PROJECT REPORT

DEPLOYMENT OF A REACT APP USING AWS S3 AND CLOUDFRONT

PREPARED BY: NDUBUISI AGUGHARA
DATE: 17TH /2/ 2025

PROJECT

1. Introduction

Deploying a React applicatioan on AWS S3 and CloudFront is a highly scalable and cost-effective solution for hosting static web applications. AWS S3 serves as a storage solution for the application files, while AWS CloudFront provides a content delivery network (CDN) to ensure low-latency access and improved performance for users across different regions.

Ndubuisi AgugharaJob

2. Project Objectives

The primary objectives of this project include:

- Hosting a React application using AWS services.
- Utilizing Amazon S3 for static file storage.
- Implementing AWS CloudFront for efficient content delivery and caching.
- Ensuring security, scalability, and cost-efficiency.

3. Technologies Used

- React.js: Front-end JavaScript framework for building the application.
- AWS S3: Object storage service used to store static files.
- AWS CloudFront: CDN service to distribute the content globally with low latency.
- AWS Route 53: (Optional) Domain Name System (DNS) service for custom domain configuration.
- AWS Certificate Manager (ACM): For securing the website using SSL/TLS.

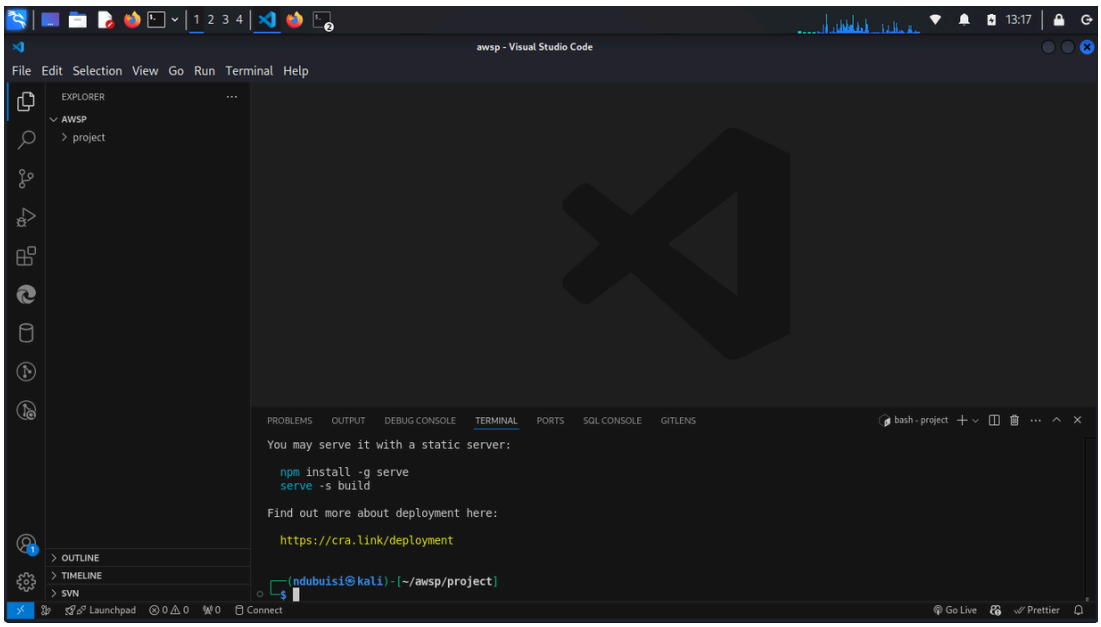
SETTING UP REACT APP

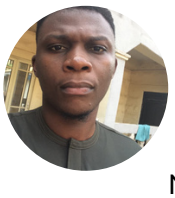
With the command ‘npx create-react-app my-project’ I created a React application, which sets up a new project structure along with default configurations and dependencies and initial scripts.

I used the npm run build command to generate static files. It builds the React application and optimizes it for production by minifying the JavaScript, CSS etc.

The output is in the build/ directory which has all the files required to deploy the app to a web server such as AWS S3.

The static files provided by React enable the application to operate efficiently in production without the requirement of a Node.js server





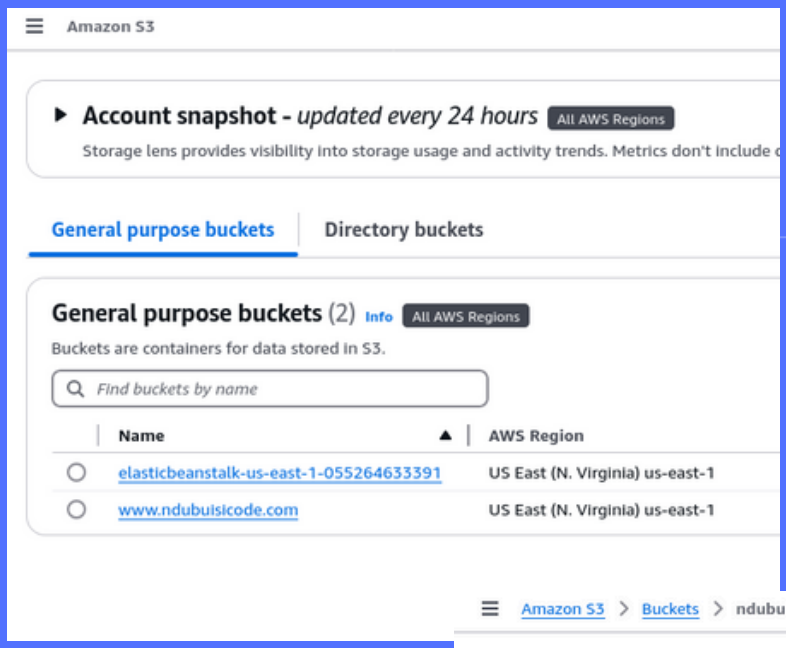
Ndubuisi AgugharaJob

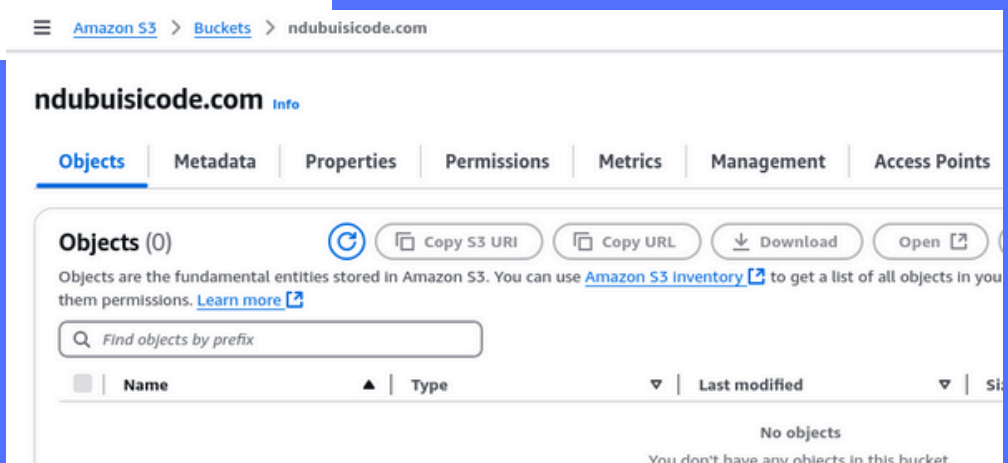
Configure AWS S3 Bucket

For this project, I created two separate S3 buckets:

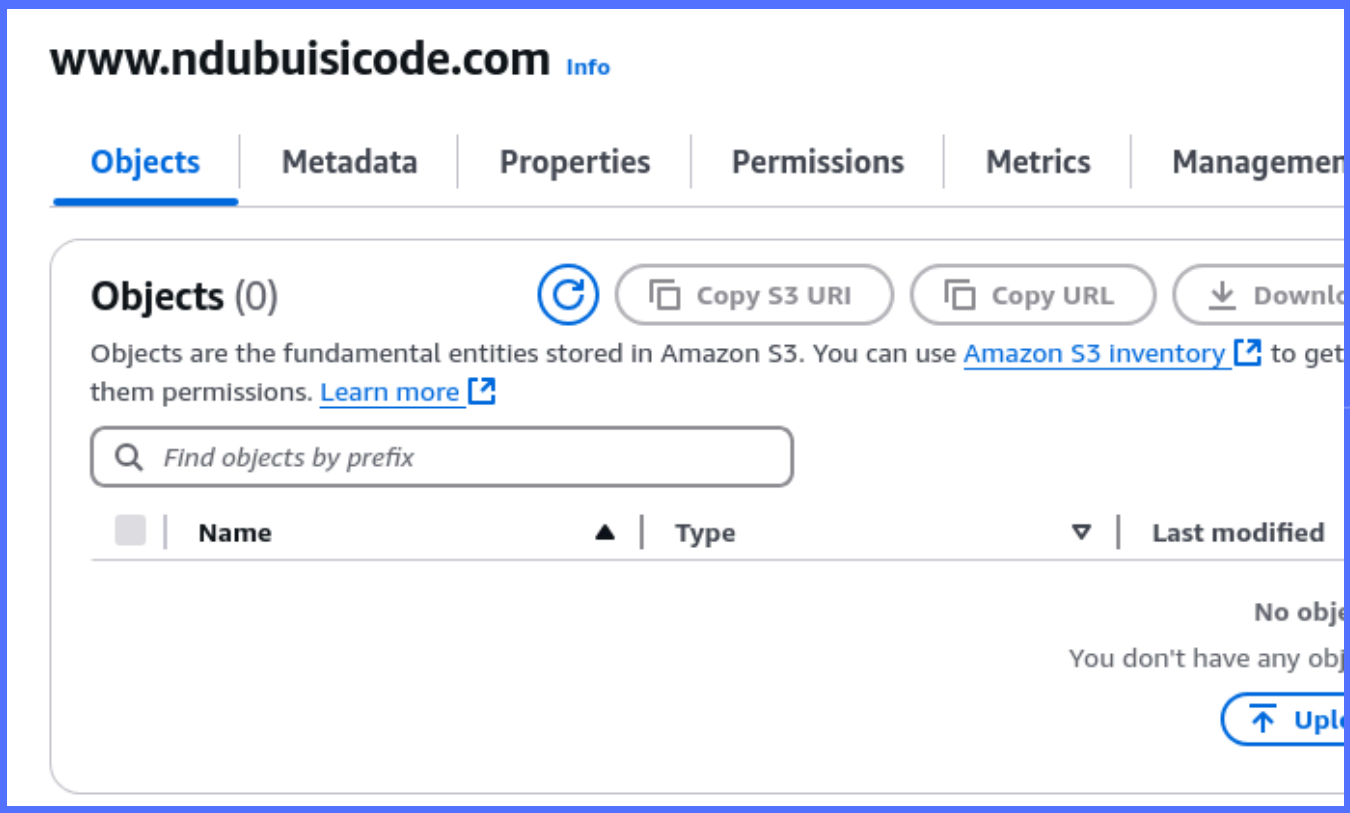
1. **Primary Bucket** – This bucket is named after the website domain without the "www" prefix (e.g., mywebsite.com). It serves as the authoritative source, hosting the static files for the React application.
2. **Redirect Bucket** – This bucket includes the "www" prefix (e.g., www.mywebsite.com). It is configured to redirect all traffic to the primary bucket, ensuring a seamless user experience regardless of whether visitors use "www" in the URL.

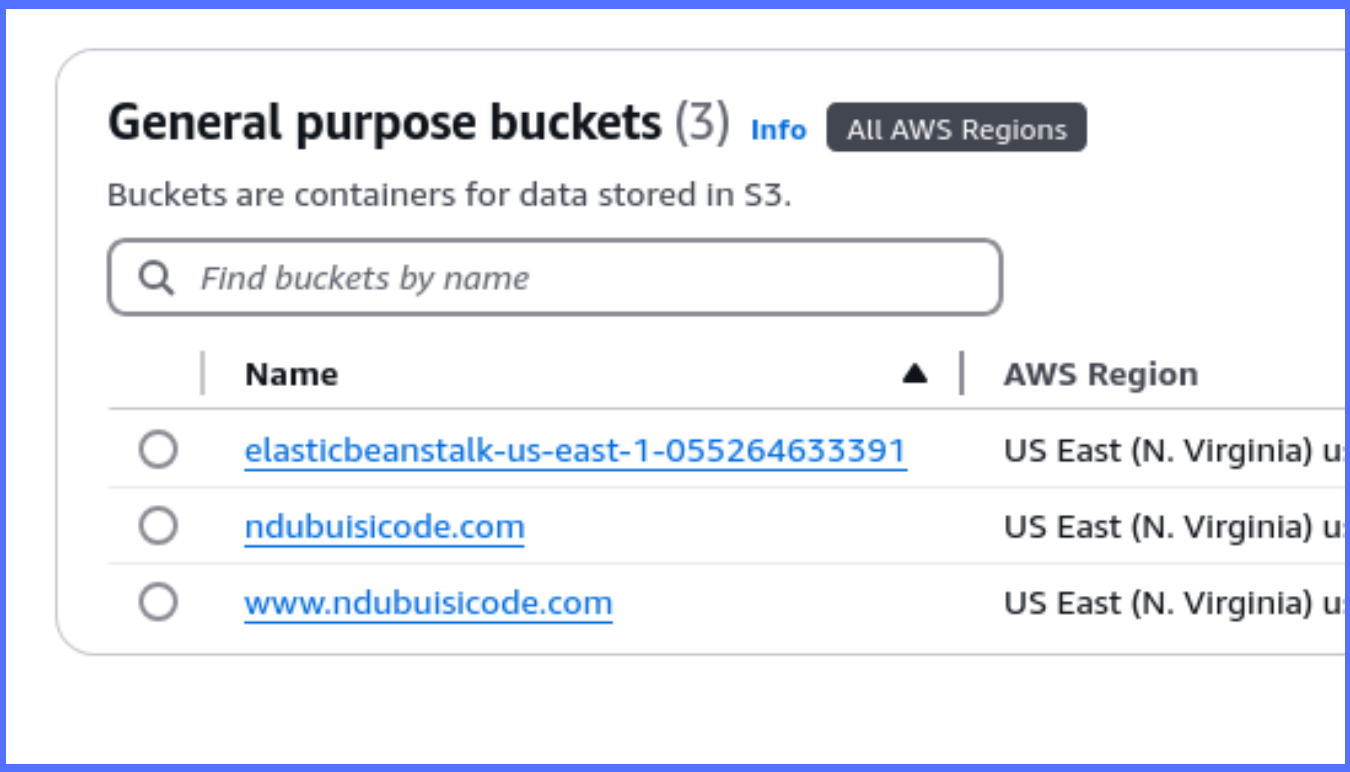
FIRST BUCKET I CREATED





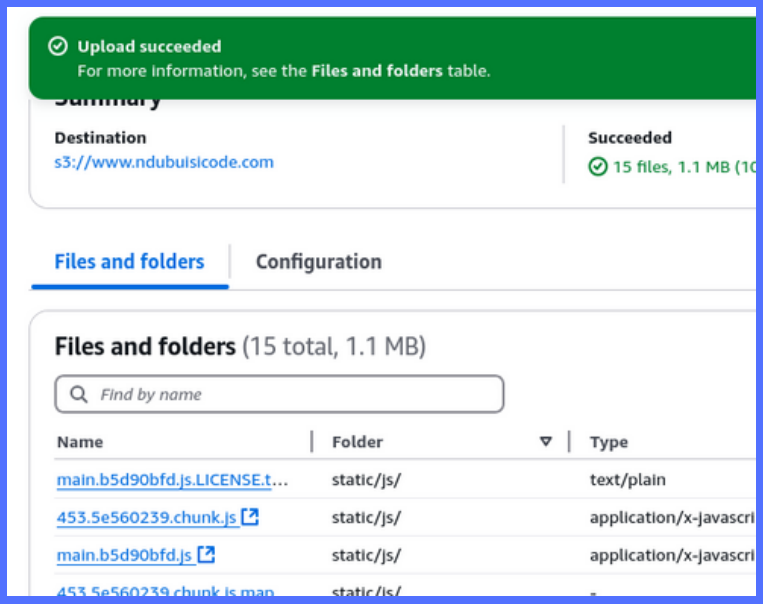
SECOND BUCKET I CREATED BUCKET NAME: WWW.NDUBUISICODE.COM



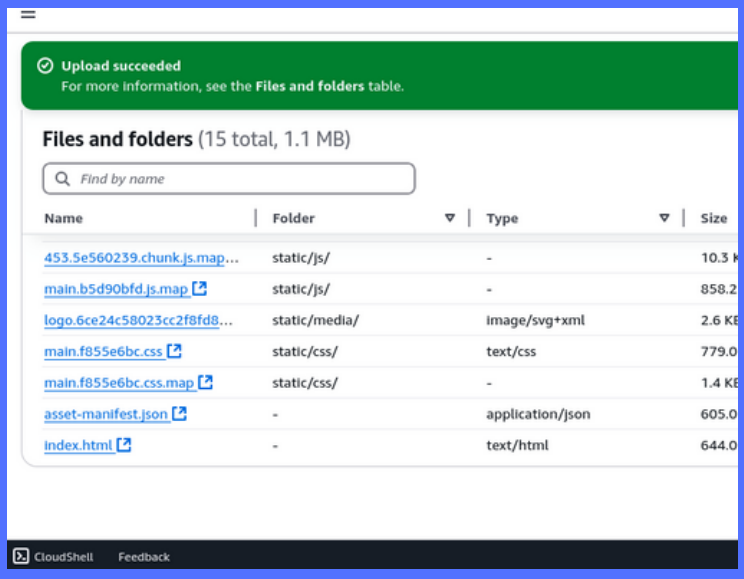


Uploaded the React build files to the S3 bucket.

uploaded the React build files to the primary S3 bucket, ensuring that all static assets, including HTML, CSS, and JavaScript files, are available for hosting. This allows the application to be served directly from S3.



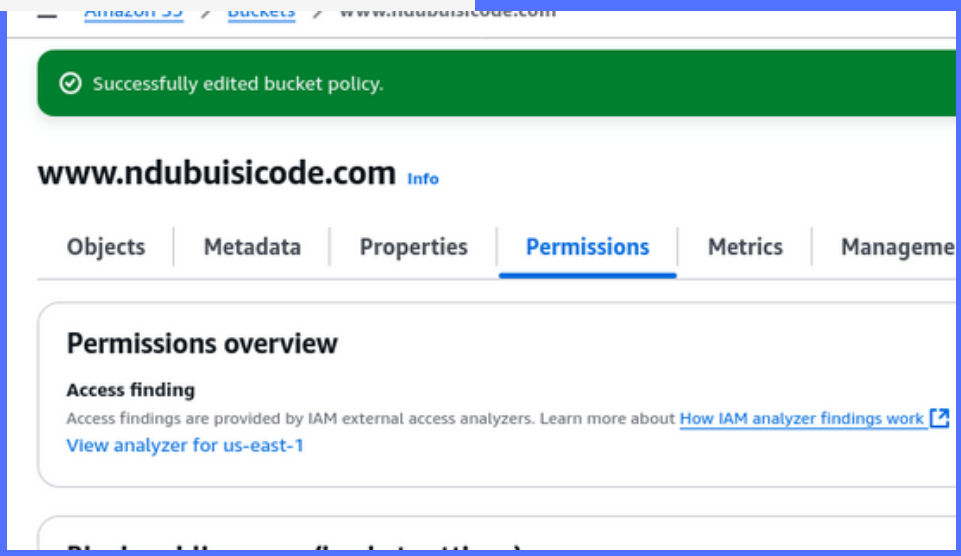
Additionally, I configured the bucket for public access (if needed) and enabled static website hosting to serve the React app properly.



Configured bucket policies to allow public read access (or used AWS IAM policies for access control).

I configured the S3 bucket policies to allow public read access, enabling users to access the React app through a web browser. This involved updating the bucket policy to grant s3:GetObject permissions for all objects in the bucket while ensuring security best practices.

Alternatively, I could use AWS IAM policies to manage access control, restricting permissions to specific users or services while maintaining the necessary access for public content delivery.



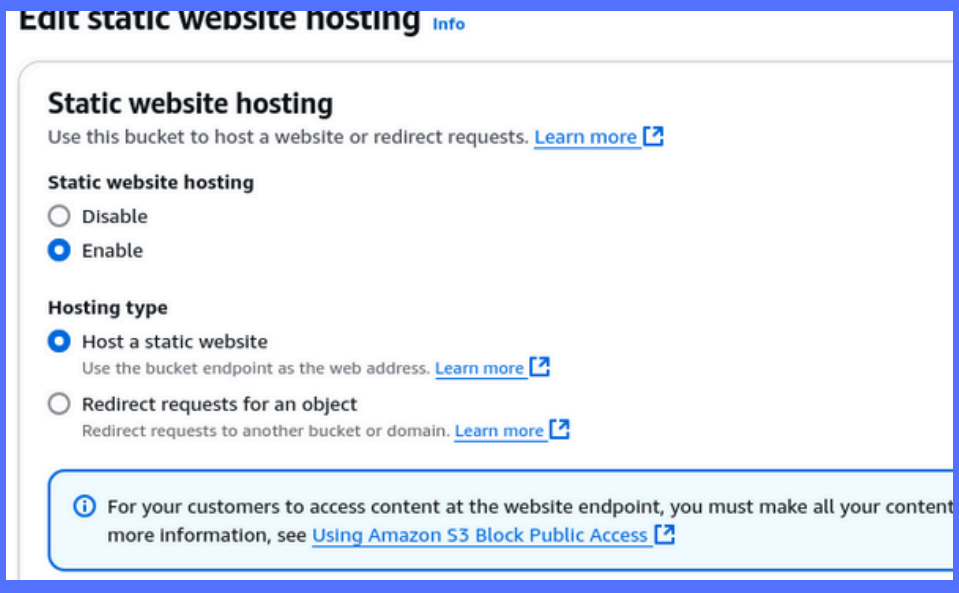
My Error

I encountered an error when trying to access my link, which resulted in a blank page.
Link: https://s3.us-east-1.amazonaws.com/www.ndubuisicode.com/index.html.

This is because is loading just the file and not all the whole react file.

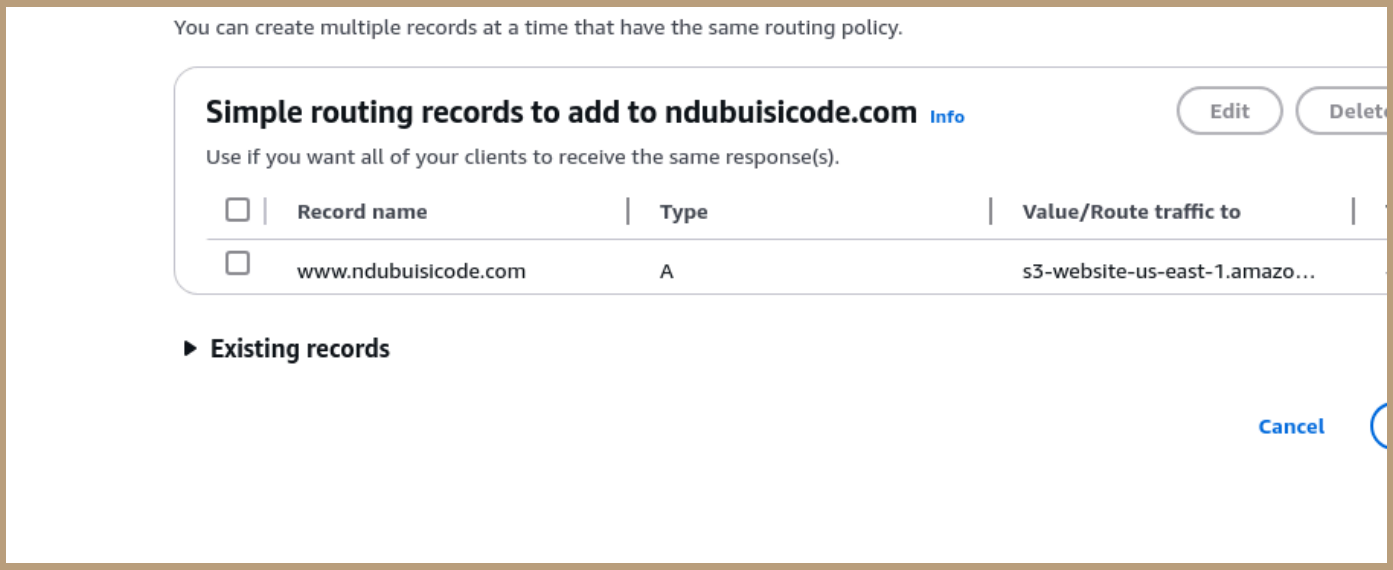
But at this stage , the bucket is public.

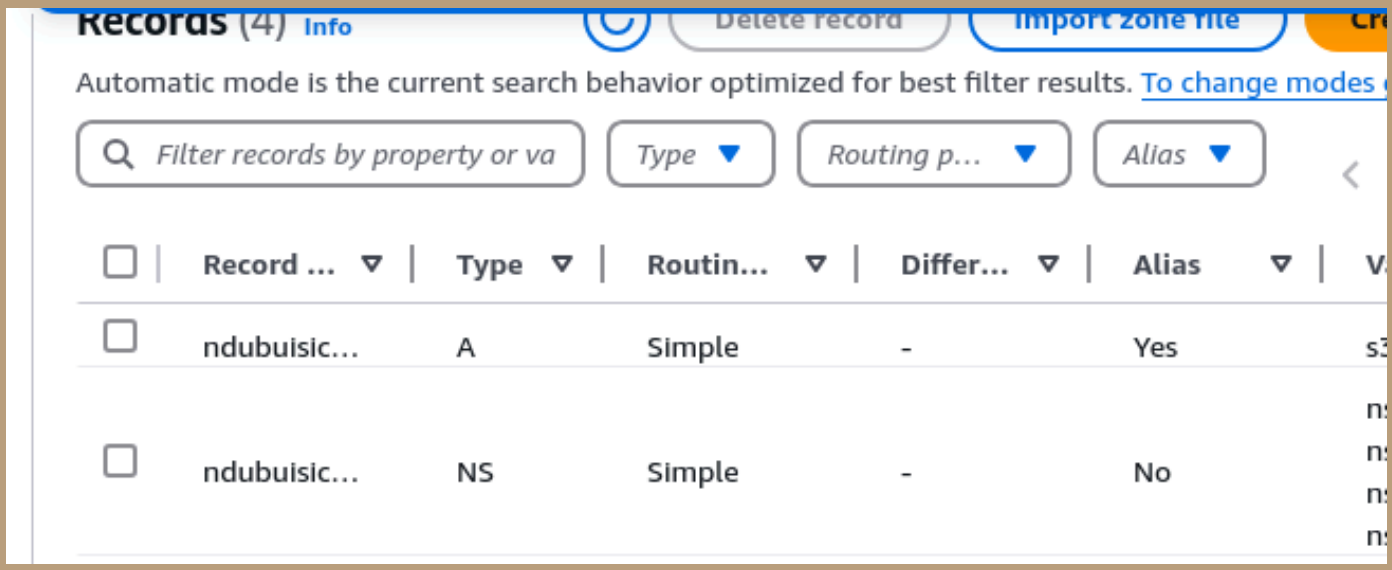
Edit static website hosting

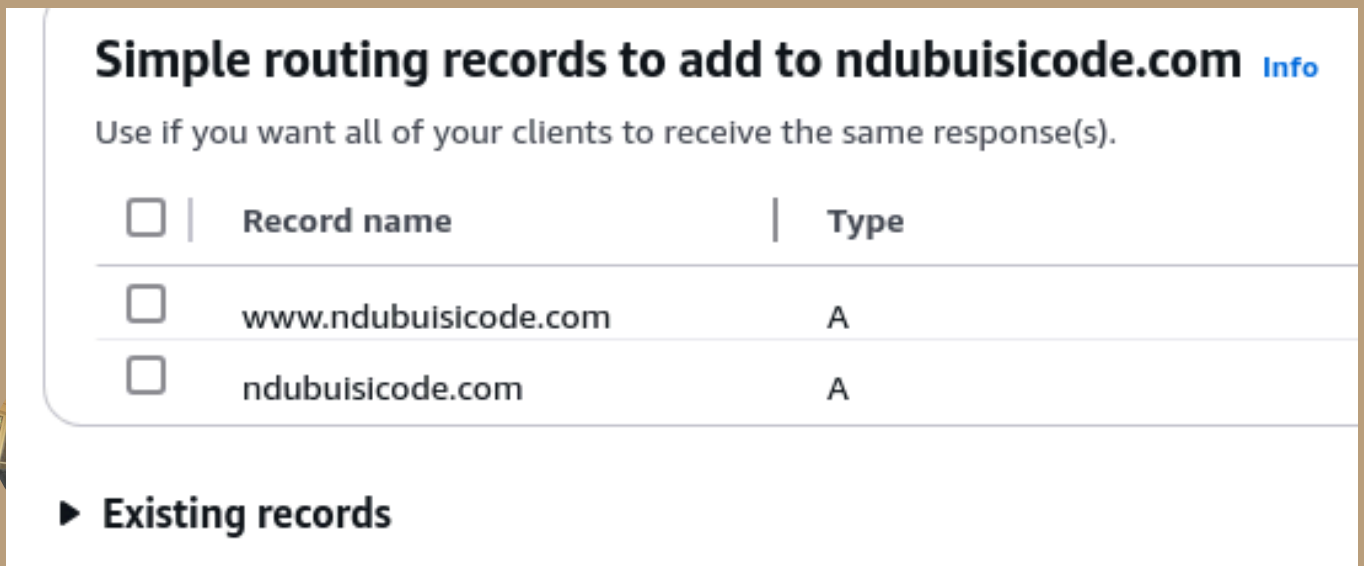


Configure records

To make the S3-hosted React app accessible via its domain, I configured DNS records in AWS Route 53.







PROJECT

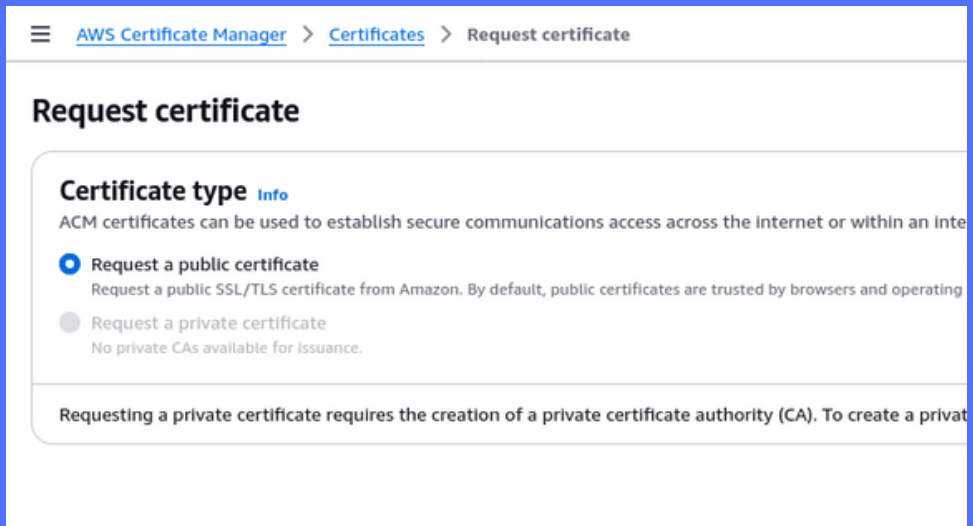
Set Up AWS CloudFront

To improve performance, security, and global accessibility, I integrated AWS CloudFront with the S3-hosted React app:

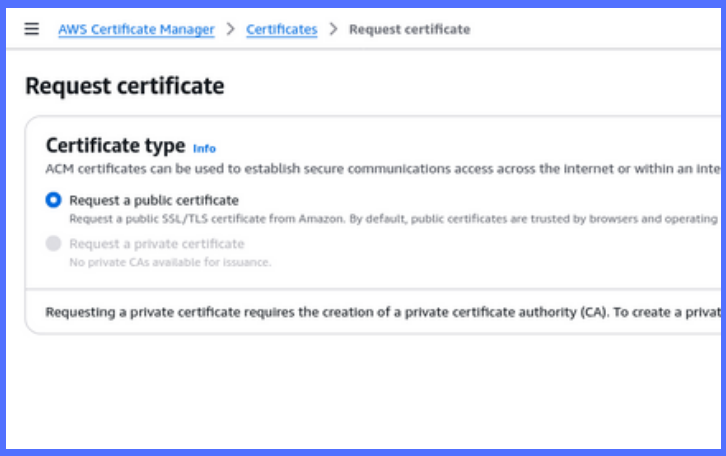
why cloudefront

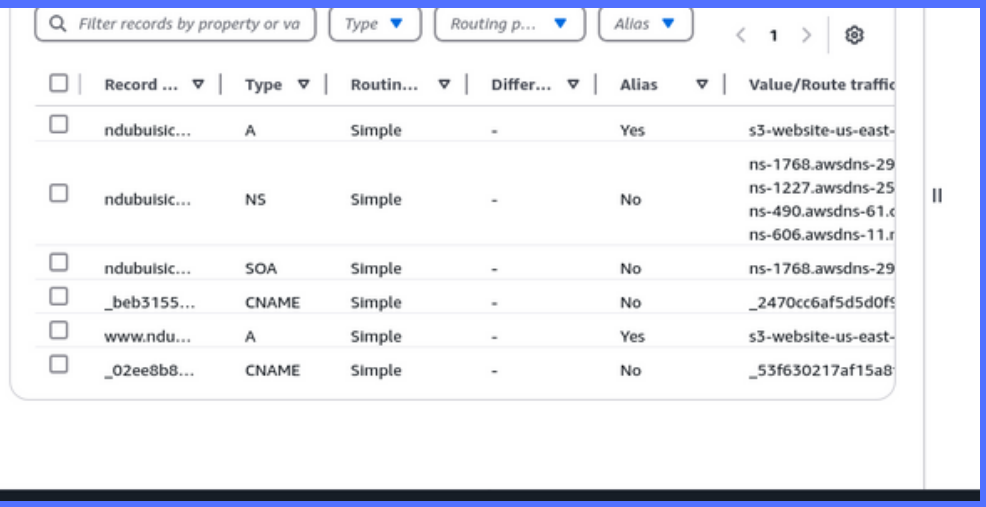
- Performance Improvement (CDN)
- Caches content at edge locations for faster load times.
- Global Distribution
- Serves content from a global network of edge locations, reducing latency.
- Scalability
- Handles traffic spikes without affecting performance.
- Security
- SSL/TLS encryption for secure HTTPS access.
- Restricts direct S3 access to CloudFront.
- Provides automatic DDoS protection.
- Cost Savings
- Reduces data transfer costs by caching content and offloading traffic from

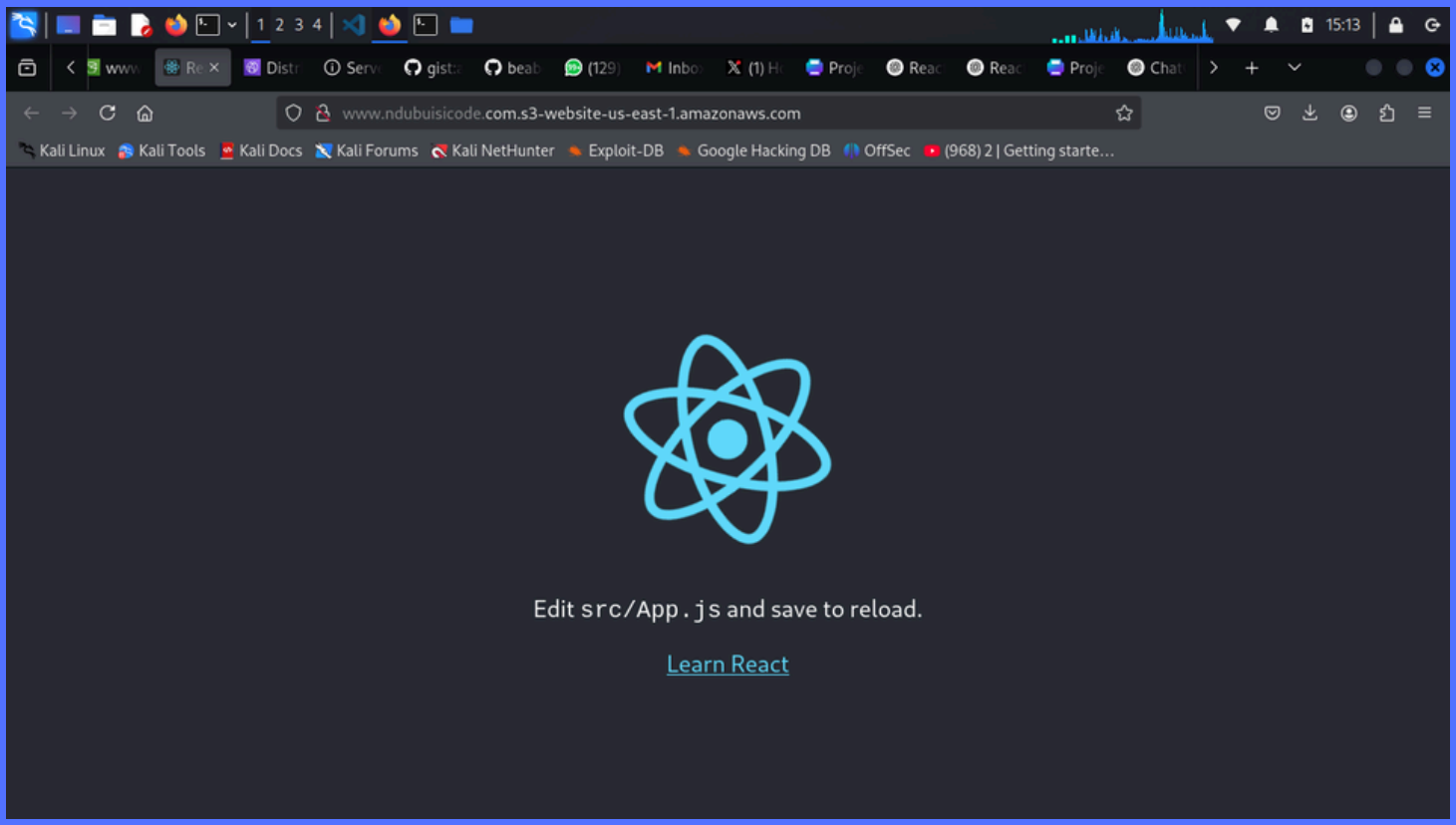
Set Up Our Certificate From Our Certificate Manager



Set Up Our Certificate From Our Certificate Manager







Summary

This project report details the deployment of a React application using AWS S3 and CloudFront. The goal was to create a scalable, cost-effective, and secure hosting solution.

- AWS S3 was used for storing static files, while CloudFront improved performance through global content distribution.
- The setup included IAM policies for security, CloudFront caching for optimization, and SSL/TLS encryption for secure access.
- Optional configurations, such as Route 53 for domain management and AWS Certificate Manager for SSL, were also considered.
- The deployment was optimized for performance, security, and cost-efficiency, making it a reliable solution for hosting static React applications.

This approach ensures minimal latency, reduced costs, and enhanced reliability for web applications.