Making changes to the SSL certificate won't incur any downtime for your active domain. For more information about certificates, see Using SSL/TLS certificates.

Use the following procedure to update the type of certificate or the custom certificate that is in use for a domain.

To update a domain's certificate

1. Sign in to the AWS Management Console and open the Amplify console.
2. Choose your app that you want to update.
3. In the navigation pane, choose **Hosting**, **Custom domains**.
4. On the **Custom domains** page, choose **Domain configuration**.
5. On the details page for your domain, locate the **Custom SSL certificate** section. The procedure for updating your certificate varies depending on the type of change you want to make.
   - To change from a custom certificate to the default Amplify managed certificate
     - Choose **Amplify managed certificate**.
   - To change from a managed certificate to a custom certificate
     a. Choose **Custom SSL certificate**.
     b. Select the certificate to use from the list.
   - To change a custom certificate to a different custom certificate
     - For **Custom SSL certificate**, select the new certificate to use from the list.
6. Choose **Save**. The status details for the domain will indicate that Amplify has initiated the SSL creation process for a managed certificate or the configuration process for a custom certificate.

Managing 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 **Hosting**, and then choose **Custom domains**.
4. On the **Custom domains** page, choose **Add domain**.
5. 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**.
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**.
7. Choose **Add domain**.

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 **Hosting**, and then choose **Custom domains**.
4. On the **Custom domains** page, choose **Add domain**.
5. 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 subdomain **beta.alpha.example.com**, you would enter **beta** as the subdomain value.
6. Choose **Add domain**.

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 **Hosting**, and then choose **Custom domains**.
4. On the **Custom domains** page, choose **Domain configuration**.
5. In the **Subdomains** section, you can edit your existing subdomains as needed.
6. (Optional) To add a new subdomain, choose **Add new**.
7. Choose **Save**.

Setting up wildcard subdomains

Amplify Hosting now supports wildcard subdomains. A wildcard subdomain is a catch-all subdomain that enables you to point existing and non-existing subdomains to a specific branch of your application. When you use a wildcard to associate all subdomains in an app to a specific branch, you can serve the same content to your app's users in any subdomain and avoid configuring each subdomain individually.

To create a wildcard subdomain, specify an asterisk (*) as the subdomain name. For example, if you specify the wildcard subdomain `*.example.com` for a specific branch of your app, any URL that ends with example.com will be routed to the branch. In this case, requests for `dev.example.com` and `prod.example.com` will be routed to the `*.example.com` subdomain.

Note that Amplify supports wildcard subdomains only for a custom domain. You can't use this feature with the default `amplifyapp.com` domain.

The following requirements apply to wildcard subdomains:

- The subdomain name must be specified with an asterisk (*) only.
- You can't use a wildcard to replace part of a subdomain name, like this: *domain.example.com.
- You can't replace a subdomain in the middle of a domain name, like this: subdomain.*.example.com.
- By default, all Amplify provisioned certificates cover all subdomains for a custom domain.

To add or delete a wildcard subdomain

After adding a custom domain to an app, you can add a wildcard subdomain for an app branch.

1. Sign in to the AWS Management Console and open the Amplify Hosting console.
2. Choose your app that you want to manage wildcard subdomains for.
3. In the navigation pane, choose **Hosting**, and then choose **Custom domains**.
4. On the **Custom domains** page, choose **Domain configuration**.
5. In the **Subdomains** section, you can add or delete wildcard subdomains.
   - To add a new wildcard subdomain
     a. Choose **Add new**.
     b. For the subdomain, enter an *****.
     c. For your app branch, select a branch name from the list.
     d. Choose **Save**.
   - To delete a wildcard subdomain
     a. Choose **Remove** next to the subdomain name. Traffic to a subdomain that is not explicitly configured stops, and Amplify Hosting returns a 404 status code to those requests.
     b. Choose **Save**.

Setting up automatic subdomains for an Amazon Route 53 custom domain

After an app is connected to a custom domain in Route 53, Amplify 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 **Hosting**, and then choose **Custom domains**.

4. On the **Custom domains** page, choose **Domain configuration**.
5. In the **Automatic subdomain creation** section, turn on the feature.

> **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 **Automatic subdomain creation** 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 for pull requests.

Troubleshooting custom domains

If you encounter issues when adding a custom domain to an app in the AWS Amplify console, consult Troubleshooting custom domains in the Amplify troubleshooting chapter. If you don't see a solution to your issue there, contact Support. For more information, see Creating a support case in the *AWS Support User Guide*.

Firewall support for Amplify hosted sites

Firewall support for Amplify hosted sites enables you to protect your web applications with a direct integration with AWS WAF. AWS WAF allows you to configure a set of rules, called a web access control list (web ACL), that allow, block, or monitor (count) web requests based on customizable web security rules and conditions that you define. When you integrate your Amplify app with AWS WAF, you gain more control and visibility into the HTTP traffic accepted by your app. To learn more about AWS WAF, see How AWS WAF Works in the *AWS WAF Developer Guide*.

Firewall support is available in all AWS Regions in which Amplify Hosting operates. This integration falls under an AWS WAF global resource, similar to CloudFront. Web ACLs can be attached to multiple Amplify Hosting apps, but they must reside in the same Region.

You can use AWS WAF to protect your Amplify app from common web exploits, such as SQL injection and cross-site scripting. These could affect your app's availability and performance, compromise security, or consume excessive resources. For example, you can create rules to allow or block requests from specified IP address ranges, requests from CIDR blocks, requests that originate from a specific country or region, or requests that contain unexpected SQL code or scripting.

You can also create rules that match a specified string or a regular expression pattern in HTTP headers, method, query string, URI, and the request body (limited to the first 8 KB). Additionally, you can create rules to block events from specific user agents, bots, and content scrapers. For example, you can use rate-based rules to specify the number of web requests that are allowed by each client IP in a trailing, continuously updated, 5-minute period.

To learn more about the types of rules that are supported and additional AWS WAF features, see the AWS WAF Developer Guide and the AWS WAF API Reference.

> **Important**
>
> Security is a shared responsibility between AWS and you. AWS WAF isn't the solution to all internet security issues and you must configure it to meet your security and compliance objectives. To help you understand how to apply the shared responsibility model when using AWS WAF, see Security in your use of the AWS WAF service.

Topics

- Enabling AWS WAF for an Amplify application in the AWS Management Console
- Disassociate a web ACL from an Amplify application
- Enabling AWS WAF for an Amplify application using the AWS CDK
- How Amplify integrates with AWS WAF
- Firewall pricing for Amplify applications

Enabling AWS WAF for an Amplify application in the AWS Management Console

You can enable AWS WAF protections for an Amplify app either in the Amplify console or in the AWS WAF console.

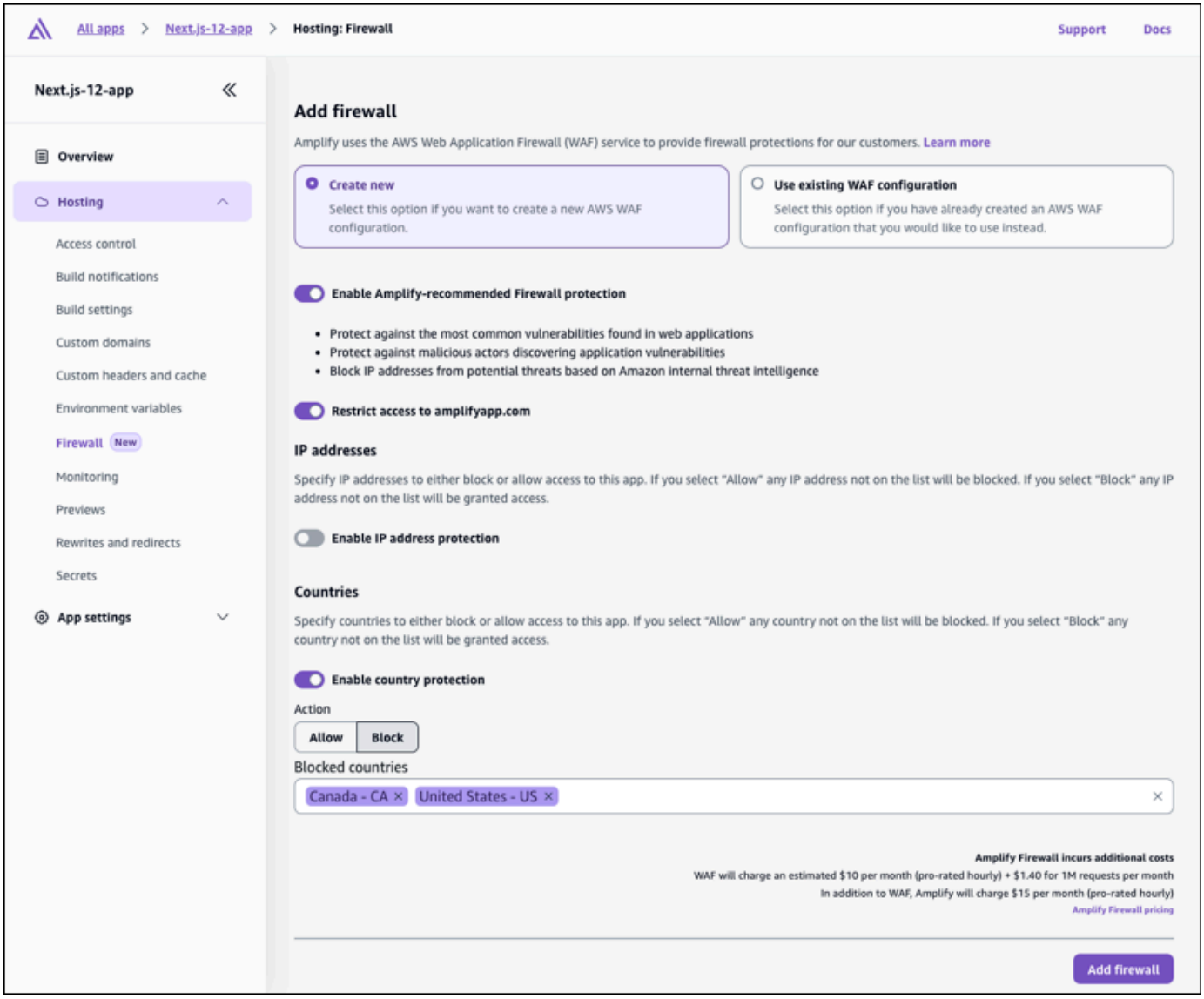
- **Amplify console** — You can enable the Firewall capabilities for an existing Amplify app by associating an AWS WAF web ACL to your app in the Amplify console. Use one-click protection to create a web ACL with pre-configured rules that we consider as best practice for most apps. You have the option to customize access by IP address and country. The instructions in this section describe setting up one-click protections.
- **AWS WAF console**— Use a preconfigured web ACL that you create in the AWS WAF console or by using the AWS WAF APIs. You must create web ACLs that you want to associate with an Amplify app in the Global (CloudFront) Region. Regional web ACLs might already exist in your AWS account, but they are not compatible with Amplify. For getting started instructions, see Setting up AWS WAF and its components in the *AWS WAF Developer Guide*.

Use the following procedure to enable AWS WAF for an existing app in the Amplify console.

Enable AWS WAF for an existing Amplify app

1. Sign in to the AWS Management Console and open the Amplify console at https://console.aws.amazon.com/amplify/.
2. On the **All apps** page, choose the name of the deployed app to enable the Firewall feature on.
3. In the navigation pane, choose **Hosting**, and then choose **Firewall**.

   The following screenshot shows how to navigate to the **Add firewall** page in the Amplify console.



4. On the **Add firewall** page, your actions will depend on whether you want to create a new AWS WAF configuration or use an existing one.

   - Create a new AWS WAF configuration.

     a. Choose **Create new**.

     b. Optionally, enable any of the following configurations:

        i. Turn on **Enable Amplify-recommended Firewall protection**.

        ii. Turn on **Restrict access to amplifyapp.com** to prevent access to your app on the default Amplify domain.

        iii. For **IP addresses**, turn on **Enable IP address protections**.

A. For **Action**, choose **Allow** if you want to specify the IP addresses that will have access and all others will be blocked. Choose **Block** if you want to specify the IP addresses that will be blocked and all others will have access.

B. For **IP version**, select either **IPV4** or **IPV6**.

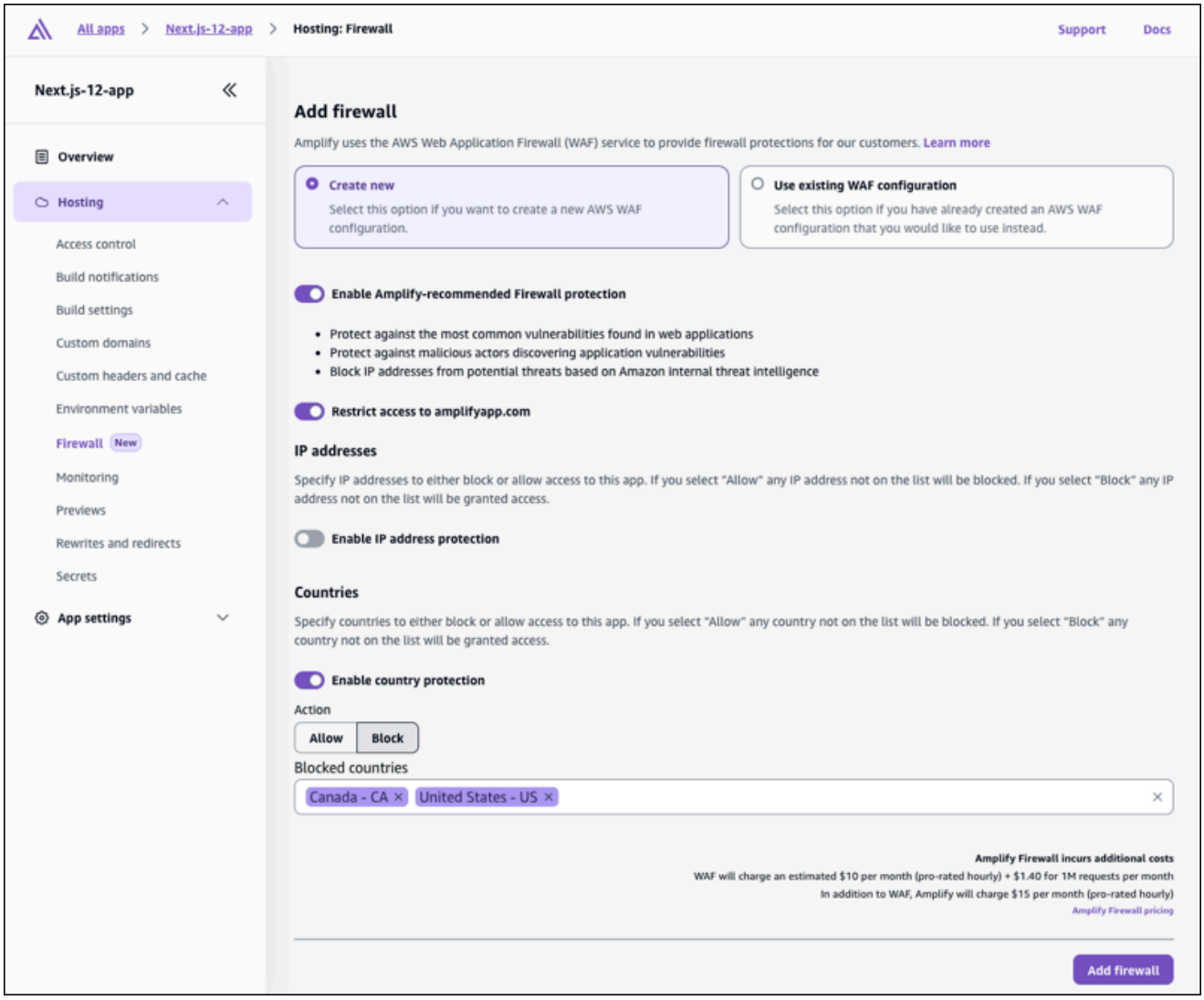
C. In the **IP addresses** text box, enter either your allowed or blocked IP addresses, one per line, in CIDR format.

iv. For **Countries**, turn on **Enable country protection**.

A. For **Action**, choose **Allow** if you want to specify the countries that will have access and all others will be blocked. Choose **Block** if you want to specify the countries that will be blocked and all others will have access.

B. For **Countries**, select either your allowed or blocked countries from the list.

The following screenshot demonstrates how to enable a new AWS WAF configuration for an app.



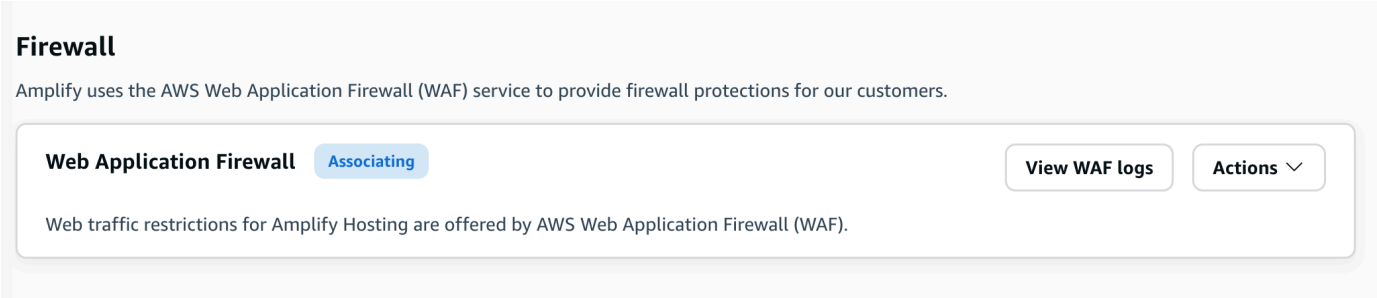
- Use an existing AWS WAF configuration.

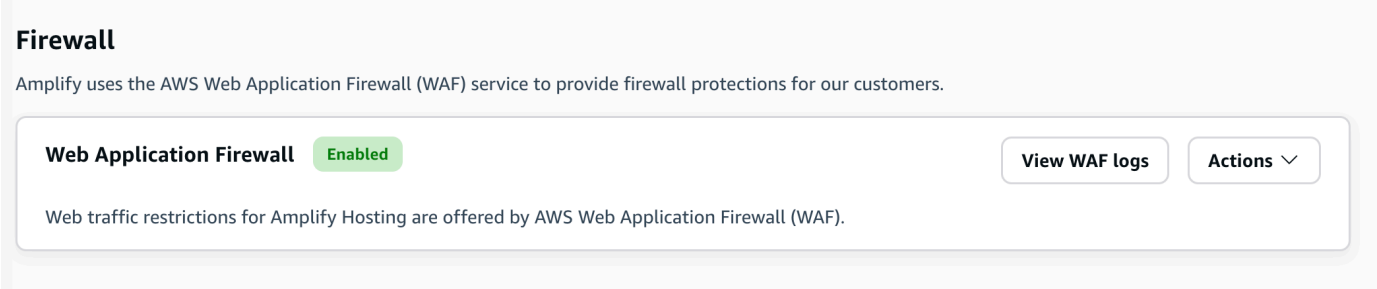
  a. Choose **Use existing AWS WAF configuration**.

  b. Select a saved configuration from the list of web ACLs in AWS WAF in your AWS account. The web ACL that you associate with your Amplify app must be created in the Global (CloudFront) Region. Regional web ACLs might already exist in your AWS account, but they are not compatible with Amplify.

5. Choose **Add firewall**.
6. On the **Firewall** page, the **Associating** status is displayed to indicate that the AWS WAF settings are being propagated. When the process is complete, the status changes to **Enabled**.

   The following screenshots show the firewall progress status in the Amplify console, indicating when the AWS WAF configuration is **Associating** and **Enabled**.





Disassociate a web ACL from an Amplify application

You can't delete a web ACL that is associated with an Amplify app. You must first disassociate the web ACL from the app in the Amplify console. Then you can delete it in the AWS WAF console.

To disassociate a web ACL from an Amplify app

1. Sign in to the AWS Management Console and open the Amplify console at https://console.aws.amazon.com/amplify/.
2. On the **All apps** page, choose the name of the app to disassociate a web ACL from.
3. In the navigation pane, choose **Hosting**, and then choose **Firewall**.
4. On the **Firewall** page, choose **Actions**, then choose **Disassociate firewall**.
5. In the confirmation modal, enter **`disassociate`**, then choose **Disassociate firewall**.
6. On the **Firewall** page, the **Disassociating** status is displayed to indicate that the AWS WAF settings are being propagated.

   When the process is complete, you can delete the web ACL in the AWS WAF console.

Enabling AWS WAF for an Amplify application using the AWS CDK

You can use the AWS Cloud Development Kit (AWS CDK) to enable AWS WAF for an Amplify application. To learn more about using the CDK, see What is the CDK? in the *AWS Cloud Development Kit (AWS CDK) Developer Guide*.

The following TypeScript code example demonstrates how to create an AWS CDK app with two CDK stacks: one for Amplify and one for AWS WAF. Notice that the AWS WAF stack must be deployed to the US East (N. Virginia) (us-east-1) Region. The Amplify application stack can be deployed to a different Region. You must create the web ACL that you want to associate with the Amplify app in the Global (CloudFront) Region. Regional web ACLs might already exist in your AWS account, but they are not compatible with Amplify.

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

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

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

export class WAFStack extends cdk.Stack {
  constructor(scope: Construct, id: string, props: WafStackProps) {
    super(scope, id, props);
    const webAcl = new wafv2.CfnWebACL(this, "WebACL", {
      defaultAction: { allow: {} },
      scope: "CLOUDFRONT",
      rules: [
```

```
        // Add your own rules here.
      ],
      visibilityConfig: {
        cloudWatchMetricsEnabled: true,
        metricName: "my-metric-name",
        sampledRequestsEnabled: true,
      },
    });

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

const app = new cdk.App();

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

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

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

How Amplify integrates with AWS WAF

The following list provides specific details about how Firewall support is integrated with AWS WAF and the constraints to consider when creating web ACLs and associating them with Amplify apps.

- You can enable AWS WAF for any type of Amplify app. This includes any supported framework, server-side rendered (SSR) apps, and fully static sites. AWS WAF is supported for Amplify Gen 1 and Gen 2 apps.

- You must create web ACLs that you want to associate with an Amplify app in the Global (CloudFront) Region. Regional web ACLs might already exist in your AWS account, but they are not compatible with Amplify.
- The web ACL and the Amplify app must be created in the same AWS account. You can use AWS Firewall Manager to replicate AWS WAF rules across AWS accounts, to simplify keeping organization rules centralized and distributed across multiple AWS accounts. For more information, see AWS Firewall Manager in the *AWS WAF Developer Guide*.
- You can share the same web ACL across multiple Amplify apps in the same AWS account. All of the apps must be in the same Region.
- When you associate a web ACL with an Amplify app, the web ACL attaches to every branch in the app by default. When you create new branches, they will have the web ACL.
- When you associate a web ACL to an Amplify app, it is automatically associated with all of the app's domains. However, you can configure rules that apply to a single domain name using Host-header matching rules.
- You can't delete a web ACL that is associated with an Amplify app. Before you delete a web ACL in the AWS WAF console, you need to disassociate it from the app.

Amplify web ACL resource policy

To allow Amplify to access your web ACL, a resource policy is attached to the web ACL during association. Amplify constructs this resource policy automatically, but you can view it using the AWS WAFV2 GetPermissionPolicy API. The following IAM permissions are required for associating a web ACL to an Amplify app.

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

Firewall pricing for Amplify applications

The cost of implementing AWS WAF on an Amplify application is calculated based on the following two components:

- **AWS WAF usage** – You will be charged for your AWS WAF usage acoording to the AWS WAF pricing model. AWS WAF charges are based on the web access control lists (web ACLs) that you create, the number of rules that you add per web ACL, and the number of web requests that you receive. For pricing details, see AWS WAF Pricing.
- **Amplify Hosting integration cost** – There is a $15.00 per month, per app charge when you attach a web ACL to an Amplify application. This is prorated hourly.

Feature branch deployments and team workflows

Amplify Hosting is designed to work with feature branch and GitFlow workflows. Amplify uses Git branches to create a new deployment each time you connect a new branch in your repository. After you connect your first branch, you create additional feature branches.

To add a branch to an app

1. Choose the app you want to add a branch to.
2. Choose **App settings**, then **Branch settings**.
3. On the **Branch settings** page, choose **Add branch**.
4. Select a branch from your repository.
5. Choose **Add branch.**
6. Redeploy your app.

After you add a branch, your app has two deployments available at the Amplify default domains, such as *https://main.appid.amplifyapp.com* and *https://dev.appid.amplifyapp.com*. This may vary from team-to-team, but typically the **main branch** tracks release code and is your production branch. The **develop branch** is used as an integration branch to test new features. This enables beta testers to 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 fullstack Amplify Gen 2 apps
- Team workflows with fullstack Amplify Gen 1 apps
- Pattern-based feature branch deployments
- Automatic build-time generation of Amplify config (Gen 1 apps only)
- Conditional backend builds (Gen 1 apps only)
- Use Amplify backends across apps (Gen 1 apps only)

Team workflows with fullstack Amplify Gen 2 apps

AWS Amplify Gen 2 introduces a TypeScript-based, code-first developer experience for defining backends. To learn about fullstack workflows with Amplify Gen 2 applications, see Fullstack workflows in the *Amplify docs*.

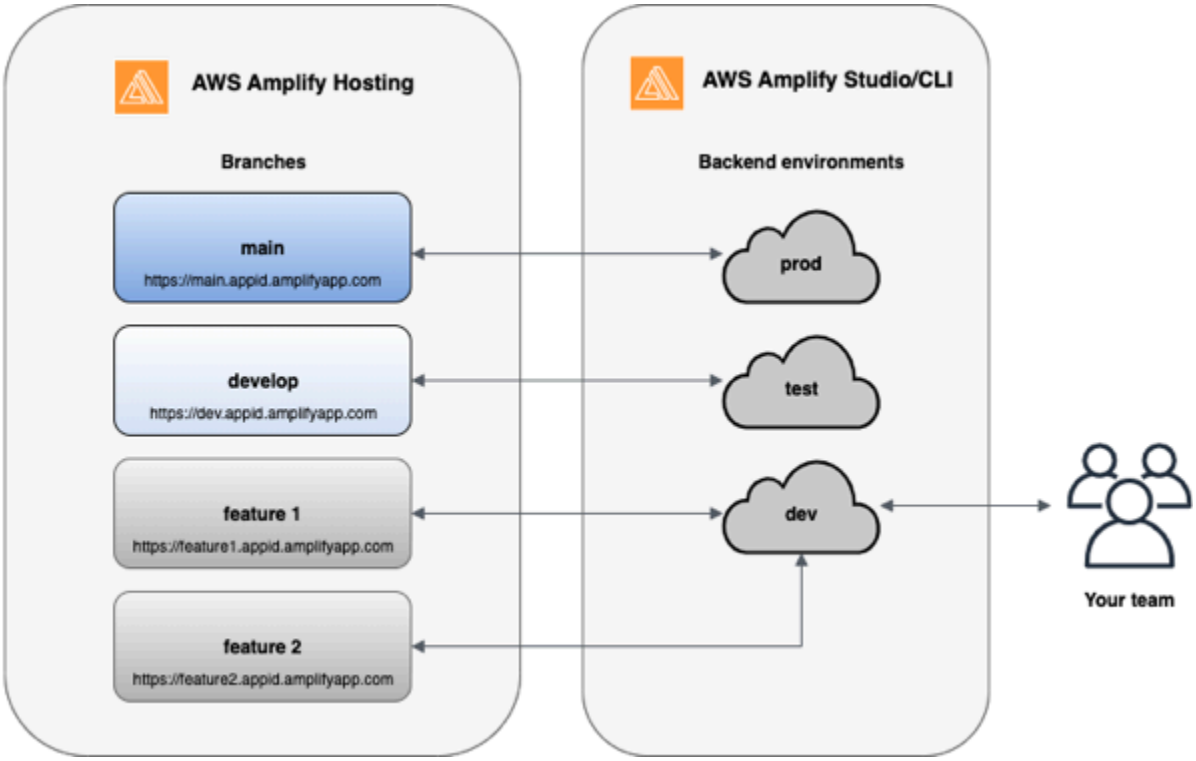
Team workflows with fullstack Amplify Gen 1 apps

A feature branch deployment consists of a **frontend**, and an optional **backend** environment. The frontend is built and deployed to a global content delivery network (CDN), while the backend is deployed by Amplify Studio or the Amplify CLI to AWS. To learn how to set up this deployment scenario, see Building a backend for an application.

Amplify Hosting continuously deploys backend resources such as GraphQL APIs and Lambda functions with your feature branch deployments. You can use the following branching models to deploy your backend and frontend with Amplify Hosting.

Feature branch workflow

- Create **prod**, **test**, and **dev** backend environments with Amplify Studio or the Amplify CLI.
- Map the **prod** backend to the **main** branch.
- Map the **test** backend to the **develop** branch.
- Team members can use the **dev** backend environment for testing individual **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.

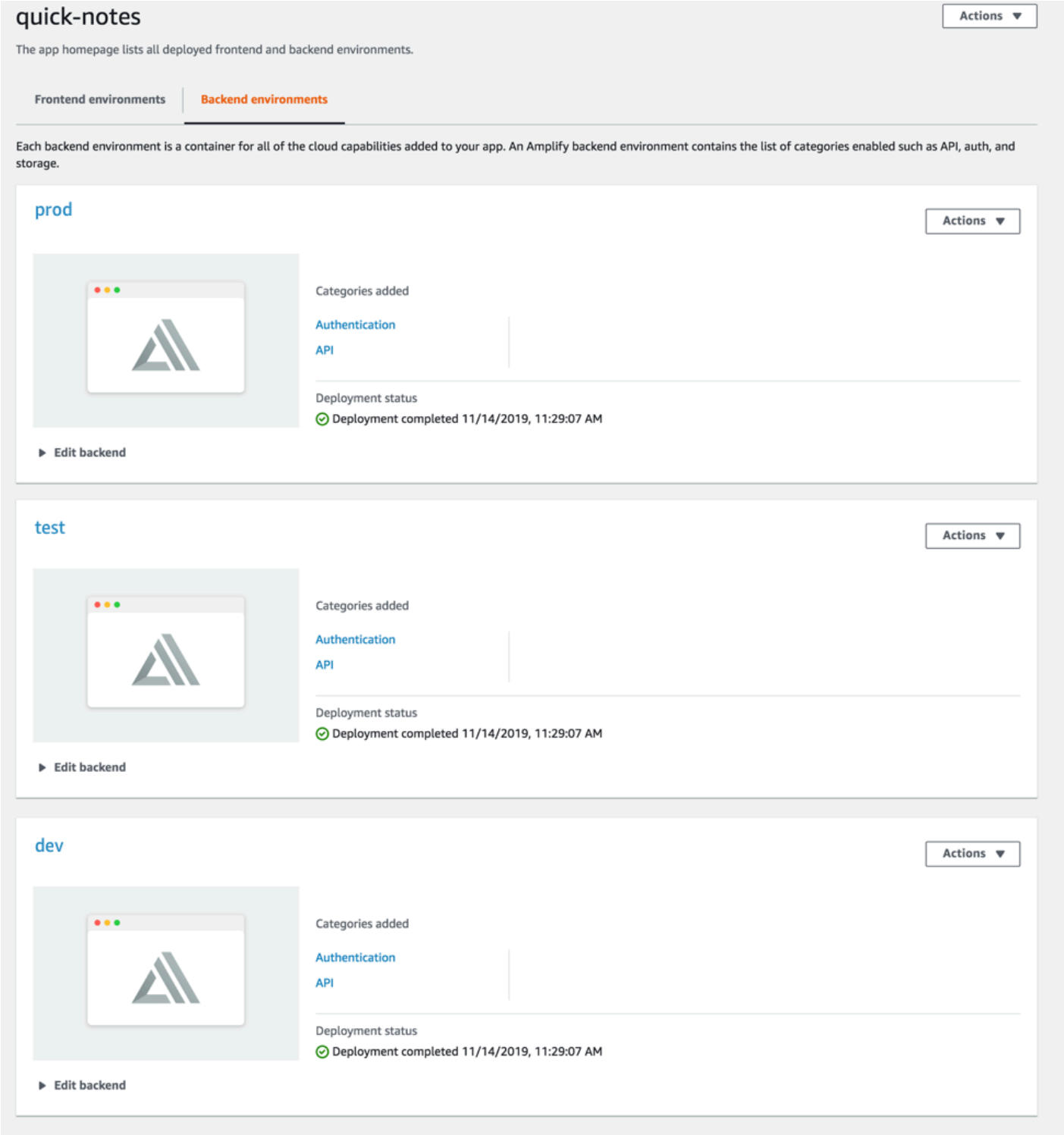
```
amplify env add
 ? Do you want to use an existing environment? (Y/n): n
 ? Enter a name for the environment: test
...
amplify push
```

```
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).

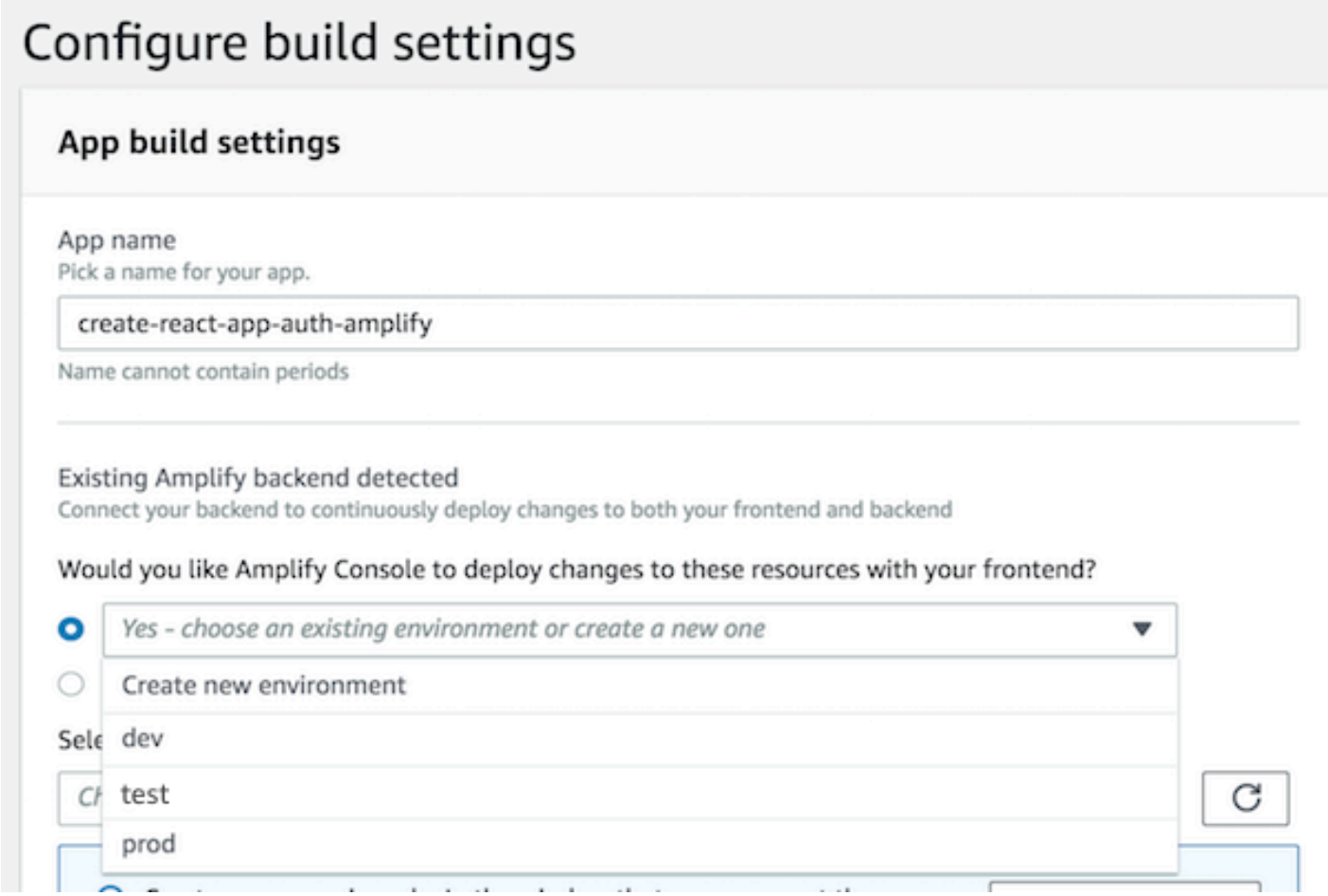
```
git commit -am 'Added dev, test, and prod environments'
git push origin main
```

5. Visit Amplify 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.

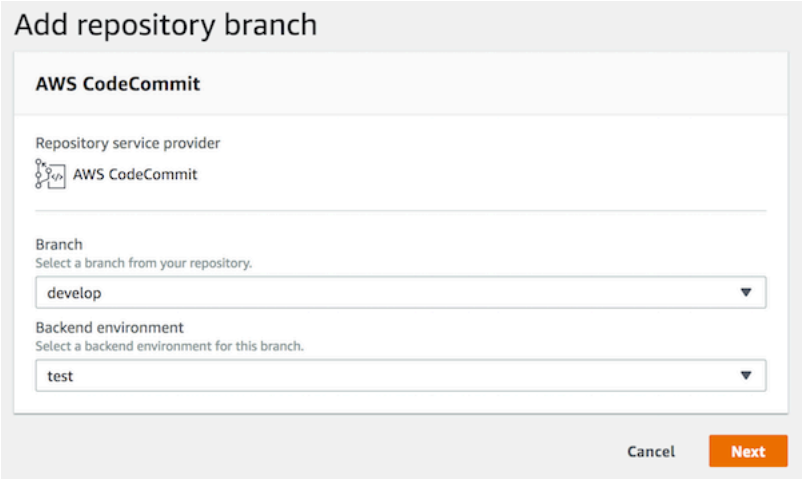


6. Switch to the **Frontend environments** tab and connect your repository provider and *main* branch.
7. On the build settings page, select an existing backend environment to set up continuous deployment with the main branch. Choose *prod* from the list and grant the service role to

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



9. Amplify is now set up. 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 Amplify 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 Amplify 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.

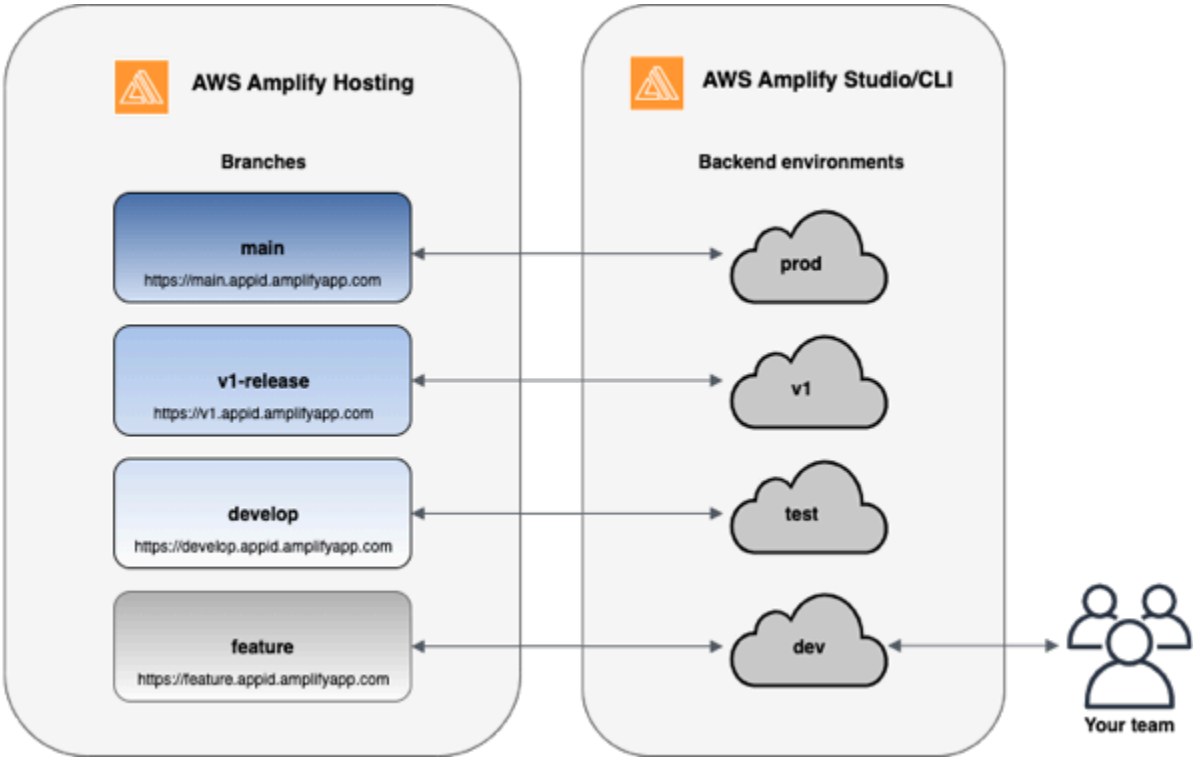
14. Delete your feature branch from Git, Amplify, and remove the backend environment from the cloud (you can always spin up a new one based on by running ‘amplify env checkout prod’ and running ‘amplify env add’).

```
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 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).

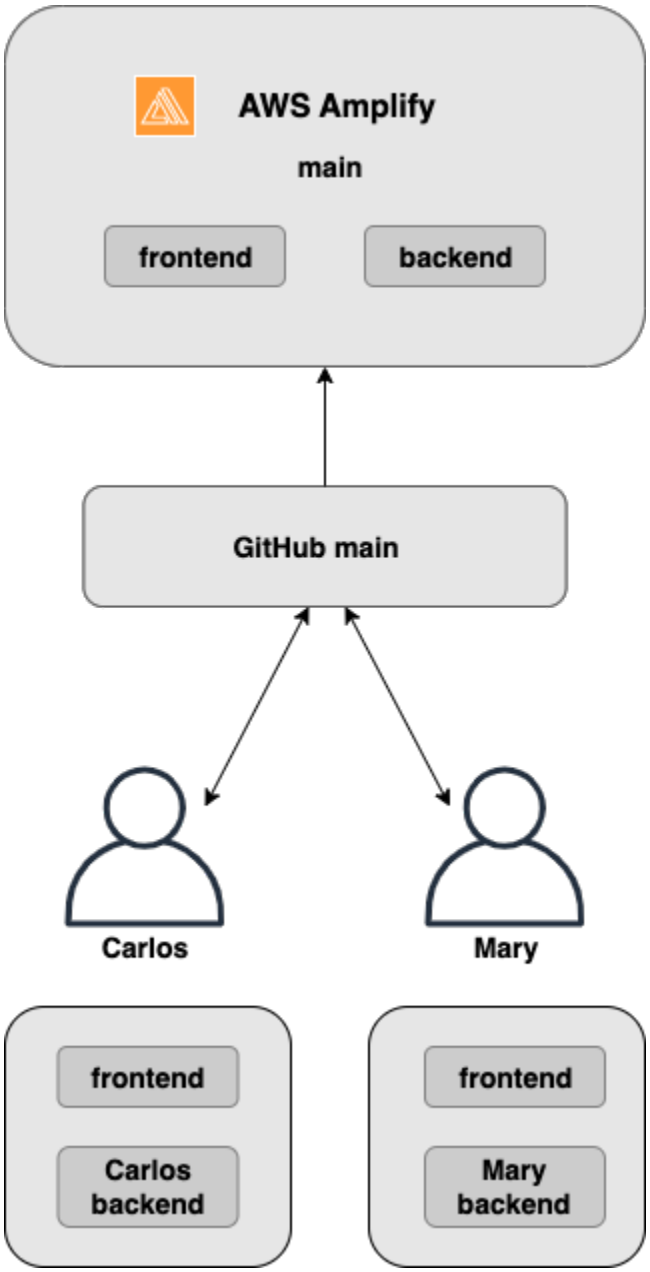
The diagram below shows a recommended setup with GitFlow. You can follow the same process as described in the feature branch workflow section above.



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 Amplify has its own backend. This ensures that the Amplify 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 *mary* 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: mary
...
amplify push
```

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

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

4. Connect your repo > *main* to Amplify.
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. 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 (assume *develop* and *main* branch are the same at this point) and choose *Create*

Pattern-based feature branch deployments

Pattern-based branch deployments allow you to automatically deploy branches that match a specific pattern to Amplify. 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.

1. Choose **App settings**, then **Branch settings**.
2. On the **Branch settings** page, choose **Edit**.
3. Select **Branch autodetection** to automatically connect branches to Amplify that match a pattern set.
4. In the **Branch autodetection - patterns** box, enter the patterns for automatically deploying branches.
   - **`*`** – Deploys all branches in your repository.
   - **`release*`** – Deploys all branches that begin 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*`**.

5. Set up automatic password protection for all branches that are automatically created by selecting **Branch autodetection access control** .
6. For Gen 1 applications built with an Amplify backend, you can choose to create a new environment for every connected branch, or point all branches to an existing backend.
7. Choose **Save**.

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 Setting up automatic subdomains for an Amazon Route 53 custom domain
- For instructions on connecting an Amplify app to a custom domain managed in Route 53, see Adding a custom domain managed by Amazon Route 53
- For more information about using Route 53, see What is Amazon Route 53.

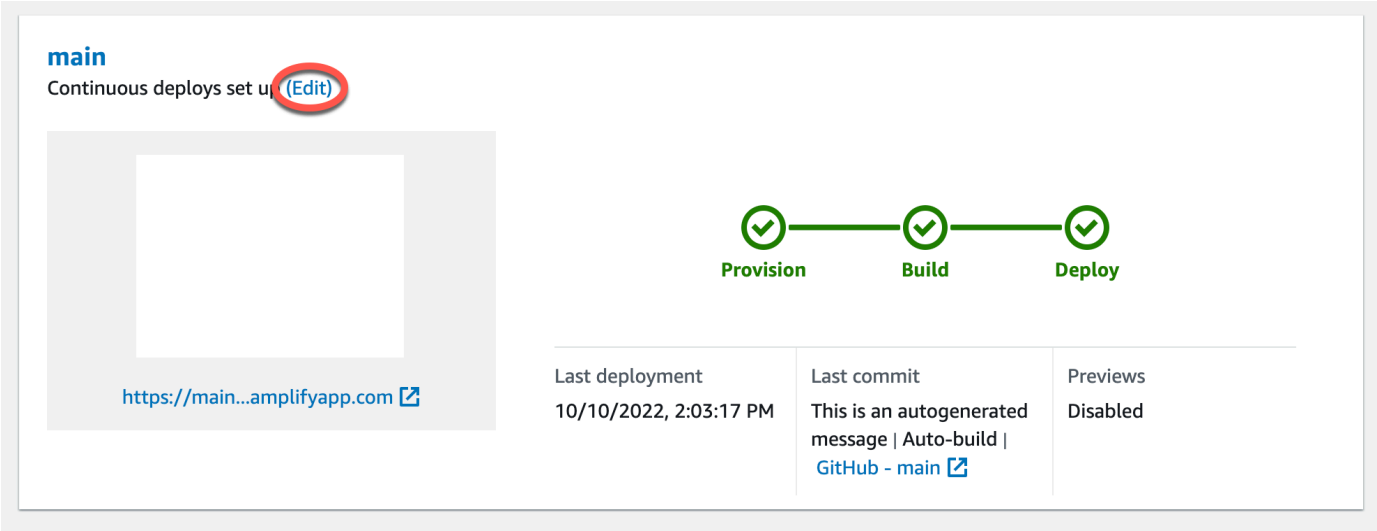
Automatic build-time generation of Amplify config (Gen 1 apps only)

> **Note**
>
> The information in this section is for Gen 1 apps only. If you want to automatically deploy infrastructure and application code changes from feature branches for a Gen 2 app, see Fullstack branch deployments in the *Amplify docs*

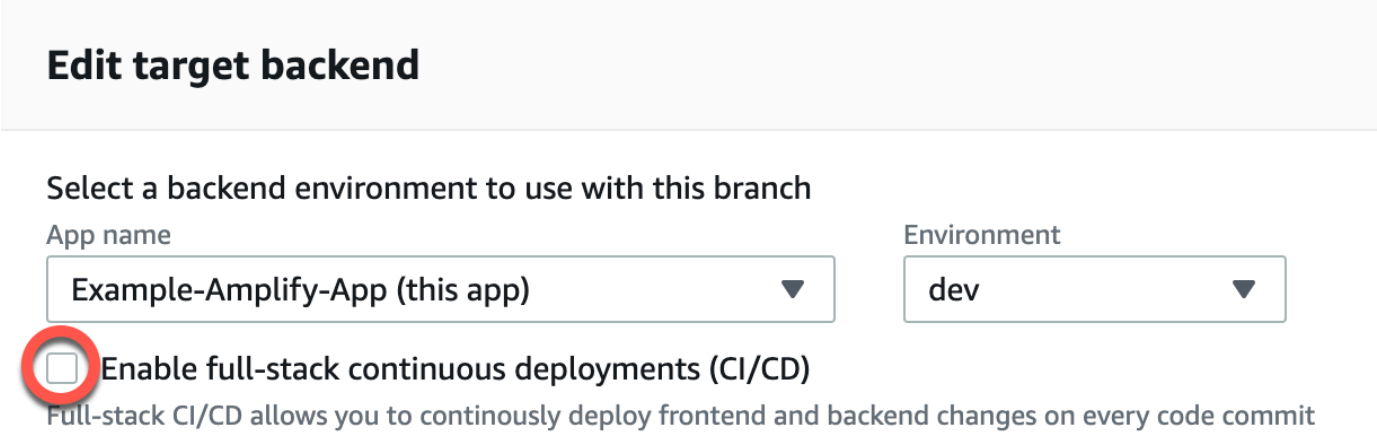
Amplify supports the automatic build-time generation of the Amplify config `aws-exports.js` file for Gen 1 apps. By turning off full stack CI/CD deployments, you enable your app to autogenerate the `aws-exports.js` file and ensure that updates are not made to your backend at build-time.

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 **Hosting environments** tab.
4. Locate the branch to edit and choose **Edit**.



5. On the **Edit target backend** 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 with permissions to deploy backend resources.
7. Choose **Save**. Amplify applies these changes the next time you build the app.

Conditional backend builds (Gen 1 apps only)

> **Note**
>
> The information in this section is for Gen 1 apps only. Amplify Gen 2 introduces a TypeScript-based, code-first developer experience. Therefore, this feature isn't necessary for Gen 2 backends.

Amplify supports conditional backend builds on all branches in a Gen 1 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 Configuring diff based backend builds for a Gen 1 app.

If you currently have custom commands specified in the build settings of your backend phase, conditional backend builds won't work. If you want those custom commands to run, you must move them to the frontend phase of your build settings in your app's `amplify.yml` file. For more information about updating the `amplify.yml` file, see Build specification reference.

Use Amplify backends across apps (Gen 1 apps only)

> **Note**
>
> The information in this section is for Gen 1 apps only. If you want to share backend resources for a Gen 2 app, see Share resources across branches in the *Amplify docs*

Amplify enables you to reuse existing backend environments across all of your Gen 1 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**, **Build an app**.
   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 **Hosting 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 **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. By default, full-stack CI/CD is turned off. 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.
   d. 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 with permissions to deploy backend resources.
   e. Choose **Next**.
7. Choose **Save and deploy**.

Reuse backends when connecting a branch to an existing app

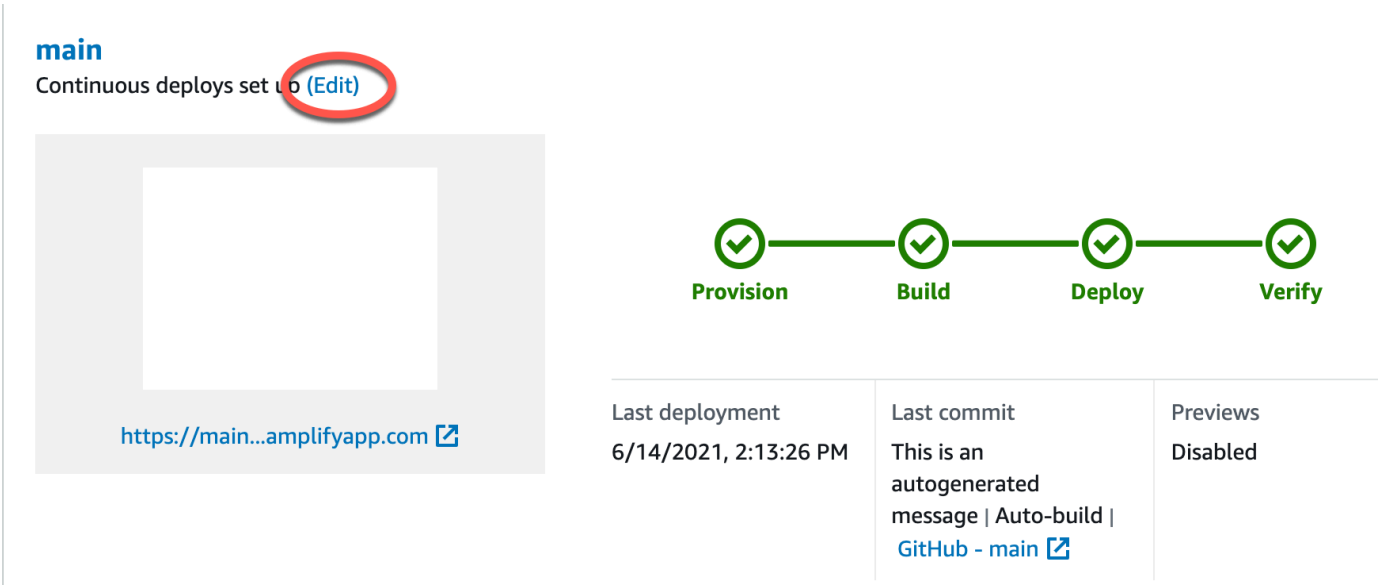
To reuse a backend when connecting a branch to an existing Amplify app

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 with permissions to deploy backend resources.
9. By default, full-stack CI/CD is turned off. 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.
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 **Hosting environments** tab.
4. Locate the branch to edit and choose **Edit**.



5. On the **Select a backend environment to use with this branch** page, for **App name**, select the frontend app that you want to edit the backend environment for. You can choose the current app or any other app in the current region.
6. For **Backend 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**. Amplify applies these changes the next time you build the app.

Building a backend for an application

With AWS Amplify you can build a fullstack application with data, authentication, storage, and frontend hosting that is deployed to AWS.

AWS Amplify Gen 2 introduces a TypeScript-based, code-first developer experience for defining backends. To learn how to use Amplify Gen 2 to build and connect a backend to your app, see Build & connect backend in the *Amplify docs*.

If you are looking for the documentation for building a backend for a Gen 1 app, using the CLI and Amplify Studio, see Build & connect backend in the *Gen 1 Amplify docs*.

Topics

- Create a backend for a Gen 2 app
- Create a backend for a Gen 1 app

Create a backend for a Gen 2 app

For a tutorial that guides you through the steps for creating an Amplify Gen 2 fullstack application with a TypeScript-based backend, see Get started in the *Amplify docs*.

Create a backend for a Gen 1 app

In this tutorial, you will set up a fullstack CI/CD workflow with Amplify. You will deploy a frontend app to Amplify Hosting. Then you will create a backend using Amplify Studio. Finally, you will connect the cloud backend to the frontend app.

Prerequisites

Before you begin this tutorial, complete the following prerequisites.

Sign up for an AWS account

If you are not already an AWS customer, you need to create an AWS account by following the online instructions. Signing up enables you to access Amplify and other AWS services that you can use with your application.

Create a Git repository

Amplify supports GitHub, Bitbucket, GitLab, and AWS CodeCommit. Push your application to your Git repository.

Install the Amplify Command Line Interface (CLI)

For instructions, see Install the Amplify CLI in the *Amplify Framework Documentation*.

Step 1: Deploy a frontend

If you have an existing frontend app in a git repository that you want to use for this example, you can proceed to the instructions for deploying a frontend app.

If you need to create a new frontend app to use for this example, you can follow the Create React App instructions in the *Create React App documentation*.

To deploy a frontend app

1. Sign in to the AWS Management Console and open the Amplify console.
2. On the **All apps** page, choose **New app**, then **Host web app** in the upper right corner.
3. Select your GitHub, Bitbucket, GitLab, or AWS CodeCommit repository provider and then choose **Continue**.
4. Amplify authorizes access to your git repository. For GitHub repositories, Amplify now uses the GitHub Apps feature to authorize Amplify access.

   For more information about installing and authorizing the GitHub App, see Setting up Amplify access to GitHub repositories.
5. On the **Add repository branch** page do the following:

   a. In the **Recently updated repositories** list, select the name of the repository to connect.

   b. In the **Branch** list, select the name of the repository branch to connect.

   c. Choose **Next**.
6. On the **Configure build settings** page, choose **Next**.
7. On the **Review** page, choose **Save and deploy**. When the deployment is complete, you can view your app on the `amplifyapp.com` default domain.

> **Note**
>
> To augment the security of your Amplify applications, the *amplifyapp.com* domain is registered in the Public Suffix List (PSL). For further security, we recommend that you use cookies with a `__Host-` prefix if you ever need to set sensitive cookies in the default domain name for your Amplify applications. This practice will help to defend your domain against cross-site request forgery attempts (CSRF). For more information see the Set-Cookie page in the Mozilla Developer Network.

Step 2: Create a backend

Now that you have deployed a frontend app to Amplify Hosting, you can create a backend. Use the following instructions to create a backend with a simple database and GraphQL API endpoint.

To create a backend

1. Sign in to the AWS Management Console and open the Amplify console.
2. On the **All apps** page, select the app that you created in *Step 1*.
3. On the app homepage, choose the **Backend environments** tab, then choose **Get started**. This initiates the set up process for a default **staging** environment.
4. After the set up finishes, choose **Launch Studio** to access the **staging** backend environment in Amplify Studio.

Amplify Studio is a visual interface to create and manage your backend and accelerate your frontend UI development. For more information about Amplify Studio, see the Amplify Studio documentation.

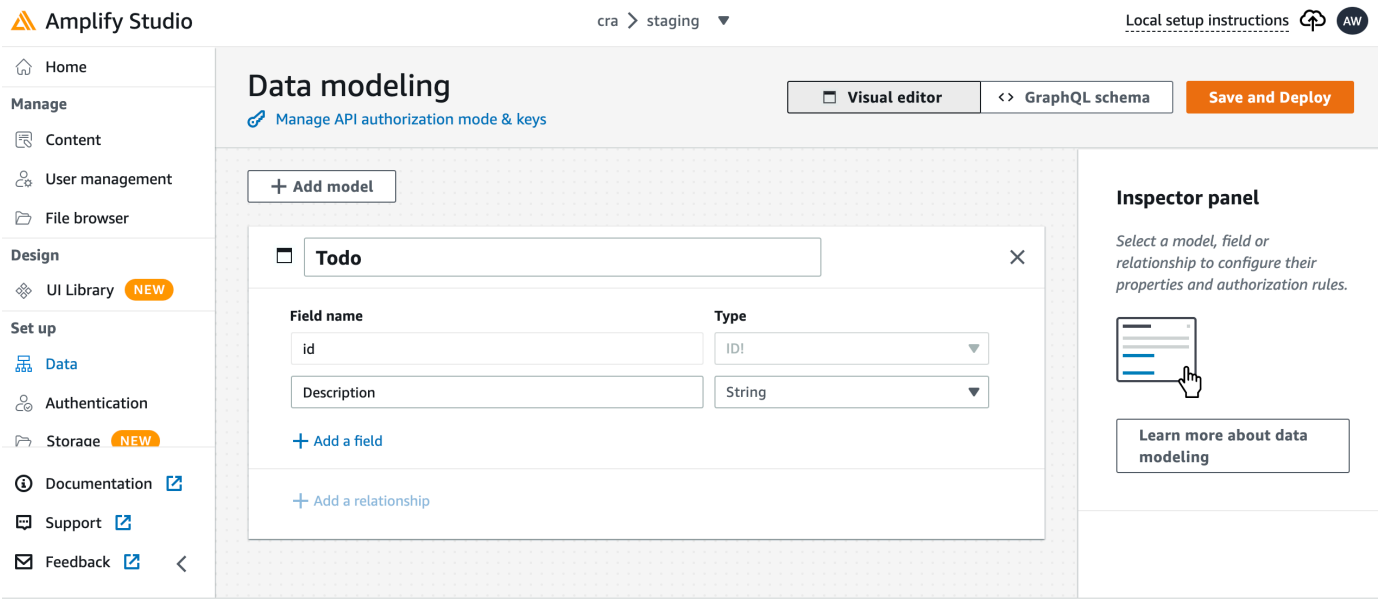
Use the following instructions to create a simple database using the Amplify Studio visual backend builder interface.

Create a data model

1. On the home page for your app's **staging** environment, choose **Create data model**. This opens the data model designer.
2. On the **Data modeling** page, choose **Add model**.
3. For the title, enter **Todo**.

4. Choose **Add a field**.
5. For **Field name**, enter **`Description`**.

   The following screenshot is an example of how your data model will look in the designer.



6. Choose **Save and Deploy**.
7. Return to the Amplify Hosting console and the **staging** environment deployment will be in progress.

During deployment, Amplify Studio creates all the required AWS resources in the backend, including an AWS AppSync GraphQL API to access data and an Amazon DynamoDB table to host the Todo items. Amplify uses AWS CloudFormation to deploy your backend, which enables you to store your backend definition as infrastructure-as-code.

Step 3: Connect the backend to the frontend

Now that you have deployed a frontend and created a cloud backend that contains a data model, you need to connect them. Use the following instructions to pull your backend definition down to your local app project with the Amplify CLI.

To connect a cloud backend to a local frontend

1. Open a terminal window and navigate to the root directory of your local project.
2. Run the following command in the terminal window, replacing the red text with the unique app ID and backend environment name for your project.

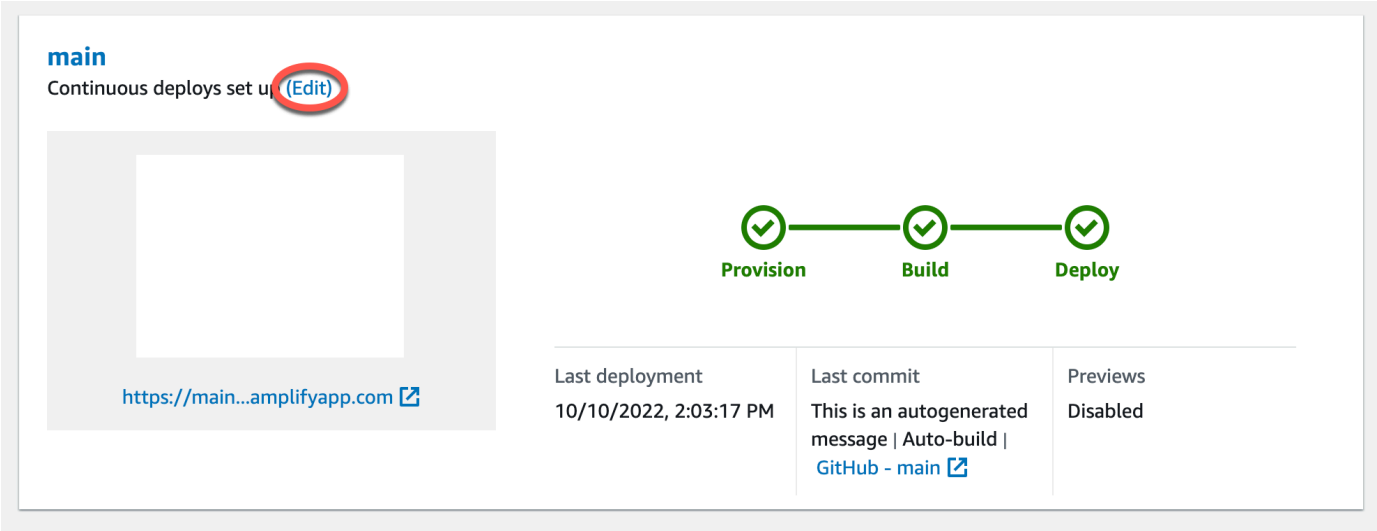
```
amplify pull --appId abcd1234 --envName staging
```

3. Follow the instructions in the terminal window to complete the project set up.

Now you can configure the build process to add the backend to the continuous deployment workflow. Use the following instructions to connect a frontend branch with a backend in the Amplify Hosting console.

To connect a frontend app branch and cloud backend

1. On the app homepage, choose the **Hosting environments** tab.
2. Locate the **main** branch and choose **Edit**.



3. In the **Edit target backend** window, for **Environment**, select the name of the backend to connect. In this example, choose the **staging** backend that you created in *Step 2*.

   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.

4. Next, you must set up a service role to give Amplify the permissions it requires to make changes to your app backend. You can either use an existing service role or create a new one. For instructions, see Adding a service role with permissions to deploy backend resources.
5. After adding a service role, return to the **Edit target backend** window and choose **Save**.

6. To finish connecting the **staging** backend to the **main** branch of the frontend app, perform a new build of your project.

   Do one of the following:

   - From your git repository, push some code to initiate a build in the Amplify console.
   - In the Amplify console, navigate to the app's build details page and choose **Redeploy this version**.

Next steps

Set up feature branch deployments

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

Create a frontend UI in Amplify Studio

Use Studio to build your frontend UI with a set of ready-to-use UI components, and then connect it to your app backend. For more information and tutorials, see the user guide for Amplify Studio in the *Amplify Framework Documentation*.

Advanced deployment features

This chapter covers advanced deployment features that enhance your Amplify Hosting workflow. These features provide additional controls and capabilities to help teams manage deployments more effectively, ensure code quality, and maintain security throughout the development lifecycle.

Learn how to protect your feature branches with password authentication to restrict access to unreleased features. Enable web previews for pull requests to review changes on unique preview URLs before merging code to production branches. Set up end-to-end testing using the Cypress framework to catch regressions before pushing code to production. While the Deploy to Amplify button feature is no longer available, you can still easily deploy applications directly from your repository using Amplify Hosting.

Topics

- Restricting access to an Amplify app's branches
- Web previews for pull requests
- Setting up end-to-end Cypress tests for your Amplify application
- Using the Deploy to Amplify button to share a GitHub project

Restricting access to an Amplify app's branches

If you are working on unreleased features, you can password protect feature branches to restrict access to specific users. When access control is set on a branch, users are prompted for a user name and password when they attempt to access the URL for the branch.

You can set a password that applies to an individual branch or globally to all connected branches. When access control is enabled at both the branch and global level, the branch level password takes precedence over a global (application) level password.

Amplify throttles failed requests that are attempting to access password protected resources. This behavior protects applications against dictionary attacks or other attempts to read data behind access controls.

Use the following procedure to set a password to restrict access to an Amplify app's branches.

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 **Hosting**, and then choose **Access control**.
4. In the **Access control settings** section, choose **Manage access**.
5. On the **Manage access control** page, do one of the following.
   - To set a username and password that applies to all connected branches
     - Turn on **Manage access for all branches**. For example, if you have **main**, **dev**, and **feature** branches connected, you can apply the same username and password for all branches.
   - To set a a username and password that applies to an individual branch
     a. Turn off **Manage access for all branches**.
     b. Locate the branch that you want to manage. For **Access settings** choose **Restricted-password required**.
     c. For **Username**, enter a username.
     d. For **Password**, enter a password.
   - Choose **Save**.
6. If you are managing access control for a server-side rendered (SSR) app, redeploy the app by performing a new build from your Git repository. This step is required to enable Amplify to apply your access control settings.

Web previews for pull requests

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 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 or Amplify Studio, every pull request (**private Git repositories only**) creates a temporary backend that is deleted when the PR is closed.

When web previews are turned on for your app, each PR counts toward the Amplify quota of 50 branches per app. To avoid exceeding this quota, make sure to close your PRs. For more information about quotas, see Amplify Hosting service quotas.

> **Note**
>
> Currently, the `AWS_PULL_REQUEST_ID` environment variable is not available when using AWS CodeCommit as your repository provider.

Web preview security

For security purposes, you can enable web previews on all apps with private repositories, but not on all apps with public repositories. If your Git repository is public, you can set up previews only for apps that don't require an IAM service role. For example, apps with backends and apps that are deployed to the `WEB_COMPUTE` hosting platform require an IAM service role. Therefore, you can't enable web previews for these types of apps if their repository is public. Amplify enforces this restriction to prevent third parties from submitting arbitrary code that would run using your app's IAM role permissions.

When web previews are enabled for an application in a public repository, with an SSR Compute role, you need to carefully manage which branches can access the role. We recommend that you don't use an app-level role. Instead, you should attach a Compute role at the branch-level. This allows you to grant permissions only to the branches that require access to specific resources. For more information, see Adding an SSR Compute role to allow access to AWS resources .

Enable web previews for pull requests

For apps stored in a GitHub repo, web previews use the Amplify GitHub App for repo access. If you are enabling web previews on an existing Amplify app that you previously deployed from a GitHub repo using OAuth for access, you must first migrate the app to use the Amplify GitHub App. For migration instructions, see Migrating an existing OAuth app to the Amplify GitHub App.

To enable web previews for pull requests

1. Choose **Hosting**, then **Previews**.

> **Note**
>
> **Previews** is visible in the **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.

2. For GitHub repositories only, do the following to install and authorize the Amplify GitHub App in your account:

   a. In the **Install GitHub App to enable previews** window, choose **Install GitHub app**.

   b. Select the GitHub account where you want to configure the Amplify GitHub App.

   c. A page opens on Github.com to configure repository permissions for your account.

   d. Do one of the following:

      - To apply the installation to all repositories, choose **All repositories**.
      - To limit the installation to the specific repositories that you select, choose **Only select repositories**. Make sure to include the repo for the app that you are enabling web previews for in the repositories that you select.

   e. Choose **Save**

3. After you enable previews for your repo, return to the Amplify console to enable previews for specific branches. On the **Previews** page, select a branch from the list and choose **Edit settings**.
4. On the **Manage preview settings** page, turn on **Pull request previews**. Then choose **Confirm**.
5. For fullstack applications do one of the following:
   - Choose, **Create new backend environment for every Pull Request**. This option enables you to test changes without impacting production.
   - Choose **Point all Pull Requests for this branch to an existing environment**.
6. Choose **Confirm**.

The next time you submit a pull request for the branch, Amplify builds and deploys your PR to a preview URL. After the pull request is closed, the preview URL is deleted, and any temporary backend environment linked to the pull request is deleted. For GitHub repositories only, you can access a preview of your URL directly from the pull request in your GitHub account.

Web preview access with subdomains

Web previews for 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 Setting up automatic subdomains for an Amazon Route 53 custom domain.

Setting up end-to-end Cypress tests for your Amplify application

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 YAML. Currently, you can run only the Cypress testing framework during a build.

Cypress is a JavaScript-based testing framework that allows you to run E2E tests on a browser. For a tutorial that demonstrates how to set up E2E tests, see the blog post Running end-to-end Cypress tests for your fullstack CI/CD deployment with Amplify.

Adding Cypress tests to an existing Amplify application

You can add Cypress tests to an existing app by updating the app's build settings in the Amplify console. The build specification YAML contains a collection of build commands and related settings that Amplify uses to run your build. Use the `test` step to run any test commands at build time. For E2E tests, Amplify Hosting offers a deeper integration with Cypress that allows you to generate a UI report for your tests.

The following list describes the test settings and how they are used.

preTest

Install the dependencies required to run Cypress tests. Amplify Hosting uses mochawesome 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 perform tests using mochawesome.

postTest

The mochawesome report is generated from the output JSON. Note that if you are using Yarn, you must run this command in silent mode to generate the mochawesome report. For Yarn, you can use the following command.

```
yarn run --silent mochawesome-merge cypress/report/mochawesome-report/
mochawesome*.json > cypress/report/mochawesome.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.

The following example excerpt from a build specification `amplify.yml` file shows how to add Cypress tests to your app.

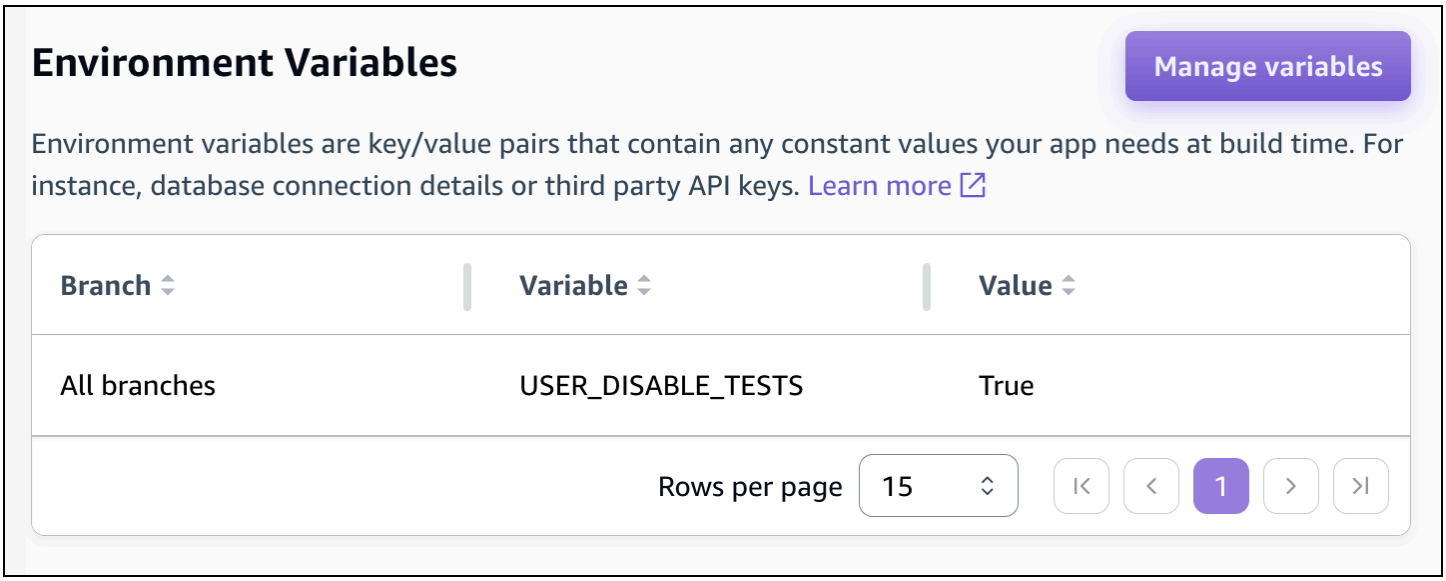
```
test:
  phases:
    preTest:
      commands:
        - npm ci
        - npm install -g pm2
        - npm install -g wait-on
        - npm install mocha mochawesome mochawesome-merge mochawesome-report-generator
        - pm2 start npm -- start
        - wait-on http://localhost:3000
    test:
      commands:
        - 'npx cypress run --reporter mochawesome --reporter-options
 "reportDir=cypress/report/mochawesome-
report,overwrite=false,html=false,json=true,timestamp=mmddyyyy_HHMMss"'
    postTest:
      commands:
```

```
        - npx mochawesome-merge cypress/report/mochawesome-report/mochawesome*.json >
cypress/report/mochawesome.json
        - pm2 kill
  artifacts:
    baseDirectory: cypress
    configFilePath: '**/mochawesome.json'
    files:
      - '**/*.png'
      - '**/*.mp4'
```

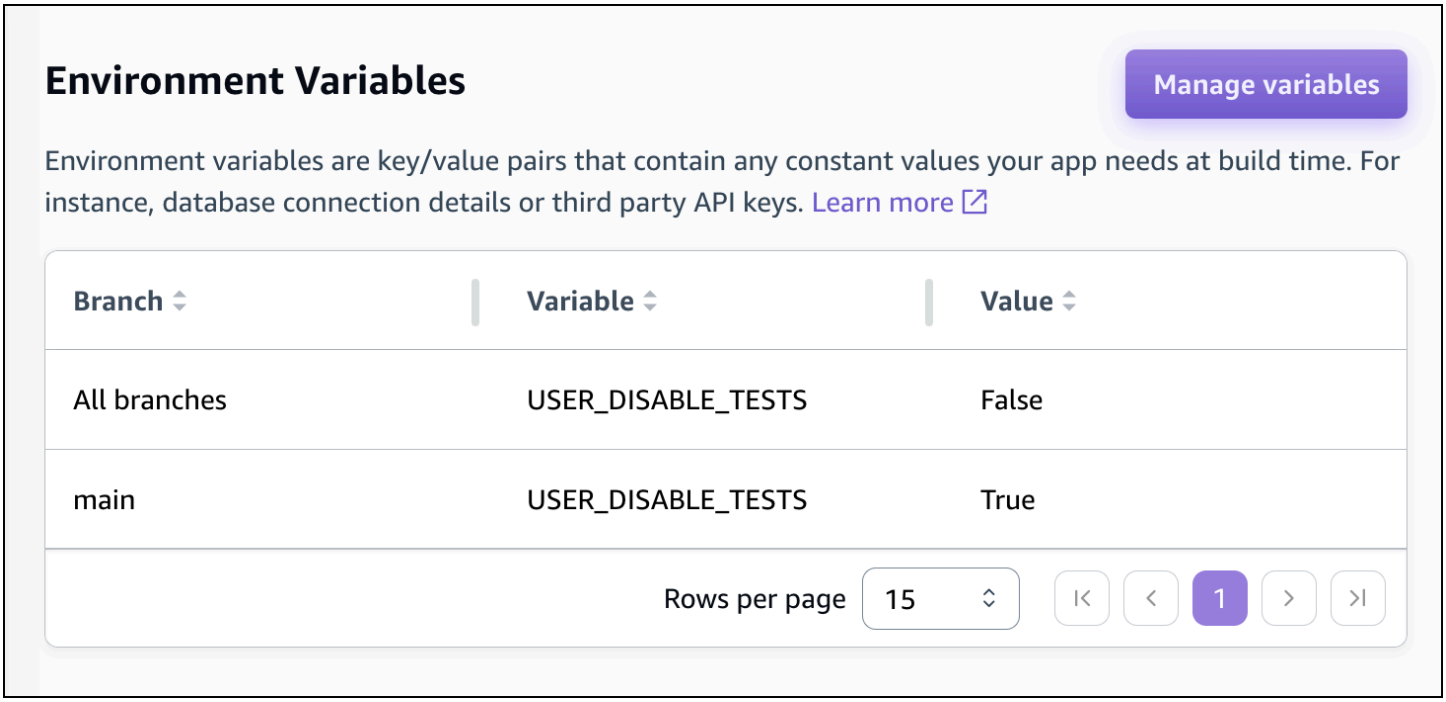
Turning off tests for an Amplify application or branch

After the test configuration has been added to your `amplify.yml` build settings, the `test` step runs for every build, on every branch. If you want to globally disable tests from running, or only run tests for specific branches, you can use the USER_DISABLE_TESTS environment variable 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. The following screenshot, shows the **Environment variables** section in the Amplify console with tests disabled for all branches.



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 want to disable with a value of `true`. In the following screenshot, 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 the Deploy to Amplify button to share a GitHub project

> **Important**
>
> One-click deployment using the **Deploy to Amplify Hosting** button is no longer available. To deploy from a repository, create a new application in Amplify Hosting. For instructions, see Getting started with deploying an app to Amplify Hosting.

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

Adding the Deploy to Amplify Hosting button to a repository or blog

Add the 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 located at the URL `https://oneclick.amplifyapp.com/button.svg`
2. The Amplify console URL with a link to your GitHub repository. Your can either copy your repository's URL, such as `https://github.com/username/repository`, or you can provide a deep link into a specific folder, such as `https://github.com/username/repository/tree/branchname/folder`. Amplify Hosting will deploy the default branch in your repository. Additional branches can be connected after the app is connected.

Use the following example to add the button to a markdown file, such as your GitHub README.md. Replace `https://github.com/username/repository` with the URL to your repository.

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

Use the following example to add the button to any HTML document. Replace `https://github.com/username/repository` with the URL to your repository.

```
<a href="https://console.aws.amazon.com/amplify/home#/deploy?repo=https://github.com/
username/repository">
    <img src="https://oneclick.amplifyapp.com/button.svg" alt="Deploy to Amplify
 Hosting">
</a>
```

Setting up redirects and rewrites for an Amplify application

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.

Understanding the redirects that Amplify supports

Amplify supports the following redirect types in the console.

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 a user interface.

> **Note**
>
> If your app is returning an unexpected 302 response, the error is likely caused by changes you've made to your app's redirect and custom header configuration. To resolve this issue, verify that your custom headers are valid, and then re-enable the default 404 rewrite rule for your app.

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.

Understanding the order of redirects

Redirects are applied 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
```

Understanding how Amplify forwards query parameters

You can use query parameters for more control over your URL matches. Amplify forwards all query parameters to the destination path for 301 and 302 redirects, with the following exceptions:

- If the original address includes a query string set to a specific value, Amplify doesn't forward query parameters. In this case, the redirect only applies to requests to the destination URL with the specified query value.
- If the destination address for the matching rule has query parameters, query parameters aren't forwarded. For example, if the destination address for the redirect is `https://example-target.com?q=someParam`, query parameters aren't passed through.

Creating and editing redirects in the Amplify console

You can create and edit redirects for an application in the Amplify console. Before you get started, you will need the following information about the parts of a redirect.

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 geographical region.

To create a redirect in the Amplify console

1. Sign in to the AWS Management Console and open the Amplify console.
2. Choose the app you want to create a redirect for.
3. In the navigation pane, choose **Hosting**, and then choose **Rewrites and redirects**.
4. On the **Rewrites and redirects** page, choose **Manage redirects**.
5. Manually add or update redirects in the **Rewrites and redirects** JSON editor.
   a. For `source`, specify the original address the user requested.
   b. For `status`, specify the type of redirect.
   c. For `target`, specify the destination address that renders the content to the user.
   d. (Optional) For `condition`, enter a two letter country code condition.
6. Choose **Save**.

Redirects and rewrites example reference

This section provides examples for a variety of common redirect scenarios. You can use these examples to understand the JSON syntax for creating your own redirects and rewrites in the Amplify console JSON editor.

> **Note**
>
> Original address domain matching is case-insensitive.

Topics

- Simple redirects and rewrites
- Redirects for single page web apps (SPA)
- Reverse proxy rewrite
- Trailing slashes and clean URLs
- Placeholders
- Query strings and path parameters
- Region-based redirects
- Using wildcard expressions in redirects and rewrites

Simple redirects and rewrites

You can use the following example 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 format

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

You can use the following example 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 format

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

```
]
```

You can use the following example 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 format

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

You can use the following example 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 format

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

You can use the following example to redirect to a different domain with a path prefix.

Original address	Destination Address	Redirect Type	Country Code
https://mydomain.com	https://www.mydomain.com/documents	temporary redirect (302)	

JSON format

```
[
  {
    "source": "https://mydomain.com",
    "status": "302",
    "target": "https://www.mydomain.com/documents/",
    "condition": null
  }
]
```

You can use the following example 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 format

```
[
  {
    "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 initiating 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.

The following example uses regular expressions to set up a 200 rewrite for all files to index.html, except for the file extensions specified in the regular expression.

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

JSON format

```
[
  {
    "source": "</^[^.]+$|\.(?!(css|gif|ico|jpg|js|png|txt|svg|woff|woff2|ttf|map|json|
webp)$)([^.]+$)/>",
    "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 the user that the domain hasn't changed. HTTPS is the only protocol supported for reverse proxies.

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

JSON format

```
[
  {
    "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*). Amplify 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
/about/	/about/	/about/index.html

Placeholders

You can use the following example 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 format

```
[
  {
    "source":  "/docs/<year>/<month>/<date>/<itemid>",
    "status": "301",
    "target": "/documents/<year>/<month>/<date>/<itemid>",
               "condition": null
   }
]
```

Query strings and path parameters

Warning

Don't include secrets, credentials, or sensitive data in URLs as path or query parameters. These values are viewable in plain text in your Amplify application's access logs.

You can use the following example 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 format

```
[
  {
    "source": "/docs?id=<my-blog-id-value>",
    "status": "301",
    "target": "/documents/<my-blog-id-value>",
    "condition": null
  }
]
```

Note

Amplify forwards all query string parameters to the destination path for 301 and 302 redirects. However, if the original address includes a query string set to a specific value, as demonstrated in this example, Amplify doesn't forward query parameters. In this case, the redirect applies only to requests to the destination address with the specified query value `id`.

You can use the following example 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	not found (404)	

JSON format

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

Region-based redirects

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

Original address	Destination Address	Redirect Type	Country Code
/documents	/documents/us/	temporary redirect (302)	<US>

JSON format

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

Using wildcard expressions in redirects and rewrites

You can use the wildcard expression, <*>, in the original address for a redirect or rewrite. You must place the expression at the end of the original address, and it must be unique. Amplify ignores original addresses that include more than one wildcard expression, or use it in a different placement.

The following is an example of a valid redirect with a wildcard expression.

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

The following two examples demonstrate *invalid* redirects with wildcard expressions.

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

Using environment variables in an Amplify application

Environment variables are key-value pairs that you can add to your application's settings to make them available to Amplify Hosting. As a best practice, you can use environment variables to expose application configuration data. All environment variables that you add are encrypted to prevent rogue access.

Amplify enforces the following constraints on the environment variables that you create.

- Amplify doesn't allow you to create environment variable names with an `AWS` prefix. This prefix is reserved for Amplify internal use only.
- The value of an environment variable can't exceed 5500 characters.

> **Important**
>
> Don't use environment variables to store secrets. For a Gen 2 app, use the **Secret management** feature in the Amplify console. For more information, see Secrets and environment vars in the *Amplify Documentation*. For a Gen 1 app, store secrets in an environment secret created using the AWS Systems Manager Parameter Store. For more information, see Managing environment secrets.

Amplify environment variable reference

The following environment variables are accessible by default within the Amplify console.

Variable name	Description	Example value
_BUILD_TIMEOUT	The build timeout duration in minutes. The minimum value is 5. The maximum value is 120.	`30`
_LIVE_UPDATES	The tool will be upgraded to the latest version.	`[{"name":"Amplify CLI","pkg":"@aws-a`

Variable name	Description	Example value
		`mplify/cli","type":"npm","version":"latest"}]`
USER_DISABLE_TESTS	The test step is skipped during a build. You can disable tests for all branches or specific branches in an app. This environment variable is used for apps that perform tests during the build phase. For more information about setting this variable, see Turning off tests for an Amplify application or branch.	`true`
AWS_APP_ID	The app ID of the current build	`abcd1234`
AWS_BRANCH	The branch name of the current build	`main`, `develop`, `beta`, `v2.0`
AWS_BRANCH_ARN	The branch Amazon Resource Name (ARN) of the current build	`aws:arn:amplify:us-west-2:123456789012: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	`abcd1234`

Variable name	Description	Example value
AWS_JOB_ID	The job ID of the current build. This includes some padding of '0' so it always has the same length.	`0000000001`
AWS_PULL_REQUEST_ID	The pull request ID of pull request web preview build. This environment variable is not available when using AWS CodeCommit as your repository provider.	`1`
AWS_PULL_REQUEST_SOURCE_BRANCH	The name of the feature branch for a pull request preview being submitted to an application branch in the Amplify console.	`featureA`
AWS_PULL_REQUEST_DESTINATION_BRANCH	The name of the application branch in the Amplify console that a feature branch pull request is being submitted to.	`main`
AMPLIFY_AMAZON_CLIENT_ID	The Amazon client ID	`123456`
AMPLIFY_AMAZON_CLIENT_SECRET	The Amazon client secret	`example123456`
AMPLIFY_FACEBOOK_CLIENT_ID	The Facebook client ID	`123456`
AMPLIFY_FACEBOOK_CLIENT_SECRET	The Facebook client secret	`example123456`

Variable name	Description	Example value
AMPLIFY_GOOGLE_CLIENT_ID	The Google client ID	`123456`
AMPLIFY_GOOGLE_CLIENT_SECRET	The Google client secret	`example123456`
AMPLIFY_DIFF_DEPLOY	Enable or disable diff based frontend deployment. For more information, see Configuring diff based frontend build and deploy.	`true`
AMPLIFY_DIFF_DEPLOY_ROOT	The path to use for diff based frontend deployment comparisons, relative to the root of your repository.	`dist`
AMPLIFY_DIFF_BACKEND	Enable or disable diff based backend builds. This applies to Gen 1 apps only. For more information, see Configuring diff based backend builds for a Gen 1 app	`true`
AMPLIFY_BACKEND_PULL_ONLY	Amplify manages this environment variable. This applies to Gen 1 apps only. For more information, see Edit an existing frontend to point to a different backend	`true`

Variable name	Description	Example value
AMPLIFY_BACKEND_APP_ID	Amplify manages this environment variable. This applies to Gen 1 apps only. For more information, see Edit an existing frontend to point to a different backend	`abcd1234`
AMPLIFY_SKIP_BACKEND_BUILD	If you do not have a backend section in your build specification and want to disable backend builds, set this environment variable to `true`. This applies to Gen 1 apps only.	`true`
AMPLIFY_ENABLE_DEBUG_OUTPUT	Set this variable to `true` to print a stack trace in the logs. This is helpful for debugging backend build errors.	`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`
AMPLIFY_USERPOOL_ID	The ID for the Amazon Cognito user pool imported for auth	`us-west-2_example`

Variable name	Description	Example value
AMPLIFY_WEBCLIENT_ID	The ID for the app client to be used by web applications The app client must be configured with access to the Amazon Cognito user pool specified by the AMPLIFY_USERPOOL_ID environment variable.	`123456`
AMPLIFY_NATIVECLIENT_ID	The ID for the app client to be used by native applications The app client must be configured with access to the Amazon Cognito user pool specified by the AMPLIFY_USERPOOL_ID environment variable.	`123456`
AMPLIFY_IDENTITYPOOL_ID	The ID for the Amazon Cognito identity pool	`example-identitypool-id`
AMPLIFY_PERMISSIONS_BOUNDARY_ARN	The ARN for the IAM policy to use as a permissions boundary that applies to all IAM roles created by Amplify.	`arn:aws:iam::123456789012:policy/example-policy`
AMPLIFY_DESTRUCTIVE_UPDATES	Set this environment variable to true to allow a GraphQL API to be updated with schema operations that can potentially cause data loss.	`true`

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

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.

Setting environment variables

Use the following instructions to set environment variables for an application in the Amplify console.

> **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.

To set environment variables

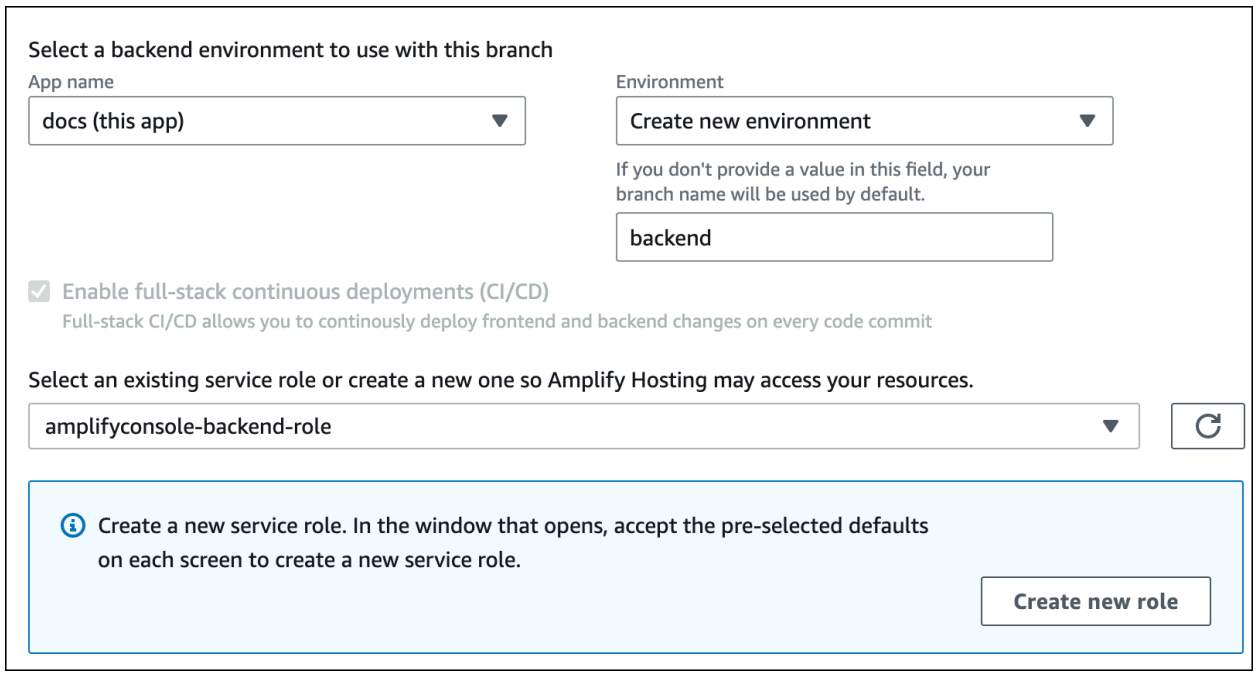
1. Sign in to the AWS Management Console and open the Amplify console.
2. In the Amplify console, choose **Hosting**, and then choose **Environment variables**.
3. On the **Environment variables** page, choose **Manage variables**.

4. For **Variable**, enter your key. For **Value**, enter your value. By default, Amplify 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**.

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. On the **Build settings** page, locate the **Select a backend environment to use with this branch** section. For **Environment**, choose **Create new environment**, and enter the name of your backend environment. The following screenshot shows the **Select a backend environment to use with this branch** section of the **Build settings** page with **backend** entered for the backend environment name.



b. Expand the **Advanced settings** section on the **Build settings** page 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 environment variable reference.

- **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, choose **Add variable**.

d. For **Variable** (key), enter your client ID. For **Value**, enter your client secret.

e. Choose, **Save**.

Managing environment secrets

With the release of Amplify Gen 2, the workflow for environment secrets is streamlined to centralize the management of secrets and environment variables in the Amplify console. For instructions on setting and accessing secrets for an Amplify Gen 2 app, see Secrets and environment vars in the *Amplify Documentation*.

Environment secrets for a Gen 1 app are similar to environment variables, but they are AWS Systems Manager 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.

Using AWS Systems Manager to set environment secrets for an Amplify Gen 1 application

Use the following instructions to set an environment secret for a Gen 1 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.

Accessing environment secrets for a Gen 1 application

Environment secrets for a Gen 1 application are stored in `process.env.secrets` as a JSON string.

Amplify environment secrets reference

Specify an Systems Manager parameter in the format `/amplify/{your_app_id}/{your_backend_environment_name}/AMPLIFY_SIWA_CLIENT_ID`.

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-----

Setting custom headers for an Amplify app

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 Amplify 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.

Previously, custom HTTP headers were specified for an app either by editing the build specification (buildspec) in the Amplify console or by downloading and updating the `amplify.yml` file and saving it in the project's root directory. We highly recommend migrating custom headers specified in this way out of the buildspec and the `amplify.yml` file. For instructions, see Migrating custom headers out of the build specification and amplify.yml.

Topics

- Custom header YAML reference
- Setting custom headers
- Migrating custom headers out of the build specification and amplify.yml
- Monorepo custom header requirements

Custom header YAML reference

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 to 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 Amplify app. You can specify headers in the Amplify 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 and save them in the 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 **Hosting**, then choose**Custom headers**.
4. On the **Custom headers** page, choose **Edit**.
5. In the **Edit custom headers** window, enter the information for your custom headers using the custom header YAML format.

   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. Redeploy the app to apply the new custom headers.
   - For a CI/CD app, navigate to the branch to deploy and choose **Redeploy this version**. You can also perform a new build from your Git repository.
   - For a manual deploy app, deploy the app again in the Amplify console.

To set custom headers for an app and save them in the root of your repository

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 **Hosting**, then choose **Custom headers**.
4. On the **Custom headers** page, choose **Download YML**.
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.

   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. Redeploy the 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 override custom headers defined in the **Custom headers** section in the Amplify console.

Security custom 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'"
```

Setting Cache-Control custom headers

Apps hosted with Amplify honor the `Cache-Control` headers that are sent by the origin, unless you override them with custom headers that you define. Amplify only applies Cache-Control custom headers for successful responses with a `200 OK` status code. This prevents error responses from being cached and served to other users that make the same request.

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).

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'
```

For more information about controlling application performance with headers, see Using the Cache-Control header to increase app performance.

Migrating custom headers out of the build specification and amplify.yml

Previously, custom HTTP headers were specified for an app either by editing the build specification in the Amplify 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 build specification and the `amplify.yml` file.

Specify your custom headers in the **Custom headers** section of the Amplify 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 **Hosting**, **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 **Hosting**, **Custom headers**.
8. On the **Custom headers** page, choose **Edit**.
9. In the **Edit custom headers** 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 **Hosting**, **Custom headers**.
7. On the **Custom headers** page, 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. Redeploy the 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 override the custom headers defined in the **Custom headers** section of the Amplify console.

Monorepo custom header requirements

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

- There is a specific YAML format for a monorepo. For the correct syntax, see Custom header YAML reference.
- You can specify custom headers for an application in a monorepo using the **Custom headers** section of the Amplify console. 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 override any custom headers specified using the **Custom headers** section of the Amplify console.

Managing the cache configuration for an app

Amplify uses Amazon CloudFront to manage the caching configuration for your hosted applications. A cache configuration is applied to each app to optimize for the best performance.

On August 13, 2024, Amplify released improvements to caching efficiency for applications. For more information, see CDN Caching Improvements for Better App Performance with AWS Amplify Hosting.

The following table summarizes Amplify support for specific caching behaviors before and after the caching improvements release.

Caching behavior	Previous support	With caching improvements
You can add custom headers for an app in the Amplify console or in a `customHeaders.yaml` file. One of the headers that you can override is `Cache-Control`. For more information, see Setting custom headers for an Amplify app.	Yes	Yes
Amplify respects the `Cache-Control` headers that you define in a `customHeaders.yaml` file and they take precedence over Amplify's default cache settings.	Yes	Yes
Amplify respects the `Cache-Control` headers set within an application's framework for dynamic routes (for example, Next.js SSR routes).	Yes	Yes

Caching behavior	Previous support	With caching improvements
If a `Cache-Control` header is set in the app's `customHeaders.yaml` file, this takes precedence over settings in the `next.config.js` file.		
Each new CI/CD app deployment clears the cache.	Yes	Yes
You can turn on performance mode for an app.	Yes	No The performance mode setting is no longer available in the Amplify console. However, you can create a `Cache-Control` header that sets the `s-maxage` directive. For instructions, see Using the Cache-Control header to increase app performance.

The following table lists the changes to the default values for specific cache settings.

Cache setting	Previous default value	Default value with caching improvements
Cache duration for static assets	Two seconds	One year
Cache duration for reverse proxy responses	Two seconds	Zero seconds (no caching)
Max Time to Live (TTL)	Ten minutes	One year

For more information about how Amplify determines the caching configuration to apply to an application and instructions on managing cache key configuration, see the following topics.

Topics

- How Amplify applies cache configuration to an app
- Managing cache key cookies
- Using the Cache-Control header to increase app performance

How Amplify applies cache configuration to an app

To manage caching for your app, Amplify determines the type of content that is being served by examining the app's platform type and rewrite rules. For `Compute` apps, Amplify also examines the routing rules in the deployment manifest.

> **Note**
>
> The app's platform type is set by Amplify Hosting during deployment. An SSG (static) app is set to the platform type `WEB`. An SSR app (Next.js 12 or later) is set to the platform type `WEB_COMPUTE`.

Amplify identifies the following four types of content and applies the specified managed cache policy.

Static

The content served from apps with the `WEB` platform, or the static routes in a `WEB_COMPUTE` app.

This content uses the Amplify-StaticContent cache policy.

Image Optimization

The images served by the `ImageOptimization` routes in a `WEB_COMPUTE` app.

This content uses the Amplify-ImageOptimization cache policy.

Compute

The content served by the `Compute` routes in a `WEB_COMPUTE` app. This includes all server-side rendered (SSR) content.

This content uses either the Amplify-Default or Amplify-DefaultNoCookies cache policy depending on the value of `cacheConfig.type` that is set on your Amplify App.

Reverse Proxy

The content served by paths that match a reverse proxy rewrite custom rule. For more information about creating this custom rule, see Reverse proxy rewrite in the *Using redirects* chapter.

This content uses either the Amplify-Default or Amplify-DefaultNoCookies cache policy depending on the value of `cacheConfig.type` that is set on your Amplify App.

Understanding Amplify's managed cache policies

Amplify uses the following predefined managed cache policies to optimize the default cache configuration for your hosted applications.

- Amplify-Default
- Amplify-DefaultNoCookies
- Amplify-ImageOptimization
- Amplify-StaticContent

Amplify-Default managed cache policy settings

View this policy in the CloudFront console

This policy is designed for use with an origin that is an AWS Amplify web app.

This policy has the following settings:

- **Minimum TTL:** 0 seconds
- **Maximum TTL:** 31536000 seconds (one year)
- **Default TTL:** 0 seconds
- **Headers included in cache key:**
  - `Authorization`
  - `Accept`
  - `CloudFront-Viewer-Country`

  - `Host`
- **Cookies included in cache key:** All cookies are included.
- **Query strings included in cache key:** All query strings are included.
- **Cache compressed objects setting:** Gzip and Brotli enabled.

Amplify-DefaultNoCookies managed cache policy settings

View this policy in the CloudFront console

This policy is designed for use with an origin that is an AWS Amplify web app.

This policy has the following settings:

- **Minimum TTL:** 0 seconds
- **Maximum TTL:** 31536000 seconds (one year)
- **Default TTL:** 0 seconds
- **Headers included in cache key:**
  - `Authorization`
  - `Accept`
  - `CloudFront-Viewer-Country`
  - `Host`
- **Cookies included in cache key:** No cookies are included.
- **Query strings included in cache key:** All query strings are included.
- **Cache compressed objects setting:** Gzip and Brotli enabled.

Amplify-ImageOptimization managed cache policy settings

View this policy in the CloudFront console

This policy is designed for use with an origin that is an AWS Amplify web app.

This policy has the following settings:

- **Minimum TTL:** 0 seconds
- **Maximum TTL:** 31536000 seconds (one year)

- **Default TTL:** 0 seconds
- **Headers included in cache key:**
  - `Authorization`
  - `Accept`
  - `Host`
- **Cookies included in cache key:** No cookies are included.
- **Query strings included in cache key:** All query strings are included.
- **Cache compressed objects setting:** Gzip and Brotli enabled.

Amplify-StaticContent managed cache policy settings

View this policy in the CloudFront console

This policy is designed for use with an origin that is an AWS Amplify web app.

This policy has the following settings:

- **Minimum TTL:** 0 seconds
- **Maximum TTL:** 31536000 seconds (one year)
- **Default TTL:** 0 seconds
- **Headers included in cache key:**
  - `Authorization`
  - `Host`
- **Cookies included in cache key:** No cookies are included.
- **Query strings included in cache key:** No query strings are included.
- **Cache compressed objects setting:** Gzip and Brotli enabled.

Managing cache key cookies

When you deploy your app to Amplify, you can choose whether you want to include or exclude cookies in the cache key. In the Amplify console, this setting is specified on the **Custom headers and cache** page using the **Cache key settings** toggle. For instructions, see Including or excluding cookies from the cache key.

Include cookies in the cache key

With this setting, Amplify automatically chooses an optimal cache configuration for your app based on the type of content that is being served. You must explicitly choose this cache configuration type.

If you are using the SDKs or the AWS CLI, this setting corresponds to setting `cacheConfig.type` to `AMPLIFY_MANAGED` with the `CreateApp` or `UpdateApp` APIs.

Exclude cookies from the cache key

This is the default cache configuration. This cache configuration is similar to the `AMPLIFY_MANAGED` configuration, except that it excludes all cookies from the cache key.

Choosing to exclude cookies from the cache key can result in better cache performance. However, before you choose this cache configuration, it is important to consider whether your app uses cookies to serve dynamic content.

If you are using the SDKs or the AWS CLI, this setting corresponds to setting the `cacheConfig.type` to `AMPLIFY_MANAGED_NO_COOKIES` with the `CreateApp` or `UpdateApp` APIs.

For more information about the cache key, see Understand the cache key in the *Amazon CloudFront Developer Guide;*.

Including or excluding cookies from the cache key

You can set the cache key cookie configuration for an app in the Amplify console, SDKs, or the AWS CLI.

Use the following procedure to specify whether to include or exclude cookies from the cache key when you are deploying a new app using the Amplify console.

To set the cache key cookie configuration when deploying an app to Amplify

1. Sign in to the AWS Management Console and open the Amplify console.
2. On the **All apps** page, choose **Create new app**.
3. On the **Start building with Amplify** page, choose your Git repository provider, then choose **Next**.

4. On the **Add repository branch** page, do the following:

   a. Select the name of the repository to connect.

   b. Select the name of the repository branch to connect.

   c. Choose **Next**.

5. If the app requires an IAM service role, you can either allow Amplify Hosting compute to automatically create a service role for you or you can specify a role that you have created.

   - To allow Amplify to automatically create a role and attach it to your app:

     - Choose **Create and use a new service role**.

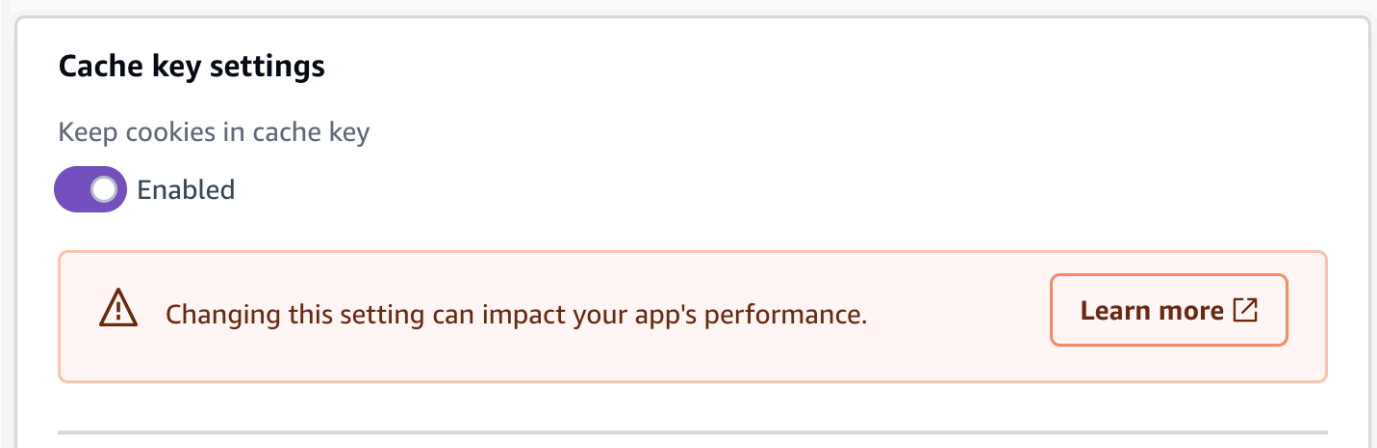
   - To attach a service role that you previously created:

     a. Choose **Use an existing service role**.

     b. Select the role to use from the list.

6. Choose **Advanced settings**, then locate the **Cache key settings** section.

7. Choose either **Keep cookies in cache key** or **Remove cookies from cache key**. The following screenshot shows the **Cache key settings** toggle in the console.



8. Choose **Next**.

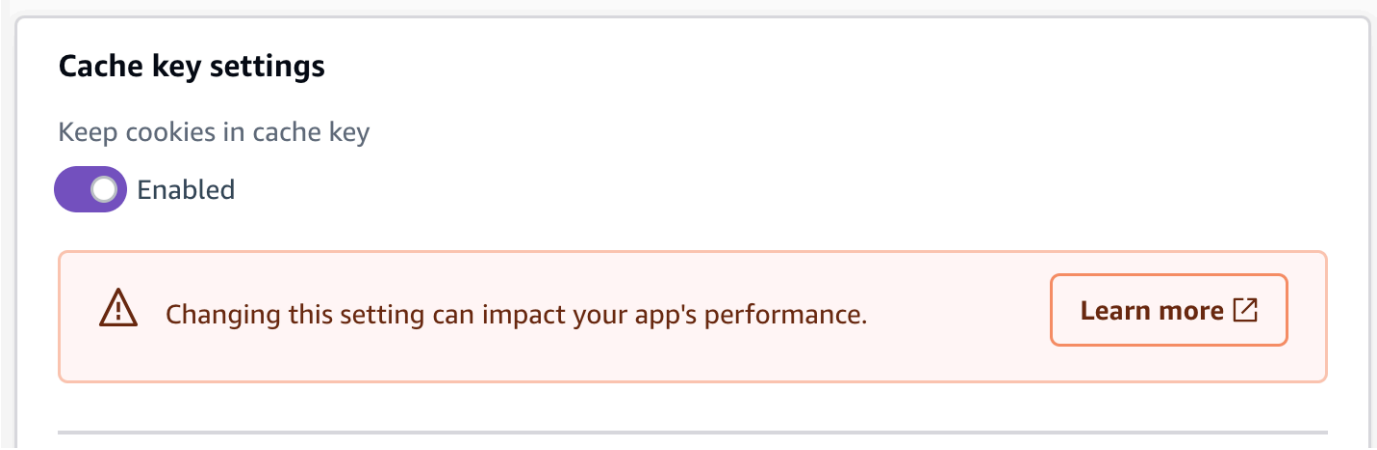
9. On the **Review** page, choose **Save and deploy**.

Changing the cache key cookie configuration for an app

You can change the cache key cookie configuration for an app that is already deployed to Amplify. Use the following procedure to change whether to include or exclude cookies from the cache key for an app using the Amplify console.

To change the cache key cookie configuration for a deployed app

1. Sign in to the AWS Management Console and open the Amplify console.
2. On the **All apps** page, choose the application you want to update.
3. In the navigation pane, choose **Hosting**, then choose **Custom headers and cache**.
4. On the **Custom headers and cache** page, locate the **Cache key settings** section and choose **Edit**.
5. Choose either **Keep cookies in cache key** or **Remove cookies from cache key**. The following screenshot shows the **Cache key settings** toggle in the console.



6. Choose **Save**.

Using the Cache-Control header to increase app performance

Amplify's default hosting architecture optimizes the balance between hosting performance and deployment availability. For most customers, we recommend that you use the default architecture.

If you require finer control over an app's performance, you can manually set the HTTP `Cache-Control` header to optimize for hosting performance by keeping content cached at the content delivery network (CDN) edge for a longer interval.

The HTTP `Cache-Control` header's `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.

Apps hosted with Amplify honor the `Cache-Control` headers that are sent by the origin, unless you override them with custom headers that you define. Amplify only applies `Cache-Control` custom headers for successful responses with a `200 OK` status code. This prevents error responses from being cached and served to other users that make the same request.

You can manually adjust the `s-maxage` directive to have more control over the performance and deployment availability of your app. For example, to change the length of time that your content stays cached at the edge, you can manually set the time to live (TTL) by updating `s-maxage` to a value other than the default 31536000 seconds (one year).

You can define custom headers for an app in the **Custom headers** section of the Amplify console. For an example of the YAML format, see Setting Cache-Control custom headers.

Use the following procedure to set the `s-maxage` directive to keep content cached at the CDN edge for 24 hours.

To set a custom Cache-Control header

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 **Hosting**, **Custom headers**.
4. On the **Custom headers** page, choose **Edit**.
5. In the **Edit custom headers** window, enter the information for your custom header as follows:
   a. For `pattern`, enter **`**/*`** for all paths.
   b. For `key`, enter **`Cache-Control`**.
   c. For `value`, enter **`s-maxage=86400`**.
6. Choose **Save**.
7. Redeploy the app to apply the new custom header.

Skew protection for Amplify deployments

Deployment skew protection is available to Amplify applications to eliminate version skew issues between client and servers in web applications. When you apply skew protection to an Amplify application, you can ensure that your clients always interact with the correct version of server-side assets, regardless of when a deployment occurs.

Version skew is a common challenge for web developers. It occurs when a web browser is running an outdated version of an application and the server is running a new one. This discrepancy can cause unpredictable behavior, errors, and a degraded experience for the user of the application. The Amplify deployment skew protection feature pins clients running on web browsers to a specific deployment. This ensures that Amplify always serves the assets for that particular deployment, keeping the client and server synchronized.

Amplify's skew protection feature can reduce errors for your application's users as you release new deployments. It can also improve the developer experience by reducing the time spent managing backward and forward compatibility issues.

Skew protection feature details:

Supported application types

You can add skew protection to static and SSR applications created with any framework that Amplify supports. Applications can be deployed from a Git repository or a manual deployment.

You can't add skew protection to an application that is deployed to the `WEB_DYNAMIC` platform (Next.js version 11 or earlier).

Duration

For static applications, Amplify serves one week of deployments. For SSR applications, we guarantee skew protection for up to eight previous deployments.

Cost

There is no additional cost for adding skew protection to an application.

Performance consideration

When skew protection is enabled for an application, Amplify must update its CDN cache configurations. Therefore, you should expect your first deployment after enabling skew protection to take up to ten minutes.

Topics

- Configuring deployment skew protection for an Amplify application
- How skew protection works

Configuring deployment skew protection for an Amplify application

You can add or remove deployment skew protection for an application using the Amplify console, the AWS Command Line Interface, or the SDKs. The feature is applied at the branch level. Only new deployments, that are made after skew protection is enabled for a branch, will be skew protected.

To add or remove deployment skew protection using the AWS CLI or SDKs, use the `CreateBranch.enableSkewProtection` and `UpdateBranch.enableSkewProtection` fields. For more information, see CreateBranch and UpdateBranch in the *Amplify API reference* documentation.

If you want to remove a specific deployment so that it no longer gets served, use the `DeleteJob` API. For more information, see DeleteJob in the *Amplify API reference* documentation.

At this time, you can only enable skew protection on an application that is already deployed to Amplify Hosting. Use the following instructions to add skew protection to a branch using the Amplify console.

Enable skew protection for branch of an Amplify application

1. Sign in to the AWS Management Console and open the Amplify console at https://console.aws.amazon.com/amplify/.
2. On the **All apps** page, choose the name of the deployed app to enable skew protection on.
3. In the navigation pane, choose **App settings**, then choose **Branch settings**.
4. In the **Branches** section, choose the name of the branch to update.
5. On the **Actions** menu, choose **Enable skew protection**.
6. In the confirmation window, choose **Confirm**. Skew protection is now enabled for the branch.
7. Redeploy your application branch. Only deployments that are made after skew protection is enabled are skew protected.

Use the following instructions to remove skew protection from a branch of an application using the Amplify console.

Remove skew protection from a branch of an Amplify application

1. Sign in to the AWS Management Console and open the Amplify console at https://console.aws.amazon.com/amplify/.
2. On the **All apps** page, choose the name of the deployed app to remove skew protection from.
3. In the navigation pane, choose **App settings**, then choose **Branch settings**.
4. In the **Branches** section, choose the name of the branch to update.
5. On the **Actions** menu, choose **Disable skew protection**. Skew protection is now disabled for the branch and only the latest content will be served.

How skew protection works

In most cases, the default behavior of the _dpl cookie will serve your skew protection needs. However, in the following advanced scenarios, skew protection is better enabled using the `X-Amplify-Dpl` header and `dpl` query parameter.

- Loading your website in multiple browser tabs at the same time
- Using service workers

Amplify evaluates the incoming request in the following order when determining the content to serve to the client:

1. **`X-Amplify-Dpl` header** – Applications can use this header to direct requests to a specific Amplify deployment. This request header can be set by using the value of `process.env.AWS_AMPLIFY_DEPLOYMENT_ID`.
2. **`dpl` query parameter** – Next.js applications will automatically set the _dpl query parameter for requests to fingerprinted assets (.js and .css files).
3. **_dpl cookie** – This is the default for all skew protected applications. For a specific browser, the same cookie is sent for every browser tab or instance that interacts with a domain.

   Be aware that if different browser tabs have different versions of a website loaded, the _dpl cookie is shared by all of the tabs. In this scenario, it isn't possible to achieve total skew

protection with the _dpl cookie and you should consider using the `X-Amplify-Dpl` header for skew protection.

X-Amplify-Dpl header example

The following example demonstrates the code for a Next.js SSR page that accesses skew protection through the `X-Amplify-Dpl` header. The page renders its content based on one of its api routes. The deployment to serve to the api route is specified by using the `X-Amplify-Dpl` header, that is set to the value of `process.env.AWS_AMPLIFY_DEPLOYMENT_ID`.

```
import { useEffect, useState } from 'react';

export default function MyPage({deploymentId}) {
    const [data, setData] = useState(null);

    useEffect(() => {
        fetch('/api/hello', {
            headers: {
                'X-Amplify-Dpl': process.env.AWS_AMPLIFY_DEPLOYMENT_ID
            },
        })
        .then(res => res.json())
        .then(data => setData(data))
        .catch(error => console.error("error", error))
    }, []);

    return <div>
        {data ? JSON.stringify(data) : "Loading ... " }
    </div>
}
```

Monitoring an Amplify application

AWS Amplify provides the following features for monitoring your hosted applications:

- **CloudWatch metrics** – Amplify emits metrics through Amazon CloudWatch that you can use to monitor traffic, errors, data transfer, and latency for your applications.
- **Access logs** – Amplify provides access logs with detailed information about requests made to your application.
- **CloudTrail logging** – Amplify is integrated with AWS CloudTrail which provides a record of actions taken by a user, role, or an AWS service in Amplify. You can view these events in the CloudTrail console.

Topics

- Monitoring an Amplify application with Amazon CloudWatch
- Retrieving and analyzing access logs for an Amplify application
- Logging Amplify API calls using AWS CloudTrail

Monitoring an Amplify application with Amazon CloudWatch

AWS Amplify is integrated with Amazon CloudWatch, enabling you to monitor metrics for your Amplify applications in near real-time, and 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.

Supported CloudWatch metrics

Amplify supports seven CloudWatch metrics in the `AWS/AmplifyHosting` namespace for monitoring traffic, errors, data transfer, latency, and request tokens 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.

The following table describes each supported metric and lists the most relevant statistics. Not all available statistics are applicable for every metric.

Metric	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 for any request, including headers. Amplify doesn't charge you for data uploaded in your applications. 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.
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 Hosting receives a request and when it returns a

Metric	Description
	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.

Metric	Description
TokensConsumed	The request tokens consumed by your app. The `Sum` statistic represents total request token consumption. You can compare this statistic to your current `Request tokens per second` service quota to determine whether you need to request a quota increase to avoid potential throttling during a future high traffic event. The `Average` statistic represents request token consumption across normal and peak times. Higher token consumption typically leads to longer time to first byte (TTFB). Therefore, you can use this statistic when evaluating your application's latency. If your latency is poor, you can improve your downstream APIs to reduce your token consumption and avoid the throttling that can occur when token consumption exceeds your application's `Request tokens per second` service quota. For more information about the `Request tokens per second` service quota, see Amplify Hosting service quotas.

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.

Accessing CloudWatch metrics

You can access CloudWatch metrics directly from the Amplify console using the following procedure.

> **Note**
>
> You can also access CloudWatch metrics in the AWS Management Console at https://console.aws.amazon.com/cloudwatch/.

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 **Monitoring**, then choose **Metrics**.

Creating CloudWatch 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 **Monitoring**, then choose **Alarms**.
4. On the **Alarms** page, choose **Create alarm**.
5. 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 invoke the alarm.
      iv. Specify the length of time of the evaluation period.

   e. Choose **Confirm**.

> **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.

Accessing CloudWatch Logs for SSR apps

Amplify sends information about your SSR runtime to Amazon CloudWatch Logs in your AWS account. When you deploy an SSR app to Amplify Hosting compute, the app requires an IAM service role that Amplify assumes when calling other services on your behalf. You can either allow Amplify Hosting compute to automatically create a service role for you or you can specify a role that you have created.

If you choose to allow Amplify to create an IAM role for you, the role will already have the permissions to create CloudWatch Logs. If you create your own IAM role, you will need to add the following permissions to your policy to allow Amplify to access Amazon CloudWatch Logs.

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

For more information about adding a service role, see Adding a service role with permissions to deploy backend resources. For more information about deploying server-side rendered apps, see Deploying server-side rendered applications with Amplify Hosting.

You can view the Amplify Hosting compute logs for an SSR application in the CloudWatch console or in the Amplify console. Use the following instructions to view the logs in the Amplify console.

To view CloudWatch logs for an SSR application in the Amplify console

1. Sign in to the AWS Management Console and open the Amplify console.
2. Choose the SSR app to view the CloudWatch logs for.
3. In the navigation pane, choose **Monitoring**, then choose **Hosting compute logs**.
4. On the **Hosting compute logs** page, search and select a CloudWatch log group for a specific branch.

Retrieving and analyzing access logs for an Amplify application

Amplify stores access logs for all of the apps you host in Amplify. Access logs contain information about requests that are made to your hosted apps. Amplify retains all access logs for an app until you delete the app. All access logs for an app are available in the Amplify console. However, each individual request for access logs is limited to a two week time period that you specify.

> **Warning**
>
> Don’t include secrets, credentials, or sensitive data in URLs as path or query parameters. These values are viewable in plain text in your Amplify application’s access logs.

Amplify never reuses CloudFront distributions between customers. Amplify creates CloudFront distributions in advance so that you don't have to wait for a CloudFront distribution to be created when you deploy a new app. Before these distributions are assigned to an Amplify app, they might receive traffic from bots. However, they're configured to always respond as *Not found* before they're assigned. If your app's access logs contain entries for a time period before you created your app, these entries are related to this activity.

> **Important**
>
> We recommend that you use the logs to understand the nature of the requests for your content, not as a complete accounting of all requests. Amplify delivers access logs on a best-effort basis. The log entry for a particular request might be delivered long after the request was actually processed and, in rare cases, a log entry might not be delivered at all. When a log entry is omitted from access logs, the number of entries in the access logs won't match the usage that appears in the AWS billing and usage reports.

Retrieving an app's access logs

Use the following procedure to retrieve access logs for an Amplify app.

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 **Monitoring**, then choose **Access logs**.
4. Choose **Edit time range**.
5. In the **Edit time range** window do the following.
    a. For **Start date**, specify the first day of the two week interval to retrieve logs for.
    b. For **Start time**, choose the time on the first day to start the log retrieval.

c. Choose **Confirm**.

6. The Amplify 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 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;
```

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, the AWS Amplify Admin UI API Reference, and the Amplify UI Builder 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 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"
}
```

Using IAM roles with Amplify applications

An IAM role is an IAM identity with specific permissions. The role's permissions determine what the identity can and cannot do in AWS. You can create IAM roles in your AWS account and use them to delegate permissions to Amplify Hosting. To learn more about roles, see IAM roles in the *IAM User Guide*.

You can use the following types of IAM roles to grant Amplify Hosting the permissions it needs to perform actions on your behalf or run compute code that accesses other AWS resources.

IAM service role

Amplify assumes this role to perform actions on your behalf. This role is required for applications with backend resources.

IAM SSR Compute role

Allows a server-side rendered (SSR) application to securely access specific AWS resources.

IAM SSR CloudWatch Logs role

When you deploy an SSR app, the app requires an IAM service role that Amplify assumes to allow Amplify to access Amazon CloudWatch Logs.

Topics

- Adding a service role with permissions to deploy backend resources
- Adding an SSR Compute role to allow access to AWS resources
- Adding a service role with permissions to access CloudWatch Logs

Adding a service role with permissions to deploy backend resources

Amplify 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 provides Amplify Hosting with permissions to deploy, create, and manage backends on your behalf.

When you are creating a new app that requires an IAM service role, you can either allow Amplify Hosting to automatically create a service role for you or you can select an IAM role that you have

already created. In this section, you will learn how to create an Amplify service role that has account administrative permissions and explicity allows direct access to resources that Amplify applications require to deploy, create, and manage backends.

Creating an Amplify service role in the IAM console

To create a service role

1. Open the IAM console and choose **Roles** from the left navigation bar, then choose **Create role**.
2. On the **Select trusted entity** page, choose **AWS service**. For **Use case**, select **Amplify - Backend Deployment**, then choose **Next**.
3. On the **Add permissions** page, choose **Next**.
4. On the **Name, view, and create** page, for **Role name** enter a meaningful name, such as **`AmplifyConsoleServiceRole-AmplifyRole`**.
5. Accept all the defaults and choose **Create role**.
6. Return to the Amplify console to attach the role to your app.
   - If you are in the process of deploying a new app, do the following:
     a. Refresh the list of service roles.
     b. Select the role you just created. For this example, it should look like **AmplifyConsoleServiceRole-AmplifyRole**.
     c. Choose **Next** and follow the steps to complete your app deployment.
   - If you have an existing app, do the following:
     a. In the navigation pane, choose **App settings**, then choose **IAM roles**.
     b. On the **IAM roles** page, in the **Service role** section, choose **Edit**.
     c. On the **Service role** page, select the role you just created from the **Service role** list.
     d. Choose **Save**.
7. Amplify now has permissions to deploy backend resources for your app.

Editing a service role's trust policy to prevent confused deputy

The confused deputy problem is a security issue where an entity that doesn't have permission to perform an action can coerce a more-privileged entity to perform the action. For more information, see Cross-service confused deputy prevention.

Currently, the default trust policy for the `Amplify-Backend Deployment` service role enforces the `aws:SourceArn` and `aws:SourceAccount` global context condition keys to prevent against confused deputy. However, if you previously created an `Amplify-Backend Deployment` role in your account, you can update the role's trust policy to add these conditions to protect against confused deputy.

Use the following example to restrict access to apps in your account. Replace the Region and application ID in the example with your own information.

```
"Condition": {
      "ArnLike": {
        "aws:SourceArn": "arn:aws:amplify:us-east-1:123456789012:apps/*"
      },
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      }
    }
```

For instructions on editing the trust policy for a role using the AWS Management Console, see Modifying a role (console) in the *IAM User Guide*.

Adding an SSR Compute role to allow access to AWS resources

This integration, enables you to assign an IAM role to the Amplify SSR Compute service to allow your server-side rendered (SSR) application to securely access specific AWS resources based on the role's permissions. For example, you can allow your app's SSR compute functions to securely access other AWS services or resources, such as Amazon Bedrock or an Amazon S3 bucket, based on the permissions defined in the assigned IAM role.

The IAM SSR Compute role provides temporary credentials, eliminating the need to hardcode long-lived security credentials in environment variables. Using the IAM SSR Compute role aligns with the AWS security best practices of granting least-privilege permissions and using short-term credentials when possible.

The instructions later in this section describe how to create a policy with custom permissions and attach the policy to a role. When you create the role, you must attach a custom trust policy that gives Amplify permission to assume the role. If the trust relationship isn't defined correctly, you will get an error when you try to add the role. The following custom trust policy grants Amplify permission to assume the role.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Statement1",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "amplify.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

You can associate an IAM role in your AWS account with an existing SSR application using the Amplify console, AWS SDKs, or the AWS CLI. The role that you attach is automatically associated with the Amplify SSR compute service, granting it the permissions that you specify to access other AWS resources. As your application's needs change over time, you can modify the attached IAM role without redeploying your application. This provides flexibility and reduces application downtime.

Important

You are responsible for configuring your application to meet your security and compliance objectives. This includes managing your SSR Compute role, which should be configured to have the minimum set of permissions needed to support your use case. For more information, see Managing IAM SSR Compute role security.

Creating an SSR Compute role in the IAM console

Before you can attach an IAM SSR Compute role to an Amplify application, the role must already exist in your AWS account. In this section, you will learn how to create an IAM policy and attach it to a role that Amplify can assume to access specific AWS resources.

We recommend that you follow the AWS best practice of granting *least-privilege* permissions when creating an IAM role. The IAM SSR Compute role is called only from SSR compute functions and therefore should only grant the permissions required to run the code.

You can use the AWS Management Console, AWS CLI, or SDKs to create policies in IAM. For more interformation, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

The following instructions demonstrate how to use the IAM console to create an IAM policy that defines the permissions to grant to the Amplify Compute service.

To use the IAM console JSON policy editor to create a policy

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane on the left, choose **Policies**.
3. Choose **Create policy**.
4. In the **Policy editor** section, choose the **JSON** option.
5. Type or paste a JSON policy document.
6. When you are finished adding permissions to the policy, choose **Next**.
7. On the **Review and create** page, type a **Policy Name** and a **Description** (optional) for the policy that you are creating. Review **Permissions defined in this policy** to see the permissions that are granted by your policy.
8. Choose **Create policy** to save your new policy.

After you create a policy, use the following instructions to attach the policy to an IAM role.

To create a role that grants Amplify permissions to specific AWS resources

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane of the console, choose **Roles** and then choose **Create role**.
3. Choose the **Custom trust policy** role type.
4. In the **Custom trust policy** section, enter the custom trust policy for the role. A role trust policy is required and defines the principals that you trust to assume the role.

Copy and paste the following trust policy to grant the Amplify service permission to assume this role.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Statement1",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "amplify.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

5. Resolve any security warnings, errors, or general warnings generated during policy validation, and then choose **Next**.
6. On the **Add permissions** page, search for the name of the policy that you created in the previous procedure and select it. Then choose **Next**.
7. For **Role name**, enter a role name. Role names must be unique within your AWS account. They are not distinguished by case. For example, you cannot create roles named both **PRODROLE** and **prodrole**. Because other AWS resources might reference the role, you cannot edit the name of the role after it has been created.
8. (Optional) For **Description**, enter a description for the new role.
9. (Optional) Choose **Edit** in the **Step 1: Select trusted entities** or **Step 2: Add permissions** sections to edit the custom policy and permissions for the role.
10. Review the role and then choose **Create role**.

Adding an IAM SSR Compute role to an Amplify app

After you have created an IAM role in your AWS account, you can associate it with an app in the Amplify console.

To add an SSR Compute role to an app in the Amplify console

1. Sign in to the AWS Management Console and open the Amplify console at https://console.aws.amazon.com/amplify/.
2. On the **All apps** page, choose the name of the app to add a Compute role to.
3. In the navigation pane, choose **App settings**, and then choose **IAM roles**.
4. In the **Compute role** section, choose **Edit**.
5. In the **Default role** list, search for the name of the role you want to attach and select it. For this example, you can choose the name of the role you created in the previous procedure. By default, the role that you select will be associated with all branches of your app.

   If the role's trust relationship isn't defined correctly, you will get an error and you won't be able to add the role.
6. (optional) If your application is in a public repository and uses auto-branch creation or has web previews for pull requests enabled, we don't recommend using an app-level role. Instead, attach the Compute role only to branches that require access to specific resources. To override the default app-level behavior and attach a role to a specific branch, do the following:

   a. For **Branch**, select the name of the branch to use.

   b. For **Compute role**, select the name of the role to associate with the branch.
7. Choose, **Save**.

Managing IAM SSR Compute role security

Security is a shared responsibility between AWS and you. You are responsible for configuring your application to meet your security and compliance objectives. This includes managing your SSR Compute role, which should be configured to have the minimum set of permissions needed to support your use case. Credentials for the SSR Compute role that you specify are immediately available in the runtime of your SSR function. If your SSR code exposes these credentials, either intentionally, due to a bug, or by permitting remote code execution (RCE), an unauthorized user can gain access to the SSR role and its permissions.

When an application in a public repository uses an SSR Compute role and auto-branch creation or web previews for pull requests, you need to carefully manage which branches can access the role. We recommend that you don't use an app-level role. Instead, you should attach a Compute role at the branch-level. This allows you to grant permissions only to the branches that require access to specific resources.

If your role's credentials are exposed, take the following actions to remove all access to the role's credentials.

1. **Revoke all sessions**

   For instructions on immediately revoking all permissions to the role's credentials, see Revoke IAM role temporary security credentials.

2. **Delete the role from the Amplify console**

   This action takes immediate effect. You don't need to redeploy your application.

To delete a Compute role in the Amplify console

1. Sign in to the AWS Management Console and open the Amplify console at https://console.aws.amazon.com/amplify/.
2. On the **All apps** page, choose the name of the app to remove the Compute role from.
3. In the navigation pane, choose **App settings**, and then choose **IAM roles**.
4. In the **Compute role** section, choose **Edit**.
5. To delete the **Default role**, choose the **X** to the right of the role's name.
6. Choose **Save**.

Adding a service role with permissions to access CloudWatch Logs

Amplify sends information about your SSR runtime to Amazon CloudWatch Logs in your AWS account. When you deploy an SSR app, the app requires an IAM service role that Amplify assumes when calling other services on your behalf. You can either allow Amplify Hosting compute to automatically create a service role for you or you can specify a role that you have created.

If you choose to allow Amplify to create an IAM role for you, the role will already have the permissions to create CloudWatch Logs. If you create your own IAM role, you will need to add the following permissions to your policy to allow Amplify to access Amazon CloudWatch Logs.

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

Unified webhooks for Git repositories

Amplify Hosting uses webhooks to automatically initiate a build after a new commit to your Git repository. The unified webhooks feature improves Amplify's integrations with Git providers and enables you to connect more Amplify applications to a single repository. With unified webhooks, Amplify now uses a single webhook per Region for all associated applications in your repository. For example, if your repository is connected to applications in both the US East (N. Virginia) and US West (Oregon) Regions, you will have two unified webhooks.

Before this release, Amplify created a new webhook for each app associated with a repository. If you had multiple apps in a single repository, you could reach the webhook limits enforced by individual Git providers and be prevented from adding more apps. This was especially challenging for teams working in monorepos, where multiple projects exist in a single repository.

Unified webhooks provide the following benefits:

- **Overcome Git provider webhook limits**: You can connect as many Amplify apps as you need to a single repository.
- **Enhanced monorepo support**: You have more flexibility and efficiency when working with monorepos, where multiple projects share a single repository.
- **Simplified management**: Managing multiple Amplify apps with a single repository webhook reduces complexity and potential points of failure.
- **Improved workflow integration**: You can use the webhooks allocated by your Git provider for other essential workflows in your development process.

Getting started with unified webhooks

Creating a new app

When you deploy a new application to Amplify Hosting from a Git repository, the unified webhooks feature is automatically implemented for your repository. For instructions on creating a new application, see Getting started with deploying an app to Amplify Hosting.

Updating an existing app

For existing Amplify applications, you must reconnect your Git repository to your application to replace the existing webhooks with a unified webhook. If you've already reached the maximum

number of webhooks allowed by your Git provider, migrating to the unified webhook might not succeed. In this case, manually remove at least one existing webhook before reconnecting.

You can have multiple applications in a repository that are deployed to different AWS Regions. Since Amplify operations are Region based, the migration to a unified webhook only occurs for the webhooks in the Region where you reconnected your Amplify app. As a result, you might see both application id-based webhooks and Region-based unified webhooks in your repository.

Use the following instructions to migrate an existing Amplify app to a unified webhook.

To migrate an existing Amplify app to a unified webhook

1. Sign in to the AWS Management Console and open the Amplify console.
2. Choose the app that you want to migrate to a unified webhook.
3. In the navigation pane, choose **App settings**, then choose **Branch settings**.
4. On the **Branch settings** page, choose **Reconnect repository**.
5. To verify successful migration to the unified webhook, navigate to the webhook settings in your Git repository. You should see a single webhook URL in the format `https://amplify-webhooks.`*`Region`*`.amazonaws.com/`*`git-provider`*.

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
- Data Protection in Amplify
- Compliance Validation for AWS Amplify
- Infrastructure Security in AWS Amplify
- Security event logging and monitoring in Amplify
- Cross-service confused deputy prevention
- Security best practices for Amplify

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
- Authenticating with identities
- Managing access using policies
- How Amplify works with IAM
- Identity-based policy examples for Amplify
- AWS managed policies for AWS Amplify
- Troubleshooting Amplify identity and access

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.

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 service users 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.

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.

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be *authenticated* (signed in to AWS) as the AWS account root user, as an IAM user, or by assuming an IAM role.

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (IAM Identity Center) users, your company's single sign-on authentication, and your Google or Facebook credentials are examples of federated identities. When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see AWS Signature Version 4 for API requests in the *IAM User Guide*.

Regardless of the authentication method that you use, you might 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 Multi-factor authentication in the *AWS IAM Identity Center User Guide* and AWS Multi-factor authentication in IAM in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one 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 don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see Tasks that require root user credentials in the *IAM User Guide*.

Federated identity

As a best practice, require human users, including users that require administrator access, to use federation with an identity provider to access AWS services by using temporary credentials.

A *federated identity* is a user from your enterprise user directory, a web identity provider, the AWS Directory Service, the Identity Center directory, or any user that accesses AWS services by using credentials provided through an identity source. When federated identities access AWS accounts, they assume roles, and the roles provide temporary credentials.

For centralized access management, we recommend that you use AWS IAM Identity Center. You can create users and groups in IAM Identity Center, or you can connect and synchronize to a set of users and groups in your own identity source for use across all your AWS accounts and applications. For information about IAM Identity Center, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see Rotate access keys regularly for use cases that require long-term credentials in the *IAM User Guide*.

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 Use cases for IAM users 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. To temporarily assume an IAM role in the AWS Management Console, you can switch from a user to an IAM role (console). 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 Methods to assume a role in the *IAM User Guide*.

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

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see Create a role for a third-party identity provider (federation) in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see Permission sets in the *AWS IAM Identity Center User Guide*.
- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **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 Cross account resource access in IAM 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.
  - **Forward access sessions (FAS)** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.
  - **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 Create 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 AWS 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 Use an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. 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**.

By default, users and roles have no permissions. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

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 Define custom IAM permissions with customer managed 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 Choose 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 an 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 Service control policies in the *AWS Organizations User Guide*.
- **Resource control policies (RCPs)** – RCPs are JSON policies that you can use to set the maximum available permissions for resources in your accounts without updating the IAM policies attached to each resource that you own. The RCP limits permissions for resources in member accounts and can impact the effective permissions for identities, including the AWS account root user, regardless of whether they belong to your organization. For more information about Organizations and RCPs, including a list of AWS services that support RCPs, see Resource control policies (RCPs) 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	Yes
Resource-based policies	No
Policy actions	Yes
Policy resources	Yes
Policy condition keys	Yes
ACLs	No
ABAC (tags in policies)	Partial
Temporary credentials	Yes
Forward access sessions (FAS)	Yes
Service roles	Yes
Service-linked roles	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 Define custom IAM permissions with customer managed 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.

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 Cross account resource access in IAM 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.

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.

Policy condition keys for Amplify

Supports service-specific 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.

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.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see Define permissions with ABAC authorization 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 Switch from a user to an IAM 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.

Forward access sessions for Amplify

Supports forward access sessions (FAS): Yes

When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.

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 Create 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 AWS 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, users and roles don't have permission to create or modify Amplify resources. They also can't perform tasks by using the AWS Management Console, AWS Command Line Interface (AWS CLI), or AWS API. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see Create IAM policies (console) in the *IAM User Guide*.

For details about actions and resource types defined by Amplify, including the format of the ARNs for each of the resource types, see Actions, resources, and condition keys for AWS Amplify in the *Service Authorization Reference*.

Topics

- Policy best practices
- Using the Amplify console
- Allow users to view their own permissions

Policy best practices

Identity-based policies 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 with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We

recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.

- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as AWS CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see Validate policies with IAM Access Analyzer in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Secure API access with MFA in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM 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 (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 they're trying to perform.

With the release of Amplify Studio, 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, 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

An AWS managed policy is a standalone policy that is created and administered by AWS. AWS managed policies are designed to provide permissions for many common use cases so that you can start assigning permissions to users, groups, and roles.

Keep in mind that AWS managed policies might not grant least-privilege permissions for your specific use cases because they're available for all AWS customers to use. We recommend that you reduce permissions further by defining customer managed policies that are specific to your use cases.

You cannot change the permissions defined in AWS managed policies. If AWS updates the permissions defined in an AWS managed policy, the update affects all principal identities (users, groups, and roles) that the policy is attached to. AWS is most likely to update an AWS managed policy when a new AWS service is launched or new API operations become available for existing services.

For more information, see AWS managed policies in the *IAM User Guide*.

AWS managed policy: AdministratorAccess-Amplify

You can attach the `AdministratorAccess-Amplify` policy to your IAM identities. 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 explicitly 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.

This policy grants the `iam:PassRole` action permission for all resources. This is required to support Amazon Cognito user pools configuration.

To view the permissions for this policy, see AdministratorAccess-Amplify in the *AWS Managed Policy Reference*.

AWS managed policy: AmplifyBackendDeployFullAccess

You can attach the `AmplifyBackendDeployFullAccess` policy to your IAM identities.

This policy grants Amplify full access permissions to deploy Amplify backend resources using the AWS Cloud Development Kit (AWS CDK). Permissions are deferred to the AWS CDK roles that have the necessary `AdministratorAccess` policy permissions.

Permissions details

This policy includes permissions to do the following .

- `Amplify`– Retrieve metadata about deployed applications.
- `AWS CloudFormation`– Create, update, and delete Amplify managed stacks.
- `SSM`– Create, update, and delete Amplify managed SSM Parameter Store `String` and `SecureString` parameters.
- `AWS AppSync`– Update and retrieve AWS AppSync schema, resolver and function resources. The purpose is to support the Gen 2 sandbox hotswapping functionality.
- `Lambda`– Update and retrieve the configuration for Amplify managed functions. The purpose is to support the Gen 2 sandbox hotswapping functionality.

  Retrieve a Lambda function's tags. The purpose is to support Lambda functions defined by customers.
- `Amazon S3`– Retrieve Amplify deployment assets.
- `AWS Security Token Service`– Enables the AWS Cloud Development Kit (AWS CDK) CLI to assume the deployment role.

- `Amazon RDS`– Read metadata of DB instances, clusters, and proxies.
- `Amazon EC2`– Read the availability zone information for a subnet.
- `CloudWatch Logs`– Retrieve the logs for a customer's Lambda function. The purpose is to allow an Amplify cloud development sandbox environment to stream a Lambda function's logs to a customer's terminal.

To view the permissions for this policy, see AmplifyBackendDeployFullAccess in the *AWS Managed Policy Reference*.

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 page.

Change	Description	Date
AmplifyBackendDeployFullAccess – Update to an existing policy	Add read access to the `logs:FilterLogEvents` resource to allow Amplify to stream logs from functions where a custom log group was created. This is an extension of the existing ability to stream a Lambda function's logs.	November 14, 2024
AmplifyBackendDeployFullAccess – Update to an existing policy	Add read access to the `lambda:ListTags` and `logs:FilterLogEvents` resources to support Lambda functions defined by customers. These permissions allow an Amplify cloud development sandbox environment to stream a	July 18, 2024

Change	Description	Date
	Lambda function's logs to a customer's terminal.	
AmplifyBackendDeployFullAccess – Update to an existing policy	Add read access to the `arn:aws:ssm:*:*:parameter/cdk-bootstrap/*` resource to allow Amplify to detect the CDK bootstrap version in a customer's account.	May 31, 2024

Change	Description	Date
AmplifyBackendDeployFullAccess – Update to an existing policy	Add a new `AmplifyDiscoverRDSVpcConfig` policy statement with Amazon RDS and Amazon EC2 read-only permissions scoped by both resource and account conditions. These permissions support the Amplify Gen 2 `npx amplify generate schema-from-database` command that allows customers to generate Typescript data schema from an existing SQL database. Add the `rds:DescribeDBProxies`, `rds:DescribeDBInstances`, `rds:DescribeDBClusters`, `rds:DescribeDBSubnetGroups`, and `ec2:DescribeSubnets` permissions. The `npx amplify generate schema-from-database` command requires these permissions to check whether a specified DB host is hosted in Amazon RDS and auto-generate the Amazon VPC configuration required to provision the other resources required to	April 17, 2024

Change	Description	Date
	set up an AWS AppSync API backed by a SQL database.	
AmplifyBackendDeployFullAccess – Update to an existing policy	Add the `cloudformation:DeleteStack` policy action to support stack deletion when the `DeleteBranch` API is called. Add the `lambda:GetFunction` policy action to support hotswapping functions. Add the `lambda:UpdateFunctionConfiguration` policy action to support updates to the Lambda function.	April 5, 2024
AdministratorAccess-Amplify – Update to an existing policy	Add the `cloudformation:TagResource` and `cloudformation:UnTagResource` permissions to support calls to AWS CloudFormation APIs.	April 4, 2024

Change	Description	Date
AmplifyBackendDeployFullAccess – Update to an existing policy	Add the `lambda:InvokeFunction` policy action to support AWS Cloud Development Kit (AWS CDK) hotswapping. The AWS CDK makes direct calls to a Lambda function to perform Amazon S3 asset hotswapping. Add the `lambda:UpdateFunctionCode` policy action to support hotswapping functions.	January 02, 2024
AmplifyBackendDeployFullAccess – Update to an existing policy	Add policy actions to support the `UpdateApiKey` operation. This is required to enable a successful app deployment after exiting and restarting the sandbox without deleting resources.	November 17, 2023
AmplifyBackendDeployFullAccess – Update to an existing policy	Add the `amplify:GetBackendEnvironment` permission to support Amplify app deployment.	November 6, 2023
AmplifyBackendDeployFullAccess – New policy	Amplify added a new policy with the minimum permissions required to deploy Amplify backend resources.	October 8, 2023

Change	Description	Date
AdministratorAccess-Amplify – Update to an existing policy	Add the `ecr:DescribeRepositories` permission that is required by the Amplify Command Line Interface (CLI).	June 1, 2023

Change	Description	Date
AdministratorAccess-Amplify – Update to an existing policy	Add a policy action to support removing tags from an AWS AppSync resource. Add a policy action to support the Amazon Polly resource. Add a policy action to support updating the OpenSearch domain configuration. Add a policy action to support removing tags from an AWS Identity and Access Management role. Add a policy action to support removing tags from an Amazon DynamoDB resource. Add the `cloudfront:GetCloudFrontOriginAccessIdentity` and `cloudfront:GetCloudFrontOriginAccessIdentityConfig` permissions to the `CLISDKCalls` statement block to support the Amplify publish and hosting workflows. Add the `s3:PutBucketPublicAccessBlock` permission to the `CLIManageviaCFNPol`	February 24, 2023

Change	Description	Date
	`icy` statement block to allow the AWS CLI to support the Amazon S3 security best practice of enabling the Amazon S3 Block Public Access feature on internal buckets. Add the `cloudformation:DescribeStacks` permission to the `CLISDKCalls` statement block to support retrieving customers' AWS CloudFormation stacks on retries in the Amplify backend processor to avoid duplicating executions if a stack is updating. Add the `cloudformation:ListStacks` permission to the `CLICloudformationPolicy` statement block. This permission is required to fully support the CloudFormation DescribeStacks action.	
AdministratorAccess-Amplify – Update to an existing policy	Add policy actions to allow the Amplify server-side rendering feature to push application metrics to CloudWatch in a customer's AWS account.	August 30, 2022

Change	Description	Date
AdministratorAccess-Amplify – Update to an existing policy	Add policy actions to block public access to the Amplify deployment Amazon S3 bucket.	April 27, 2022
AdministratorAccess-Amplify – Update to an existing policy	Add an action to allow customers to delete their server-side rendered (SSR) apps. This also allows the corresponding CloudFront distribution to be deleted successfully. Add an action to allow customers to specify a different Lambda function to handle events from an existing event source using the Amplify CLI. With these changes, AWS Lambda will be able to perform the UpdateEventSourceMapping action.	April 17, 2022
AdministratorAccess-Amplify – Update to an existing policy	Add a policy action to enable Amplify UI Builder actions on all resources.	December 2, 2021

Change	Description	Date
AdministratorAccess-Amplify – Update to an existing policy	Add policy actions to support the Amazon Cognito authentication feature that uses social identity providers. Add a policy action to support Lambda layers. Add a policy action to support the Amplify Storage category.	November 8, 2021

Change	Description	Date
AdministratorAccess-Amplify – Update to an existing policy	Add Amazon Lex actions to support the Amplify Interactions category. Add Amazon Rekognition actions to support the Amplify Predictions category. Add an Amazon Cognito action to support MFA configuration on Amazon Cognito user pools. Add CloudFormation actions to support AWS CloudFormation StackSets. Add Amazon Location Service actions to support the Amplify Geo category. Add a Lambda action to support Lambda layers in Amplify. Add CloudWatch Logs actions to support CloudWatch Events. Add Amazon S3 actions to support the Amplify Storage category. Add policy actions to support server-side rendered (SSR) apps.	September 27, 2021

Change	Description	Date
AdministratorAccess-Amplify – 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 originati on phone numbers, and viewing, creating, verifying , and deleting destination phone numbers. Amplify Studio: Add Amazon Cognito, AWS Lambda, IAM, and AWS CloudFormation policy actions to enable managing backends in the Amplify console and Amplify Studio. Add an AWS Systems Manager (SSM) policy statement to manage Amplify environment secrets. Add an AWS CloudFormation `ListResources` action to	July 28, 2021

Change	Description	Date
	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
- I am not authorized to perform iam:PassRole
- I want to allow people outside of my AWS account to access my Amplify resources

I am not authorized to perform an action in Amplify

If you receive an error that you're not authorized to perform an action, your policies must be updated to allow you to perform the action.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a fictional *`my-example-widget`* resource but doesn't have the fictional `amplify:`*`GetWidget`* permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 amplify:GetWidget on resource: my-example-widget
```

In this case, the policy for the `mateojackson` user must be updated to allow access to the *`my-example-widget`* resource by using the `amplify:`*`GetWidget`* action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

With the release of Amplify Studio, 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, you 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, your policies must be updated 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 that are 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's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

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.
- 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 Cross account resource access in IAM in the *IAM 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 users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way 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 Hosting 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 Hosting endpoints use SHA-256 certificates that are managed by AWS Private Certificate Authority. For more information, see Signature Version 4 signing process and What is AWS Private Certificate Authority.

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 an AWS service is within the scope of specific compliance programs, see AWS services in Scope by Compliance Program and choose the compliance program that you are interested in. 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 Compliance & Governance – These solution implementation guides discuss architectural considerations and provide steps for deploying security and compliance features.
- HIPAA Eligible Services Reference – Lists HIPAA eligible services. Not all AWS services are HIPAA eligible.
- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- AWS Customer Compliance Guides – Understand the shared responsibility model through the lens of compliance. The guides summarize the best practices for securing AWS services and map the guidance to security controls across multiple frameworks (including National Institute of Standards and Technology (NIST), Payment Card Industry Security Standards Council (PCI), and International Organization for Standardization (ISO)).
- 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. Security Hub uses security controls to evaluate your AWS resources and to check your compliance against security industry standards and best practices. For a list of supported services and controls, see Security Hub controls reference.
- Amazon GuardDuty – This AWS service detects potential threats to your AWS accounts, workloads, containers, and data by monitoring your environment for suspicious and malicious

activities. GuardDuty can help you address various compliance requirements, like PCI DSS, by meeting intrusion detection requirements mandated by certain compliance frameworks.

- 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 AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well-Architected Framework*.

You use AWS published API calls to access Amplify through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). 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.

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 an Amplify application.

- *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.
- *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.

Cross-service confused deputy prevention

The confused deputy problem is a security issue where an entity that doesn't have permission to perform an action can coerce a more-privileged entity to perform the action. In AWS, cross-service impersonation can result in the confused deputy problem. Cross-service impersonation can occur when one service (the *calling service*) calls another service (the *called service*). The calling service can be manipulated to use its permissions to act on another customer's resources in a way it should not otherwise have permission to access. To prevent this, AWS provides tools that help you protect your data for all services with service principals that have been given access to resources in your account.

We recommend using the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in resource policies to limit the permissions that AWS Amplify gives another service to the resource. If you use both global condition context keys, the `aws:SourceAccount` value and the account in the `aws:SourceArn` value must use the same account ID when used in the same policy statement.

The value of `aws:SourceArn` must be the branch ARN of the Amplify app. Specify this value in the format `arn:`*`Partition`*`:amplify:`*`Region`*`:`*`Account`*`:apps/`*`AppId`*`/branches/`*`BranchName`*.

The most effective way to protect against the confused deputy problem is to use the `aws:SourceArn` global condition context key with the full ARN of the resource. If you don't know the full ARN of the resource or if you are specifying multiple resources, use the `aws:SourceArn` global context condition key with wildcards (*) for the unknown portions of the ARN. For example, `arn:aws:`*`servicename`*`::`*`123456789012`*`:*`.

The following example shows a role trust policy you can apply to limit access to any Amplify app in your account and prevent the confused deputy problem. To use this policy, replace the red italicized text in the example policy with your own information.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Sid": "ConfusedDeputyPreventionExamplePolicy",
    "Effect": "Allow",
    "Principal": {
      "Service": [
          "amplify.me-south-1.amazonaws.com",
          "amplify.eu-south-1.amazonaws.com",
          "amplify.ap-east-1.amazonaws.com",
          "amplifybackend.amazonaws.com",
          "amplify.amazonaws.com"
        ]
    },
    "Action": "sts:AssumeRole",
    "Condition": {
      "ArnLike": {
        "aws:SourceArn": "arn:aws:amplify:us-east-1:123456789012:apps/*"
      },
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      }
    }
  }
}
```

The following example shows a role trust policy you can apply to limit access to a specified Amplify app in your account and prevent the confused deputy problem. To use this policy, replace the red italicized text in the example policy with your own information.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Sid": "ConfusedDeputyPreventionExamplePolicy",
    "Effect": "Allow",
    "Principal": {
      "Service": [
          "amplify.me-south-1.amazonaws.com",
          "amplify.eu-south-1.amazonaws.com",
          "amplify.ap-east-1.amazonaws.com",
          "amplifybackend.amazonaws.com",
          "amplify.amazonaws.com"
        ]
    },
    "Action": "sts:AssumeRole",
    "Condition": {
      "ArnLike": {
        "aws:SourceArn": "arn:aws:amplify:us-east-1:123456789012:apps/d123456789/
branches/*"
      },
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      }
    }
  }
}
```

Security best practices for Amplify

Amplify provides a number of security features to consider as you develop and implement your own security policies. The following best practices are general guidelines and don't represent a complete security solution. Because these best practices might not be appropriate or sufficient for your environment, treat them as helpful recommendations rather than prescriptions.

Using cookies with the Amplify default domain

When you use Amplify to deploy a web app, Amplify hosts it for you on the default `amplifyapp.com` domain. You can view your app on a URL formatted as `https://branch-name.d1m7bkiki6tdw1.amplifyapp.com`.

To augment the security of your Amplify applications, the *amplifyapp.com* domain is registered in the Public Suffix List (PSL). For further security, we recommend that you use cookies with a `__Host-` prefix if you ever need to set sensitive cookies in the default domain name for your Amplify applications. This practice will help to defend your domain against cross-site request forgery attempts (CSRF). For more information see the Set-Cookie page in the Mozilla Developer Network.

Amplify Hosting service quotas

The following are the service quotas for AWS Amplify Hosting. Service quotas (previously referred to as *limits*) are the maximum number of service resources or operations for your AWS account.

New AWS accounts have reduced apps and concurrent jobs quotas. AWS raises these quotas automatically based on your usage. You can also request a quota increase.

The Service Quotas console provides information about the quotas for your account. You can use the Service Quotas console to view default quotas and request quota increases for adjustable quotas. For more information, see Requesting a quota increase in the *Service Quotas User Guide*.

Name	Default	Adjustable	Description
Apps	Each supported Region: 25	Yes	The maximum number of apps that you can create in AWS Amplify Console in this account in the current Region.
Branches per app	Each supported Region: 50	No	The maximum number of branches per app that you can create in this account in the current Region.
Build artifact size	Each supported Region: 5 Gigabytes	No	The maximum size (in GB) of an app build artifact. A build artifact is deployed by AWS Amplify Console after a build.
Cache artifact size	Each supported Region: 5 Gigabytes	No	The maximum size (in GB) of a cache artifact.

Name	Default	Adjustable	Description
Concurrent jobs	Each supported Region: 5	Yes	The maximum number of concurrent jobs that you can create in this account in the current Region.
Domains per app	Each supported Region: 5	Yes	The maximum number of domains per app that you can create in this account in the current Region.
Environment cache artifact size	Each supported Region: 5 Gigabytes	No	The maximum size (in GB) of the environment cache artifact.
Manual deploy ZIP file size	Each supported Region: 5 Gigabytes	No	The maximum size (in GB) of a manual deploy ZIP file.
Maximum app creations per hour	Each supported Region: 25	No	The maximum number of apps that you can create in AWS Amplify Console per hour in this account in the current Region.
Request tokens per second	Each supported Region: 20,000	Yes	The maximum number of request tokens per second for an app. Amplify Hosting allocates tokens to requests based on the amount of resources (processing time and data transfer) that they consume.

Name	Default	Adjustable	Description
Subdomains per domain	Each supported Region: 50	No	The maximum number of subdomains per domain that you can create in this account in the current Region.
Webhooks per app	Each supported Region: 50	Yes	The maximum number of webhooks per app that you can create in this account in the current Region.

For more information about Amplify service quotas, see AWS Amplify endpoints and quotas in the *AWS General Reference*.

Troubleshooting Amplify Hosting

If you encounter errors or deployment issues when working with Amplify Hosting, consult the topics in this section.

Topics

- Troubleshooting general Amplify issues
- Troubleshooting Amazon Linux 2023 build image issues
- Troubleshooting build issues
- Troubleshooting custom domains
- Troubleshooting server-side rendered applications
- Troubleshooting redirects and rewrites
- Troubleshooting caching
- Setting up Amplify access to GitHub repositories

Troubleshooting general Amplify issues

The following information can help you troubleshoot general issues with Amplify Hosting.

Topics

- HTTP 429 status code (Too many requests)
- The Amplify console doesn't display the build status and last update time for my app
- Web previews are not being created for new pull requests
- My manual deployment is stuck with a pending status in the Amplify console
- I need to update my application's Node.js version

HTTP 429 status code (Too many requests)

Amplify controls the number of requests per second (RPS) to your website based on the processing time and data transfer that incoming requests consume. If your application returns an HTTP 429 status code, incoming requests are exceeding the amount of processing time and data transfer allotted to your application. This application limit is managed by Amplify's

REQUEST_TOKENS_PER_SECOND service quota. For more information about quotas, see Amplify Hosting service quotas.

To fix this issue, we recommend optimizing your application to reduce request duration and data transfer to increase the app's RPS. For example, with the same 20,000 tokens, a highly optimized SSR page that responds within 100 milliseconds can support higher RPS as compared to a page with latency higher than 200 milliseconds.

Similarly, an application that returns a 1 MB response size will consume more tokens than an application that returns a 250 KB response size.

We also recommend that you leverage the Amazon CloudFront cache by configuring Cache-Control headers that maximize the time that a given response is kept in the cache. Requests that are served from the CloudFront cache don't count towards the rate limit. Each CloudFront distribution can handle up to 250,000 requests per second, enabling you to scale your app very high using the cache. For more information about the CloudFront cache, see Optimizing caching and availability in the *Amazon CloudFront Developer Guide*.

The Amplify console doesn't display the build status and last update time for my app

When you navigate to the **All apps** page in the Amplify console, a tile is displayed for each of your apps in the current Region. If you don't see the build status, such as **Deployed**, and the **Last update** time displayed for an app, the app doesn't have a `Production` stage branch associated with it.

To list the apps in the console, Amplify uses the `ListApps` API. Amplify uses the `ProductionBranch.status` attribute to display the build status and the `ProductionBranch.lastDeployTime` attribute to display the last update time. For more information about this API, see ProductionBranch in the *Amplify Hosting API documentation*.

Use the following instructions to associate a `Production` stage to your app's branch.

1. Sign in to the Amplify console.
2. On the **All apps** page, choose the app that you want to update.
3. In the navigation pane choose **App settings**, then **Branch settings**.
4. In the **Branch settings** section, choose **Edit**.
5. For **Production branch**, choose the branch name that you want to use.

6. Choose **Save**.
7. Return to the **All apps** page. The build status and last update time should now be displayed for your app.

Web previews are not being created for new pull requests

The web previews feature enables you to preview changes from pull requests before merging them into an integration branch. A web preview deploys every pull request made to your repository to a unique preview URL which is different from the URL that your main site uses.

If you have turned on web previews for your app, but they aren't being created for new PRs, investigate whether one of the following is the cause of your issue.

1. Check to see whether your app has reached the maximum `Branches per app` service quota. For more information about quotas, see Amplify Hosting service quotas.

   To stay within the default quota of 50 branches per app, consider enabling auto branch deletion in your app. This will prevent you from accumulating branches in your account that no longer exist in your repository.
2. If you are using a public GitHub repository and your Amplify app has an IAM service role attached to it, Amplify doesn't create previews for security reasons. For example, apps with backends and apps that are deployed to the `WEB_COMPUTE` hosting platform require an IAM service role. Therefore, you can't enable web previews for these types of apps if their repository is public.

   To enable web previews to work for your app, you can either disassociate the service role (if the app doesn't have a backend or isn't a `WEB_COMPUTE` app), or you can make the GitHub repository private.

My manual deployment is stuck with a pending status in the Amplify console

Manual deployments enable you to publish your web app with Amplify Hosting without connecting a Git provider. You can use one of the following four deployment options.

1. Drag and drop your application folder in the Amplify console.
2. Drag and drop a .zip file (that contains the build artifacts of your site) in the Amplify console.

3. Upload a .zip file (that contains the build artifacts of your site) to an Amazon S3 bucket and connect the bucket to an app in the Amplify console.
4. Use a public URL that points to a .zip file (that contains the build artifacts of your site) in the Amplify console.

We are aware of issues with the drag a drop functionality when using an application folder for a manual deployment in the Amplify console. These deployments can fail for the following reasons.

- Transient network issues occur.
- There is a local change to the files during upload.
- The browser session attempts to upload a large amount of static assets simultaneously.

While we work on improving the reliability of our drag and drop uploads, we recommend that you use a .zip file instead of dragging and dropping the application folders.

We highly recommend uploading a .zip file to an Amazon S3 bucket, as this avoids file uploads from the Amplify console and provides a higher reliability for manual deployments. Amplify's integration with Amazon S3 simplifies this process. For more information, see Deploying a static website to Amplify from an Amazon S3 bucket.

I need to update my application's Node.js version

Amplify support for apps using Node.js versions 16 and 18 ends on September 15, 2025. Applications that are already deployed will continue running. However, after this date, you won't be able to deploy updates to your application until you upgrade to Node.js version 20 or later.

If you are using the Amazon Linux 2023 build image, Node.js version 20 is supported by default. Starting on September 15, 2025, the AL2023 image will automatically support Node.js 22 and change its default Node.js version from 18 to 22.

Amazon Linux 2 (AL2) doesn't automatically support Node.js version 20 or later. If you are currently using AL2, we recommend that you switch to AL2023. You can change the build image in the Amplify console. You can also use a custom build image that supports the Node.js version that you specify.

Prior to upgrading, we recommend that you test your application on a new branch to verify that it functions correctly.

Upgrade options

Amplify console

You can use the live package updates feature in the Amplify console to specify the version of Node.js to use. For instructions, see Using specific package and dependency versions in the build image.

Custom build image

If you are using a custom build image and NVM is installed on your image, you can add `nvm install 20` to your Dockerfile. To learn more about the requirements and configuration instructions for a custom build image, see Customizing the build image.

Build settings

You can specify the Node.js version to use in your application's `amplify.yml` build settings, by adding the `nvm use` command to the preBuild commands section. For instructions on updating an application's build settings, see Configuring the build settings for an Amplify application.

The following example demonstrates how to customize the build settings to set the default Node.js version to Node.js 18 and upgrade to Node.js version 20 on a test branch named `node-20`.

```
frontend:
  phases:
    preBuild:
      commands:
        - nvm use 18
        - if [ "${AWS_BRANCH}" = "node-20" ]; then nvm use 20; fi
```

Warning

Be aware that `preBuild` commands run after live package updates. The Node.js version specified by the `nvm use` command will override the Node.js version set by live package updates.

Troubleshooting Amazon Linux 2023 build image issues

The following information can help you troubleshoot issues with the Amazon Linux 2023 (AL2023) build image.

Topics

- I want to run Amplify functions with the Python runtime
- I want to run commands that require superuser or root privileges

I want to run Amplify functions with the Python runtime

Amplify Hosting now uses the Amazon Linux 2023 build image by default when you deploy a new application. AL2023 comes pre-installed with Python versions 3.8, 3.9, 3.10, and 3.11.

For backwards compatibility with the Amazon Linux 2 image, the AL2023 build image has symlinks for older versions of Python pre-installed.

By default, Python version 3.10 is used globally. To build your functions using a specific Python version, run the following commands in your application's build specification file.

```
version: 1
backend:
  phases:
    build:
      commands:
        # use a python version globally
        - pyenv global 3.11
        # verify python version
        - python --version
        # install pipenv
        - pip install --user pipenv
        # add to path
        - export PATH=$PATH:/root/.local/bin
        # verify pipenv version
        - pipenv --version
        - amplifyPush --simple
```

I want to run commands that require superuser or root privileges

If you are using the Amazon Linux 2023 build image and get an error when running system commands that require superuser or root privileges, you must run these commands using the Linux `sudo` command. For example, if you get an error running `yum install -y gcc`, use `sudo yum install -y gcc`.

The Amazon Linux 2 build image used the root user, but Amplify's AL2023 image runs your code with a custom `amplify` user. Amplify grants this user privileges to run commands using the Linux `sudo` command. It is a best practice to use `sudo` for commands that require superuser privileges.

Troubleshooting build issues

If you encounter issues when creating or building an Amplify application, consult the topics in this section for help.

Topics

- New commits to my repository aren't triggering Amplify builds
- My repository name isn't listed in the Amplify console when creating a new application
- My build fails with the Cannot find module aws-exports error (Gen 1 apps only)
- I want to override a build timeout

New commits to my repository aren't triggering Amplify builds

If new commits to your Git repository aren't triggering Amplify builds, verify that your webhook is still present on your repository. If it's present, check the history of webhook requests to see if there are any failures. Amplify has a payload size limit of 256 KB for incoming webhooks. If you push a commit to your repository that has a large number of changed files, you might exceed this limit and cause builds to not be triggered.

My repository name isn't listed in the Amplify console when creating a new application

When you create a new application in the Amplify console, you can choose from your organization's available repositories on the **Add repository and branch** page. Your target repository might not be displayed in the list if it hasn't been recently updated. This might occur if

your organization has a large number of repositories. To resolve this issue, push a commit to the repository, then refresh the repository list in the console. This should cause the repository to be displayed.

My build fails with the `Cannot find module aws-exports` error (Gen 1 apps only)

If your app can't find the `aws-exports.js` file during a build, the following error is returned.

```
TS2307: Cannot find module 'aws-exports'
```

The Amplify command line interface (CLI) generates the `aws-exports.js` file during your backend build. To resolve this error, you must create an `aws-exports.js` file for use in the build. Add the following code to your build specification to create the file:

```
backend:
  phases:
    build:
      commands:
        - "# Execute Amplify CLI with the helper script"
        - amplifyPush --simple
```

For a full example of the build specification settings for an Amplify app, see Build specification YAML syntax reference.

I want to override a build timeout

The default build timeout is 30 minutes. You can override the default build timeout using the `_BUILD_TIMEOUT` environment variable. The minimum build timeout is 5 minutes. The maximum build timeout is 120 minutes.

For instructions on setting an environment variable for an app in the Amplify console, see Setting environment variables.

Troubleshooting custom domains

If you encounter issues when connecting a custom domain to your Amplify application, consult the topics in this section for help.

If you don't see a solution to your issue here, contact Support. For more information, see Creating a support case in the *AWS Support User Guide*.

Topics

- I need to verify that my CNAME resolves
- My domain hosted with a third-party is stuck in the Pending Verification state
- My domain hosted with Amazon Route 53 is stuck in the Pending Verification state
- My app with multi-level subdomains is stuck in the Pending Verification state
- My DNS provider doesn't support A records with fully qualified domain names
- I get a CNAMEAlreadyExistsException error
- I get an Additional Verification Required error
- I get a 404 error on the CloudFront URL
- I get SSL certificate or HTTPS errors when visiting my domain

I need to 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.

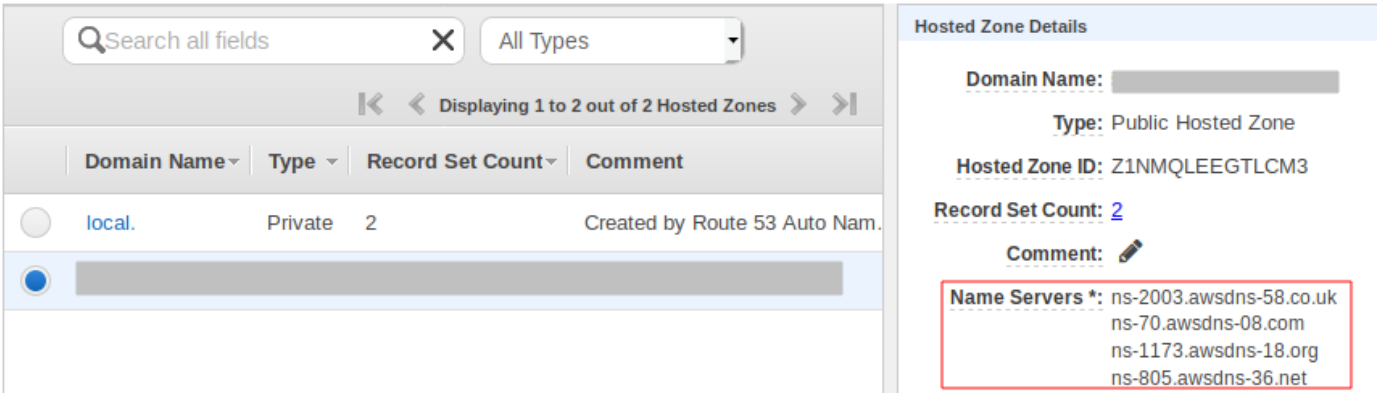
3. If you have verified that the CNAME record exists, then there may be an issue with your DNS provider. You can either contact the DNS provider to diagnose why the DNS verification CNAME is not resolving or you can migrate your DNS to Route 53. For more information, see Making Amazon Route 53 the DNS service for an existing domain.

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

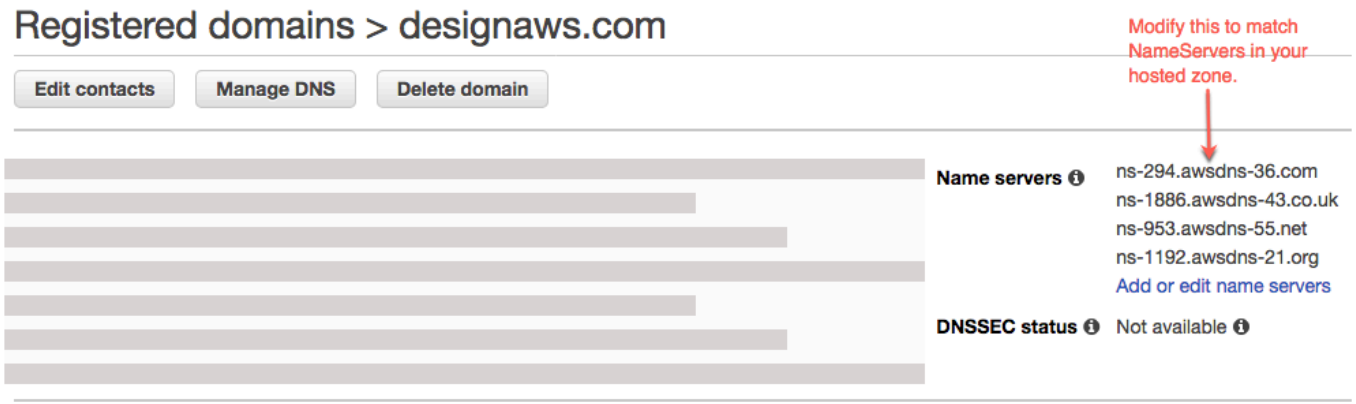
If you transferred your domain to Amazon Route 53, it is possible that your domain has different name servers than those issued by Amplify when your app was created. Perform the following steps to diagnose the cause of the error.

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, contact Support. For more information, see Creating a support case in the *AWS Support User Guide*.

My app with multi-level subdomains is stuck in the Pending Verification state

If an app with multi-level subdomains is stuck in the **Pending Verification** state when connecting to a third-party DNS provider, there might be an issue with the format of your DNS records. Some DNS providers automatically add the second-level domain (SLD) and top-level domain (TLD)

domain suffixes to your records. If you are also specifying the domain in the format that includes the SLD and TLD, this can cause a domain verification issue.

When you connect a domain, first try specifying the domain name using the full format provided by Amplify, for example `_hash.docs.backend.example.com`. If the SSL configuration gets stuck in the **Pending Verification** state, try removing the TLD and SLD from the records. For example, if the full format is `_hash.docs.backend.example.com`, specify `_hash.docs.backend`. Wait 15 to 30 minutes to allow the records to propagate. Then use a tool such as MX Toolbox to check whether the verification process is working.

My DNS provider doesn't support A records with fully qualified domain names

Some DNS providers don't support A records with a fully qualified domain name (FQDN), such as `example.cloudfront.net`. For example, Cloudflare A records can only write IPv4 addresses and don't support FQDNs. To work around this limitation, we recommend using CNAME records instead of A records in your DNS configuration.

As an example, the following DNS configuration uses an A record.

```
A     | @ | ***.cloudfront.net
CNAME | www | ***.cloudfront.net
```

Change it to the following DNS configuration to use CNAME records only.

```
CNAME | @ | ***.cloudfront.net
CNAME | www | ***.cloudfront.net
```

This workaround enables you to properly point your apex domain (@ record) to services like CloudFront, while avoiding the IPv4-only limitation of A records in Cloudflare's system.

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. The source of your error depends on your current hosting and DNS providers.

A CNAME alias, such as `example.com` or `sub.example.com` can only be associated with a single CloudFront distribution at a time. The **CNAMEAlreadyExistsException** indicates that your domain

is already associated with another CloudFront distribution, either within the same AWS account, or potentially in a different account. The domain must be disassociated from the previous CloudFront distribution before the new distribution created by Amplify Hosting will work. You might need to check more than one account if you or your organization owns multiple AWS accounts.

Perform the following steps to diagnose the cause of the **CNAMEAlreadyExistsException** error.

1. Sign in to the Amazon CloudFront console and verify that you don't have this domain deployed to another 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 name of the distribution to edit.
    c. Choose the **General** tab. In the **Settings** section, choose **Edit**.
    d. Remove the domain name from **Alternate domain name (CNAME)**. Then choose, **Save changes**.
3. Confirm that no other CloudFront distribution exists that is using this domain in the current AWS account or other AWS accounts. If it won't disrupt any currently running services, try deleting and recreating the hosted zone.
4. 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`.
5. 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, contact Support. For more information, see Creating a support case in the *AWS Support User Guide*.
6. If you manage your domain through Route 53, make sure to clean up any hosted zone CNAME or ALIAS records that point to the old CloudFront distribution.
7. After completing the preceding steps, remove the custom domain from Amplify Hosting and start over with the workflow to connect a custom domain in the Amplify console.

I get an Additional Verification Required error

If you get an **Additional Verification Required** error, this means that AWS Certificate Manager (ACM) requires additional information to process this certificate request. This can happen as a fraud-protection measure, such as when the domain ranks within the Alexa top 1000 websites. To provide the required information, use the Support Center to contact Support. If you don't have a support plan, post a new thread in the ACM Discussion Forum.

> **Note**
>
> You cannot request a certificate for Amazon-owned domain names such as those ending in amazonaws.com, cloudfront.net, or elasticbeanstalk.com.

I get a 404 error on the CloudFront URL

To serve traffic, Amplify Hosting points to a CloudFront URL via a CNAME record. In the process of connecting an app to a custom domain, the Amplify console displays the CloudFront URL for the app. However, you cannot access your application directly using this CloudFront URL. It returns a 404 error. Your application resolves only using the Amplify app URL (for example, `https://main.d5udybEXAMPLE.amplifyapp.com`, or your custom domain (for example `www.example.com`).

Amplify needs to route requests to the correct deployed branch and uses the hostname to do this. For example, you can configure the domain `www.example.com` that points to the mainline branch of an app, but also configure `dev.example.com` that points to the dev branch of the same app. Therefore, you must visit your application based on it's configured subdomains so that Amplify can route the requests accordingly.

I get SSL certificate or HTTPS errors when visiting my domain

If you have Certificate Authority Authorization (CAA) DNS records configured with your third-party DNS provider, AWS Certificate Manager (ACM) might not be able to update or reissue intermediate certificates for your custom domain SSL certificate. To resolve this, you need to add a CAA record to trust at least one of Amazon's certificate authority domains. The following procedure describes the steps you need to perform.

To add a CAA record to trust an Amazon certificate authority

1. Configure a CAA record with your domain provider to trust at least one of Amazon's certificate authority domains. For more information about configuring the CAA record, see Certification Authority Authorization (CAA) problems in the *AWS Certificate Manager User Guide*.
2. Use one of the following methods to update your SSL certificate:
   - Manually update using the Amplify console.

     > **Note**
     >
     > This method will cause down time for your custom domain.

     a. Sign in to the AWS Management Console and open the Amplify console.
     b. Choose your app that you want to add a CAA record to.
     c. In the navigation pane, choose **App Settings**, **Domain management**.
     d. On the **Domain management** page, delete the custom domain.
     e. Connect your app to the custom domain again. This process issues a new SSL certificate and its intermediate certificates can now be managed by ACM.

        To reconnect your app to your custom domain, use one of the following procedures that corresponds to the domain provider you are using.

        - Adding a custom domain managed by Amazon Route 53.
        - Adding a custom domain managed by a third-party DNS provider.
        - Updating DNS records for a domain managed by GoDaddy.
   - Contact Support to have your SSL certificate reissued.

Troubleshooting server-side rendered applications

If you experience unexpected issues when deploying an SSR app with Amplify Hosting compute, review the following troubleshooting topics. If you don't see a solution to your issue here, see the SSR web compute troubleshooting guide in the Amplify Hosting GitHub Issues repository.

Topics

- I need help using a framework adapter

- Edge API routes cause my Next.js build to fail
- On-Demand Incremental Static Regeneration isn't working for my app
- My application's build output exceeds the maximum allowed size
- My build fails with an out of memory error
- My application's HTTP response size is too large
- How do I measure my compute app's start up time locally?

I need help using a framework adapter

If you are having issues deploying an SSR app that uses a framework adapter, see Using open source adapters for any SSR framework.

Edge API routes cause my Next.js build to fail

Currently, Amplify doesn't support Next.js Edge API Routes. You must use non-edge APIs and middleware when hosting your app with Amplify.

On-Demand Incremental Static Regeneration isn't working for my app

Starting with version 12.2.0, Next.js supports Incremental Static Regeneration (ISR) to manually purge the Next.js cache for a specific page. However, Amplify doesn't currently support On-Demand ISR. If your app is using Next.js on-demand revalidation, this feature won't work when you deploy your app to Amplify.

My application's build output exceeds the maximum allowed size

Currently, the maximum build output size that Amplify supports for SSR apps is 220 MB. If you get an error message stating that the size of your app's build output exceeds the maximum allowed size, you must take steps to reduce it.

To reduce the size of an app's build output, you can inspect the app's build artifacts and identify any large dependencies to update or remove. First, download the build artifacts to your local computer. Then, check the size of the directories. For example, the `node_modules` directory might contain binaries such as `@swc` and `@esbuild` that are referenced by Next.js server runtime files. Since these binaries aren't required in the runtime, you can delete them *after* the build.

Use the following instructions to download an app's build output and inspect the size of the directories using the AWS Command Line Interface (CLI).

To download and inspect the build output for a Next.js app

1. Open a terminal window and run the following command. Change the app id, branch name, and job id to your own information. For the job id, use the build number for the failed build that you are investigating.

```
aws amplify get-job --app-id abcd1234 --branch-name main --job-id 2
```

2. In the terminal output, locate the presigned artifacts URL in the `job`, `steps`, `stepName: "BUILD"` section. The URL is highlighted in red in the following example output.

```
"job": {
    "summary": {
        "jobArn": "arn:aws:amplify:us-west-2:111122223333:apps/abcd1234/main/
jobs/0000000002",
        "jobId": "2",
        "commitId": "HEAD",
        "commitTime": "2024-02-08T21:54:42.398000+00:00",
        "startTime": "2024-02-08T21:54:42.674000+00:00",
        "status": "SUCCEED",
        "endTime": "2024-02-08T22:03:58.071000+00:00"
    },
    "steps": [
        {
            "stepName": "BUILD",
            "startTime": "2024-02-08T21:54:42.693000+00:00",
            "status": "SUCCEED",
            "endTime": "2024-02-08T22:03:30.897000+00:00",
            "logUrl": "https://aws-amplify-prod-us-west-2-artifacts.s3.us-
west-2.amazonaws.com/abcd1234/main/0000000002/BUILD/log.txt?X-Amz-Security-
Token=IQoJb3JpZ2luX2V...Example
```

3. Copy and paste the URL into a browser window. An `artifacts.zip` file is downloaded to your local computer. This is your build output.
4. Run the du disk usage command to inspect the size of the directories. The following example command returns the size of the `compute` and `static` directories.

```
du -csh compute static
```

The following is an example of the output with size information for the `compute` and `static` directories.

```
 29M    compute
3.8M    static
 33M    total
```

5. Open the `compute` directory, and locate the `node_modules` folder. Review your dependencies for files that you can update or remove to decrease the size of the folder.
6. If your app includes binaries that aren't required in the runtime, delete them after the build by adding the following commands to the build section of your app's `amplify.yml` file.

```
- rm -f node_modules/@swc/core-linux-x64-gnu/swc.linux-x64-gnu.node
- rm -f node_modules/@swc/core-linux-x64-musl/swc.linux-x64-musl.node
```

The following is an example of the build commands section of an `amplify.yml` file with these commands added *after* running a production build.

```
frontend:
  phases:
    build:
      commands:
         -npm run build

         // After running a production build, delete the files
         - rm -f node_modules/@swc/core-linux-x64-gnu/swc.linux-x64-gnu.node
         - rm -f node_modules/@swc/core-linux-x64-musl/swc.linux-x64-musl.node
```

My build fails with an out of memory error

Next.js enables you to cache build artifacts to improve performance on subsequent builds. In addition, Amplify's AWS CodeBuild container compresses and uploads this cache to Amazon S3, on your behalf, to improve subsequent build performance. This could cause your build to fail with an out of memory error.

Perform the following actions to prevent your app from exceeding the memory limit during the build phase. First, remove `.next/cache/**/*` from the cache.paths section of your build settings. Next, remove the `NODE_OPTIONS` environment variable from your build settings file. Instead, set the `NODE_OPTIONS` environment variable in the Amplify console to define the Node maximum memory limit. For more information about setting environment variables using the Amplify console, see Setting environment variables.

After making these changes, try your build again. If it succeeds, add `.next/cache/**/*` back to the cache.paths section of your build settings file.

For more information about Next.js cache configuration to improve build performance, see AWS CodeBuild on the Next.js website.

My application's HTTP response size is too large

Currently, the maximum response size that Amplify supports for Next.js 12 and later apps using the Web Compute platform is 5.72 MB. Responses over that limit return 504 errors with no content to clients.

How do I measure my compute app's start up time locally?

Use the following instructions to determine the local initialization/start up time for your Next.js 12 or later Compute app. You can compare your app's performance locally vs. on Amplify Hosting and use the results to improve your app's performance.

To measure a Next.js Compute app's initialization time locally

1. Open the app's `next.config.js` file and set the `output` option to `standalone` as follows.

```
** @type {import('next').NextConfig} */
const nextConfig = {
  // Other options
  output: "standalone",
};

module.exports = nextConfig;
```

2. Open a terminal window and run the following command to build the app.

```
next build
```

3. Run the following command to copy the `.next/static` folder to `.next/standalone/.next/static`.

```
cp -r .next/static .next/standalone/.next/static
```

4. Run the following command to copy the `public` folder to `.next/standalone/public`.

```
cp -r public .next/standalone/public
```

5. Run the following command to start the Next.js server.

```
node .next/standalone/server.js
```

6. Note how long it takes between running the command in step 5 and the server starting. When the server is listening on a port, it should print the following message.

```
Listening on port 3000
```

7. Note how long it takes for any other modules to load after the starting of the server in step 6. For example, libraries like `bugsnag` take 10-12 seconds to load. After it is loaded, it will display the confirmation message `[bugsnag] loaded`.
8. Add the time durations from step 6 and step 7 together. This result is your Compute app's local initialization/start up time.

Troubleshooting redirects and rewrites

If you encounter issues when setting up redirects and rewrites for an Amplify application, consult the topics in this section for help.

Topics

- Access is denied for certain routes even with the SPA redirect rule.
- I want to set up a reverse proxy to an API

Access is denied for certain routes even with the SPA redirect rule.

If you are getting an access denied error for certain routes with an SPA redirect rule, the `baseDirectory` might not be set correctly in the app's build settings. For example, if your app's frontend is built to the `build` directory, your build settings must also point to the `build` directory. The following build specification example demonstrates this setting.

```
frontend:
  artifacts:
    baseDirectory: build
```

```
    files:
        - '**/*'
```

For a full example of the build specification settings for an Amplify app, see Build specification YAML syntax reference

I want to set up a reverse proxy to an API

You can use the following JSON to set up a reverse proxy to a dynamic endpoint.

```
[
  {
    "source": "/documents/<*>",
    "target": "https://otherdomain/resource/<*>",
    "status": "200",
    "condition": null
  }
]
```

For a basic example of creating a reverse proxy for your Amplify app to a third-party API, see Reverse proxy rewrite.

Troubleshooting caching

If you encounter caching issues for an Amplify application, consult the topics in this section for help.

Topics

- I want to reduce the size of the cache for an app
- I want to disable reading from the cache for an app

I want to reduce the size of the cache for an app

If you are using the cache, you might be caching intermediate files that aren't cleaned up between builds. Caching these infrequently used files will increase the size of your cache. To prevent this, you can exclude specific folders from being cached by using the `!` directive in the `cache` section of your app's build specification.

The following build settings example demonstrates how to use the ! directive to specify a folder that you don't want to cache.

```
cache:
  paths:
    - node_modules/**/*
    - "!node_modules/path/not/to/cache"
```

When you cache the `node_modules` folder, `node_modules/.cache` is omitted by default.

For a full example of the build specification settings for an Amplify app, see Build specification YAML syntax reference

I want to disable reading from the cache for an app

If you want to disable reading from the cache for an app, remove the cache section from your app's build specification.

Setting up Amplify access to GitHub repositories

Amplify now uses the GitHub Apps feature to authorize Amplify read-only access to GitHub repositories. With the Amplify GitHub App, permissions are more fine-tuned, enabling you to grant Amplify access to only the repositories that you specify. To learn more about GitHub Apps, see About GitHub Apps on the GitHub website.

When you connect a new app stored in a GitHub repo, by default Amplify uses the GitHub App to access the repo. However, existing Amplify apps that you previously connected from GitHub repos use OAuth for access. CI/CD will continue to work for these apps, but we highly recommend that you migrate them to use the new Amplify GitHub App.

When you deploy a new app or migrate an existing app using the Amplify console, you are automatically directed to the installation location for the Amplify GitHub App. To manually access the installation landing page for the app, open a web browser and navigate to the app by region. Use the format `https://github.com/apps/aws-amplify-`*`REGION`*, replacing *`REGION`* with the region where you will deploy your Amplify app. For example, to install the Amplify GitHub App in the US West (Oregon) region, navigate to https://github.com/apps/aws-amplify-us-west-2.

Topics

- Installing and authorizing the Amplify GitHub App for a new deployment
- Migrating an existing OAuth app to the Amplify GitHub App
- Setting up the Amplify GitHub App for AWS CloudFormation, CLI, and SDK deployments
- Setting up web previews with the Amplify GitHub App

Installing and authorizing the Amplify GitHub App for a new deployment

When you deploy a new app to Amplify from existing code in a GitHub repo, use the following instructions to install and authorize the GitHub App.

To install and authorize the Amplify GitHub App

1. Sign in to the AWS Management Console and open the Amplify console.
2. From the **All apps** page, choose **New app**, then **Host web app**.
3. On the **Get started with Amplify Hosting** page, choose **GitHub**, then choose **Continue**.
4. If this is the first time connecting a GitHub repository, A new page opens in your browser on GitHub.com, requesting permission to authorize AWS Amplify in your GitHub account. Choose **Authorize**.
5. Next, you must install the Amplify GitHub App in your GitHub account. A page opens on Github.com requesting permission to install and authorize AWS Amplify in your GitHub account.
6. Select the GitHub account where you want to install the Amplify GitHub App.
7. Do one of the following:
   - To apply the installation to all repositories, choose **All repositories**.
   - To limit the installation to the specific repositories that you select, choose **Only select repositories**. Make sure to include the repo for the app that you are migrating in the repos that you select.
8. Choose **Install & Authorize**.
9. You are redirected to the **Add repository branch** page for your app in the Amplify console.
10. In the **Recently updated repositories** list, select the name of the repository to connect.
11. In the **Branch** list, select the name of the repository branch to connect.
12. Choose **Next**.

13. On the **Configure build settings** page, choose **Next**.
14. On the **Review** page, choose **Save and deploy**.

Migrating an existing OAuth app to the Amplify GitHub App

Existing Amplify apps that you previously connected from GitHub repositories use OAuth for repo access. We strongly recommend that you migrate these apps to use the Amplify GitHub App.

Use the following instructions to migrate an app and delete its corresponding OAuth webhook in your GitHub account. Note that the procedure for migrating varies depending on whether the Amplify GitHub app is already installed. After you migrate your first app and install and authorize the GitHub App, you only need to update the repository permissions for subsequent app migrations.

To migrate an app from OAuth to the GitHub App

1. Sign in to the AWS Management Console and open the Amplify console.
2. Choose the app that you want to migrate.
3. On the app's information page, locate the blue **Migrate to our GitHub App** message and choose **Start migration**.
4. On the **Install and authorize GitHub App** page, choose **Configure GitHub App**.
5. A new page opens in your browser on GitHub.com, requesting permission to authorize AWS Amplify in your GitHub account. Choose **Authorize**.
6. Select the GitHub account where you want to install the Amplify GitHub App.
7. Do one of the following:
   - To apply the installation to all repositories, choose **All repositories**.
   - To limit the installation to the specific repositories that you select, choose **Only select repositories**. Make sure to include the repo for the app that you are migrating in the repositories that you select.
8. Choose **Install & Authorize**.
9. You are redirected to the **Install and authorize GitHub App** page for your app in the Amplify console. If GitHub authorization was successful, you will see a success message. Choose, **Next**.
10. On the **Complete installation** page, choose **Complete installation**. This step deletes your existing webhook, creates a new one, and completes the migration.

Setting up the Amplify GitHub App for AWS CloudFormation, CLI, and SDK deployments

Existing Amplify apps that you previously connected from GitHub repositories use OAuth for repo access. This can include apps that you deployed using the Amplify Command Line Interface (CLI), AWS CloudFormation, or the SDKs. We strongly recommend that you migrate these apps to use the new Amplify GitHub App. Migration must be performed in the Amplify console in the AWS Management Console. For instructions, see Migrating an existing OAuth app to the Amplify GitHub App.

You can use AWS CloudFormation, the Amplify CLI, and the SDKs to deploy a new Amplify app that uses the GitHub App for repo access. This process requires that you first install the Amplify GitHub App in your GitHub account. Next, you will need to generate a personal access token in your GitHub account. Lastly, deploy the app and specify the personal access token.

Install the Amplify GitHub App in your account

1. Open a web browser and navigate to the installation location for the Amplify GitHub App in the AWS Region where you will deploy your app.

   Use the format `https://github.com/apps/aws-amplify-`*`REGION`*`/installations/new`, replacing *`REGION`* with your own input. For example, if you are installing your app in the US West (Oregon) region, specify `https://github.com/apps/aws-amplify-us-west-2/installations/new`.
2. Select the GitHub account where you want to install the Amplify GitHub app.
3. Do one of the following:
   - To apply the installation to all repositories, choose **All repositories**.
   - To limit the installation to the specific repositories that you select, choose **Only select repositories**. Make sure to include the repo for the app that you are migrating in the repos that you select.
4. Choose **Install**.

Generate a personal access token in your GitHub account

1. Sign in to your GitHub account.
2. In the upper right corner, locate your profile photo and choose **Settings** from the menu.

3. In the left navigation menu, choose **Developer settings**.
4. On the **GitHub Apps** page, in the left navigation menu, choose **Personal access tokens**.
5. On the **Personal access tokens** page, choose **Generate new token**.
6. On the **New personal access token** page, for **Note** enter a descriptive name for the token.
7. In the **Select scopes** section, select **admin:repo_hook**.
8. Choose **Generate token**.
9. Copy and save the personal access token. You will need to provide it when you deploy an Amplify app with the CLI, AWS CloudFormation, or the SDKs.

After the Amplify GitHub app is installed in your GitHub account and you have generated a personal access token, you can deploy a new app with the Amplify CLI, AWS CloudFormation, or the SDKs. Use the `accessToken` field to specify the personal access token that you created in the previous procedure. For more information, see CreateApp in the *Amplify API reference* and AWS::Amplify::App in the *AWS CloudFormation User Guide*.

The following CLI command deploys a new Amplify app that uses the GitHub App for repository access. Replace *myapp-using-githubapp*, *https://github.com/Myaccount/react-app*, and *MY_TOKEN* with your own information.

```
aws amplify create-app --name myapp-using-githubapp --repository https://github.com/
Myaccount/react-app --access-token MY_TOKEN
```

Setting up web previews with the Amplify GitHub App

A web preview deploys every pull request (PR) made to your GitHub repository to a unique preview URL. Previews now use the Amplify GitHub App for access to your GitHub repo. For instructions on installing and authorizing the GitHub App for web previews, see Enable web previews for pull requests .

AWS Amplify Hosting reference

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

Topics

- AWS CloudFormation support
- AWS Command Line Interface support
- Resource tagging support
- Amplify Hosting API

AWS CloudFormation support

Use AWS CloudFormation templates to provision Amplify 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 Amplify Hosting, see the Amplify CloudFormation documentation. For Amplify Studio, see the Amplify UI Builder CloudFormation documentation.

AWS Command Line Interface support

Use the AWS Command Line Interface to create Amplify 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.

Amplify Hosting API

This reference provides descriptions of the actions and data types for the Amplify Hosting API. For more information, see the Amplify API reference 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:** May 28, 2025

Change	Description	Date
Updated Build settings and configuration chapter	Updated the Managing the build configuration for an Amplify application chapter to describe the new configura ble build instance type feature that enables you to choose an instance type that provides your application with the CPU, memory, and disk space resources that it requires.	May 28, 2025
Updated Firewall chapter	Updated the Firewall support for Amplify hosted sites chapter to describe the general availability (GA) of Amplify's integration with AWS WAF, including the GA functionality and pricing structure.	March 26, 2025
New Skew protection chapter	Added the Skew protectio n for Amplify deploymen ts chapter to describe the skew protection feature that eliminates version skew issues	March 10, 2025

Change	Description	Date
	between client and servers in Amplify web applications.	
Updated Webhooks chapter	Added the Unified webhooks for Git repositories topic to describe the unified webhooks feature that uses one comprehensive webhook for all Amplify applications associated with a single Git repository.	March 10, 2025
New Adding an SSR Compute role to allow access to AWS resources topic	Added the Adding an SSR Compute role to allow access to AWS resources topic to describe how to create and associate an Amplify SSR Compute role with an app to give the Amplify Compute service access to AWS resources.	February 17, 2025
New Using AWS WAF to protect your Amplify apps chapter	Added the Firewall support for Amplify hosted sites chapter to describe Amplify's integration with AWS WAF (in preview) that enables you to protect your web applications with a web access control list (web ACL).	December 18, 2024
Updated managed policies topic	Updated the AWS managed policies for AWS Amplify topic to describe recent changes to the AWS managed policies for Amplify.	November 14, 2024

Change	Description	Date
Updated Amplify support for Next.js topic	Updated the Amplify support for Next.js topic to describe Amplify's support for Next.js version 15.	November 6, 2024
New Deploying a static website to Amplify from Amazon S3 chapter	Added the Deploying a static website to Amplify from an Amazon S3 bucket chapter to describe Amplify's new integration with Amazon S3 that enables you to host static website content stored on S3 with just a few clicks.	October 16, 2024
New Managing cache configuration chapter	Added the Managing the cache configuration for an app chapter to describe Amplify's default caching behavior and how it applies managed cache policies to content.	August 13, 2024
Updated managed policies topic	Updated the AWS managed policies for AWS Amplify topic to describe recent changes to the AWS managed policies for Amplify.	July 18, 2024
Updated managed policies topic	Updated the AWS managed policies for AWS Amplify topic to describe recent changes to the AWS managed policies for Amplify.	May 31, 2024

Change	Description	Date
Updated managed policies topic	Updated the AWS managed policies for AWS Amplify topic to describe recent changes to the AWS managed policies for Amplify.	April 17, 2024
Updated getting started chapter	Updated the Getting started with deploying an app to Amplify Hosting chapter to use a Next.js example application in the tutorial.	April 12, 2024
Updated managed policies topic	Updated the AWS managed policies for AWS Amplify topic to describe recent changes to the AWS managed policies for Amplify.	April 5, 2024
Updated managed policies topic	Updated the AWS managed policies for AWS Amplify topic to describe recent changes to the AWS managed policies for Amplify.	April 4, 2024
New Troubleshooting chapter	Added the Troubleshooting Amplify Hosting chapter to describe how to fix issues that you encounter with applicati ons deployed to Amplify Hosting.	April 2, 2024

Change	Description	Date
New support for custom SSL/TLS certificates	Added the Using SSL/TLS certificates topic to the Connecting a custom domain chapter to describe Amplify support for custom SSL/TLS certificates when connecting an app to a custom domain.	February 20, 2024
Updated managed policies topic	Updated the AWS managed policies for AWS Amplify topic to describe recent changes to the AWS managed policies for Amplify.	January 2, 2024
New support for SSR frameworks	Updated the Deploying server-side rendered applications with Amplify Hosting topic to describe Amplify support for any Javascript-based SSR framework with an open-source adapter.	November 19, 2023
New support for image optimization feature launch	Added the Image optimization for SSR apps topic to describe the built-in support for image optimization for server-side rendered apps.	November 19, 2023
Updated managed policies topic	Updated the AWS managed policies for AWS Amplify topic to describe recent changes to the AWS managed policies for Amplify.	November 17, 2023

Change	Description	Date
Updated managed policies topic	Updated the AWS managed policies for AWS Amplify topic to describe recent changes to the AWS managed policies for Amplify.	November 6, 2023
New wildcard subdomains topic	Added the Setting up wildcard subdomains topic to describe support for wildcard subdomains on custom domains.	November 6, 2023
New managed policy	Updated the AWS managed policies for AWS Amplify topic to describe the new AmplifyBackendDeployFullAccess AWS managed policy for Amplify.	October 8, 2023
New support for monorepo frameworks feature launch	Updated the Configuring monorepo build settings topic to describe support for deploying apps in monorepos created using npm workspace, pnpm workspace, Yarn workspace, Nx, and Turborepo.	June 19, 2023
Updated managed policies topic	Updated the AWS managed policies for AWS Amplify topic to describe recent changes to the AWS managed policies for Amplify.	June 1, 2023

Change	Description	Date
Updated managed policies topic	Updated the AWS managed policies for AWS Amplify topic to describe recent changes to the AWS managed policies for Amplify.	February 24, 2023
Updated server-side rendering chapter	Updated the Deploying server-side rendered applications with Amplify Hosting chapter to describe recent changes to Amplify's support for Next.js versions 12 and 13.	November 17, 2022
Updated managed policies topic	Updated the AWS managed policies for AWS Amplify topic to describe recent changes to the AWS managed policies for Amplify.	August 30, 2022
Updated managed policies topic	Updated the Building a backend for an application topic to describe how to deploy a backend using Amplify Studio.	August 23, 2022
Updated managed policies topic	Updated the AWS managed policies for AWS Amplify topic to describe recent changes to the AWS managed policies for Amplify.	April 27, 2022
Updated managed policies topic	Updated the AWS managed policies for AWS Amplify topic to describe recent changes to the AWS managed policies for Amplify.	April 17, 2022

Change	Description	Date
New GitHub App feature launch	Added the Setting up Amplify access to GitHub repositories topic to describe the new GitHub App for authorizing Amplify access to your GitHub repository.	April 5, 2022
New Amplify Studio feature launch	Updated the Welcome to AWS Amplify Hosting topic to describe the updates to Amplify Studio that provide a visual designer to create UI components that you can connect to your backend data.	December 2, 2021
Updated managed policies topic	Updated the AWS managed policies for AWS Amplify topic to describe recent changes to the AWS managed policies for Amplify to support Amplify Studio.	December 2, 2021
Updated managed policies topic	Updated the AWS managed policies for AWS Amplify topic to describe recent changes to the AWS managed policies for Amplify.	November 8, 2021
Updated managed policies topic	Updated the AWS managed policies for AWS Amplify topic to describe recent changes to the AWS managed policies for Amplify.	September 27, 2021

Change	Description	Date
New managed policies topic	Added the AWS managed policies for AWS Amplify 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 Deploying server-side rendered applications with Amplify Hosting 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 Configuring monorepo build settings 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

Change	Description	Date
Updated Feature branch deployments chapter	Added the Automatic build-time generation of Amplify config (Gen 1 apps only) topic to describe how to autogenerate the `aws-exports.js` file at build-time. Added the Conditional backend builds (Gen 1 apps only) topic to describe how to enable conditional backend builds. Added the Use Amplify backends across apps (Gen 1 apps only) 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 existing frontend to point to a different backend environment.	June 30, 2021
Updated Security chapter	Added the Data Protection in Amplify 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

Change	Description	Date
New support for SSR feature launch	Added the Deploying server-side rendered applications with Amplify Hosting 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 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 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 topic to describe how to access Amazon CloudWatch metrics data and set alarms.	February 2, 2021

Change	Description	Date
New CloudTrail logging topic	Added the Logging Amplify API calls using AWS CloudTrail 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
New Admin UI feature launch	Updated the Welcome to AWS Amplify Hosting 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** 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 topic to describe how to define custom headers for an Amplify app using the console or by editing a YML file.	October 28, 2020

Change	Description	Date
New auto subdomains feature launch	Added the Set up automatic subdomains for a Route 53 custom domain 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 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 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 Connecting a custom domain topic to improve the procedures for adding custom domains in Amazon Route 53, GoDaddy, and Google Domains. This update also includes new troubleshooting informati on for setting up custom domains.	May 12, 2020

Change	Description	Date
AWS Amplify release	This release introduces Amplify.	November 26, 2018