# Churchwell Family Reunion Website (CFR Next)

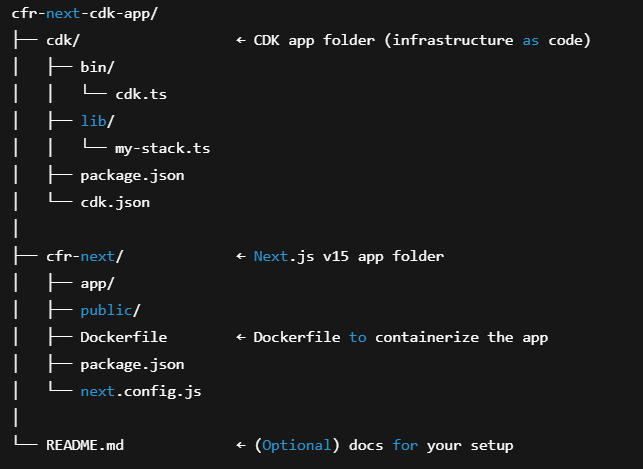
## Overview

The **CFR Next** application is a modern, full-stack web platform built with Next.js for the frontend and AWS CDK for infrastructure-as-code. The project is organized into two main directories:

* [cfr-next](vscode-file://vscode-app/c:/Users/mudge/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html): Contains the Next.js application, API routes, React components, and supporting libraries.
* [cdk](vscode-file://vscode-app/c:/Users/mudge/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html): Contains AWS CDK stacks and constructs that provision and manage cloud resources.

This architecture leverages AWS managed services—**Cognito**, **DynamoDB**, **S3**, and **Lambda**—to deliver a secure, scalable, and cost-effective solution for user authentication, data storage, file management, and business logic. Below, I’ll break down the design, resource choices, and how the app aligns with the six pillars of the AWS Well-Architected Framework.

Here is a look at our file structure:



## Project Set Up

1. Initialize a new cdk project
   1. Install AWS CDK npm install -g aws-cdk
   2. Initialize a new CDK project cdk init app --language typescript
2. Install dependencies npm install aws-cdk-lib constructs
   1. This installs the core libraries needed to write and deploy CDK infrastructure
      1. It’s a single unified library, no longer need to install each service as a separate package.
      2. Allows you to perform the following:

A computer screen with text

AI-generated content may be incorrect.

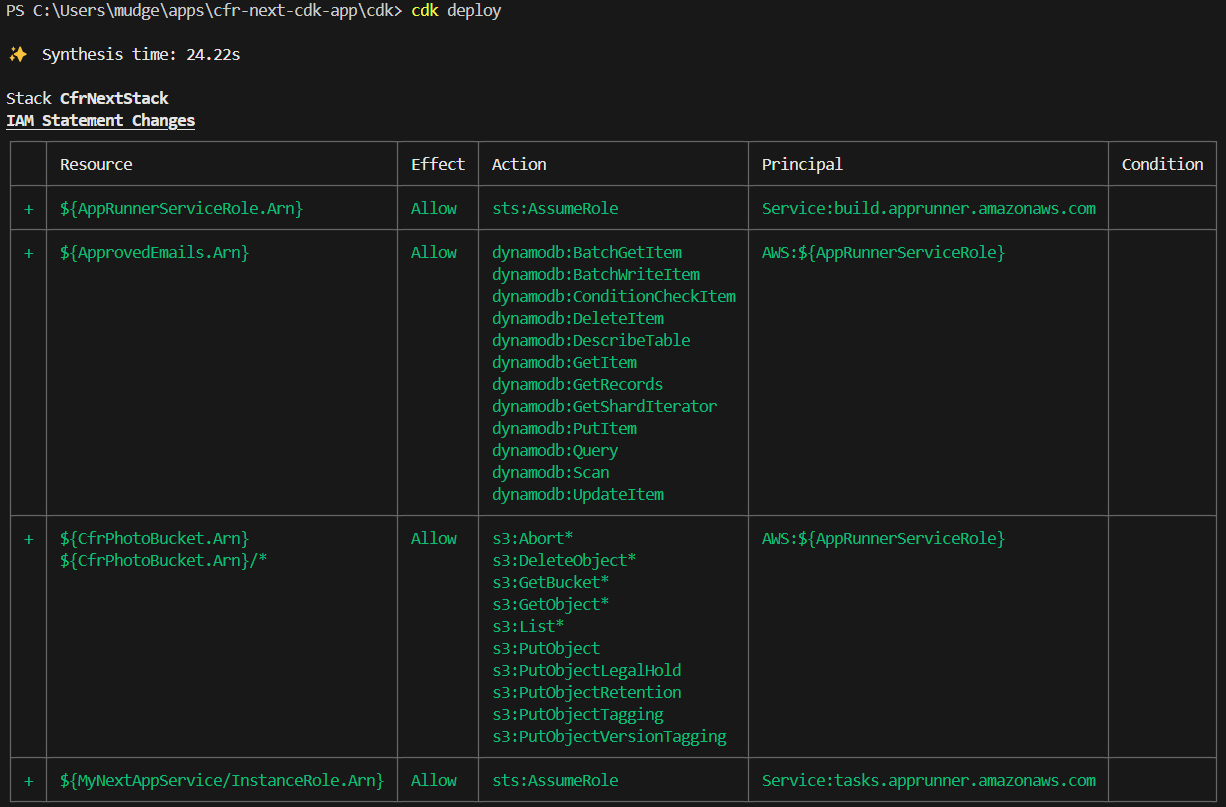
* 1. It’s the base class for defining reusable infrastructure components in CDK
     1. All stacks and constructs inherit from Construct

A screen shot of a computer code

AI-generated content may be incorrect.

* + 1. s

1. Configure AWS CLI
   1. Using the command aws configure you can set the following:
      1. AWS Access Key ID
      2. AWS Secret Access Key
      3. Default region name
      4. Default output format
2. Create the Next.js app
   1. cd .. into the root directory. Create a new app using npx create-next-app@latest
   2. Name your app. We will use cfr-next
   3. Accept the standard defaults. TypeScript, ESLint, Tailwind CSS, scr/ = No, App Router, Turbopack, Import alias = no,
3. Add Dockerfile
   1. Place the Dockerfile in the app directory (cfr-next)
4. # Use official Node.js LTS image
5. FROM node:20-alpine
6. # Set working directory
7. WORKDIR /app
8. # Copy package.json and package-lock.json (or yarn.lock/pnpm-lock.yaml)
9. COPY package.json .
10. COPY yarn.lock .
11. # Install dependencies
12. RUN yarn install --frozen-lockfile
13. # Copy rest of the app
14. COPY . .
15. # Build Next.js app
16. RUN yarn build
17. # Expose port 3000
18. EXPOSE 3000
19. # Start Next.js app
20. CMD ["yarn", "start"]
21. Define Infrastructure with CDK
    1. For this app we require a S3 bucket, DynamoDB table, and an IAM Role for App Runner service to access the S3 bucket and DynaoDB table and Cognito
    2. Create the following cdk/lib/cdk-stack.ts
22. import { Stack, StackProps, CfnOutput } from 'aws-cdk-lib';
23. import { Construct } from 'constructs';
24. import \* as s3 from 'aws-cdk-lib/aws-s3';
25. import \* as dynamodb from 'aws-cdk-lib/aws-dynamodb';
26. import \* as iam from 'aws-cdk-lib/aws-iam';
27. import \* as apprunner from '@aws-cdk/aws-apprunner-alpha';
28. import \* as cognito from 'aws-cdk-lib/aws-cognito';
29. export class CfrNextStack extends Stack {
30. constructor(scope: Construct, id: string, props?: StackProps) {
31. super(scope, id, props);
32. // S3 Bucket
33. const bucket = new s3.Bucket(this, 'CfrPhotoBucket');
34. // DynamoDB Table
35. const approvedUsersTable = new dynamodb.Table(this, 'ApprovedEmails', {
36. partitionKey: { name: 'email', type: dynamodb.AttributeType.STRING },
37. tableName: 'ApprovedEmails'
38. });
39. // Cognito User Pool
40. const userPool = new cognito.UserPool(this, 'UserPool', {
41. userPoolName: 'CfrUserPool',
42. selfSignUpEnabled: true,
43. signInAliases: { email: true },
44. autoVerify: { email: true },
45. accountRecovery: cognito.AccountRecovery.EMAIL\_ONLY,
46. });
47. const userPoolClient = new cognito.UserPoolClient(this, 'UserPoolClient', {
48. userPool,
49. generateSecret: false,
50. authFlows: { userPassword: true, userSrp: true },
51. });
52. new CfnOutput(this, 'UserPoolId', {
53. value: userPool.userPoolId,
54. });
55. new CfnOutput(this, 'UserPoolClientId', {
56. value: userPoolClient.userPoolClientId,
57. });
58. CDK Bootstrap
    1. Run the command cdk bootstrap
    2. This sets up the initial infrastructure that the CDK needs in your AWS environment before it can deploy your application
    3. It creates resources in your AWS account that the CDK needs to perform deployments
59. CDK Deploy
    1. Run the command cdk deploy
    2. Synthesizes the cdk-stack.ts code into a CloudFormation template (cdk synth)
    3. Uploads any assets (like Lambda code or Docker images)
    4. Deploys the synthesized template using CloudFormation



* 1. Type y to accept these change and continue the deployment
  2. When complete you will see a list of outputs. Copy these and save them because we will use them in the future

A computer screen with white text

AI-generated content may be incorrect.

1. Verify Resources Created
   1. Navigate to the AWS Console, Click on CloudFormation for the account that is associated with this app
   2. You should see a stack created. “CfrNextStack” in our case
   3. Clicking on the Resources tab will show all the resources that were just created

A screenshot of a computer

AI-generated content may be incorrect.

* 1. Clicking the plus by each resource will open a sub window with a link to take you to that associated resource. Note that these resources are defined with a CDK created custom identifier after the name we defined in the cdk-stack.ts

A screenshot of a computer

AI-generated content may be incorrect.

1. Set Up our Next.js App with Cognito Auth
   1. Move to the app directory cd ../cfr-next
   2. Install packages npm install @aws-sdk/client-cognito-identity-provider @aws-sdk/client-dynamodb jsonwebtoken cookie
   3. Create .env.local
      1. Using the outputs from the stack define the Cognito Client ID, User Pool ID, and JWT Secret
2. NEXT\_PUBLIC\_COGNITO\_CLIENT\_ID=XXXXXXXXXXXXXXXXXXXX
3. NEXT\_PUBLIC\_COGNITO\_USER\_POOL\_ID=us-east-1\_XXXXXXXX
4. COGNITO\_JWT\_SECRET=XXXXXXX-XX-XXXXXX-XXX-XXXXXXXXXXXXXXXXXXXXXXXX
   1. Create Cognito Auth Helper (lib/cognitoClient.ts)
5. Dev permissions
   1. Create iam user for dev testing
   2. Create custom policy that gives permissions to dynamodb table, cognito-idp:AdminInitiateAuth