Building a cloud-based Photo managemaent app with AWS S3

Today, we're embarking on an exciting hands-on project: building a cloud-based photo management application using AWS. This app will enable users to upload, view, and delete photos directly from their web browser, with all images securely stored in Amazon S3. Using a simple HTML and JavaScript interface, the app will interact with AWS services via the AWS SDK, providing a seamless connection to cloud storage.

This project is ideal for anyone interested in learning how web applications integrate with AWS services. By the end of this tutorial, you'll have a fully functional photo management app deployed in the cloud, ready to use.

Application Overview

The app features an intuitive and user-friendly interface. Users can select an image from their device and upload it to the cloud with just one click. Once uploaded, the image is displayed in a gallery that showcases all stored photos. Each image includes a delete button, allowing users to remove unwanted files instantly. Everything happens within the browser, eliminating the need for additional software and making photo management effortless.

Key Components

1. **Amazon S3**: The app uses Amazon S3 as the storage backend, ensuring secure and scalable storage for all uploaded photos.
2. **AWS SDK for JavaScript**: The app leverages the AWS SDK to handle interactions between the browser and AWS services, enabling seamless uploads, deletions, and retrievals.
3. **Amazon Cognito Identity Pool**: For secure access to AWS resources, the app uses Amazon Cognito to authenticate users and grant temporary permissions.
4. **S3 Static Website Hosting**: The app is hosted directly on S3 using its static website hosting feature, eliminating the need for a separate web server.

How It Works

This diagram visually represents how the **S3 Photo App** works using AWS services. On the left, a web browser executes a **JavaScript-based script**, which interacts with AWS services. The script communicates with the **Amazon SDK for JavaScript**, enabling seamless integration between the web application and AWS resources. The SDK acts as the bridge, sending requests to **Amazon Simple Storage Service (S3)**, where all uploaded photos and albums are stored.

S3 consists of **buckets**, which serve as storage containers. Inside each bucket, users can organize their files into **albums and individual photos**. The entire system allows users to upload, retrieve, and delete images directly from the web application while leveraging **AWS's scalability and security**. This architecture ensures a **serverless, highly available, and easily manageable photo storage solution**.

Business Case

The **S3 Photo App** is designed to provide a scalable, cost-effective, and highly available cloud-based photo storage and management solution. As digital media consumption grows, individuals and businesses require reliable ways to store, retrieve, and share images securely. Traditional storage solutions face limitations in scalability, redundancy, and maintenance costs, making cloud-based solutions the preferred approach. This application leverages **Amazon S3** as its core storage service, ensuring durability, accessibility, and seamless integration with modern cloud technologies.

Businesses across various industries, including media, e-commerce, and social platforms, rely on efficient storage systems to handle vast amounts of image data. With the rise of digital marketing, content creation, and user-generated media, there is a growing demand for storage solutions that offer instant access, automated processing, and cost optimization. The **S3 Photo App** addresses these needs by providing a seamless platform for users to upload, manage, and retrieve photos with minimal latency and maximum reliability.

Scalability is a key advantage of the **S3 Photo App**, allowing businesses to scale storage requirements dynamically without infrastructure constraints. Unlike traditional storage solutions that require hardware upgrades and capacity planning, Amazon S3 offers infinite scalability, ensuring businesses can handle traffic spikes and increased storage demands effortlessly. The application is designed to accommodate both small-scale users, such as freelance photographers, and large enterprises managing extensive image libraries.

Security is an essential aspect of photo storage, particularly for organizations handling sensitive or proprietary visual content. The **S3 Photo App** integrates **AWS security best practices**, including encryption, access control policies, and secure authentication mechanisms. By utilizing **AWS Identity and Access Management (IAM)**, the application ensures that only authorized users can access specific photo collections. Object-level

permissions provide fine-grained access control, making it suitable for businesses that need to manage multiple levels of user access.

Cost optimization is a critical factor in cloud storage adoption. Traditional on-premise storage solutions require high upfront investments and ongoing maintenance costs. The **S3 Photo App** takes advantage of Amazon S3's **tiered storage model**, automatically transitioning infrequently accessed photos to lower-cost storage classes such as **S3 Intelligent-Tiering or Glacier** for archival purposes. This results in significant cost savings while maintaining access to critical assets when needed.

Performance and user experience are optimized by leveraging **AWS CloudFront**, a global content delivery network that accelerates the retrieval of stored photos. Users experience faster load times and reduced latency, making the application ideal for high-traffic environments such as e-commerce websites, digital media platforms, and online portfolios. Integration with **AWS Lambda** allows real-time image processing, enabling features like automatic resizing, watermarking, and metadata extraction upon upload.

Disaster recovery and data resilience are seamlessly handled through **S3's built-in redundancy and versioning capabilities**. The application ensures that users never lose important visual content due to accidental deletions or system failures. Cross-region replication further enhances availability by maintaining copies of critical data in geographically distributed locations, ensuring business continuity in the event of regional outages.

The **S3 Photo App** presents a **strong business case** for companies looking to modernize their digital asset management infrastructure. It eliminates the complexities of maintaining physical servers, reduces storage costs through automation, enhances security, and improves accessibility. Whether for personal use, creative agencies, or large enterprises, this solution provides a **reliable, future-proof platform for storing, organizing, and delivering images with high availability and minimal operational overhead**

All user interactions occur within the browser, while the photos are securely stored and managed in the cloud.

Why This Project?

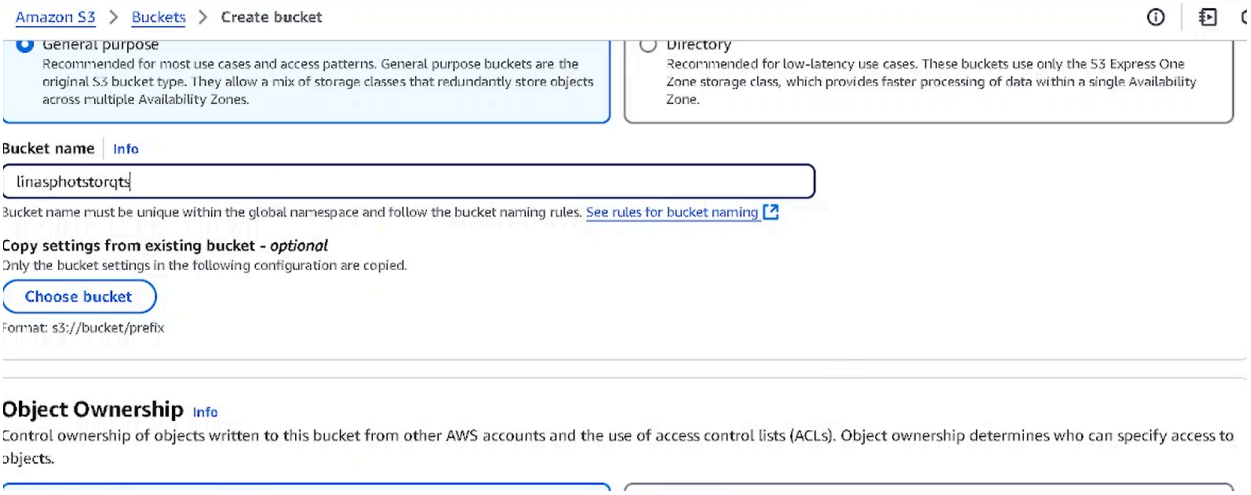
This project is a great way to gain hands-on experience with AWS services, including S3, Cognito, and the AWS SDK. It also demonstrates how to build a serverless web application that leverages cloud storage for scalability and security. By the end, you'll have a practical understanding of how to integrate AWS services into a web application and deploy it in the cloud.

Let’s get started and build your very own cloud-based photo management app!

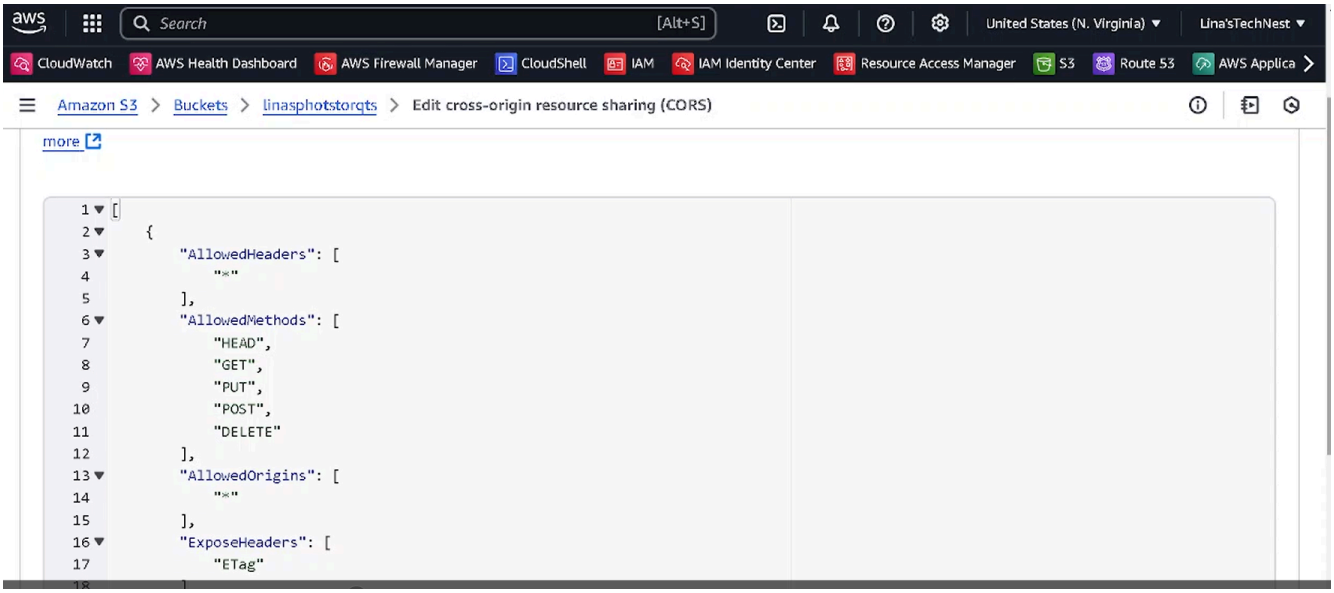
Step 1: Create an S3 Bucket for Storage

Start by creating an Amazon S3 bucket where images will be stored.

1. Open the AWS S3 Console and click **Create Bucket**.
2. Choose a unique bucket name (e.g.,linasphotostorqts).



1. Select the AWS region closest to your users for faster access.
2. Scroll to **Block Public Access settings**, uncheck **Block all public access**, and confirm the changes.
3. Navigate to the **Permissions** tab and update the bucket policy by copying the contents of S3-Bucket-permissions.json from the provided resources.
4. Update the **CORS configuration** by pasting the contents of CORS.json into the CORS editor and saving the changes.



1.

What is CORS and Why Do We Enable It for Amazon S3?

CORS (Cross-Origin Resource Sharing) is a security feature implemented by web browsers that controls how resources on a web page can be requested from a different domain. By default, web browsers enforce the **same-origin policy**, which prevents JavaScript from making requests to a different domain than the one that served the web page.

Since the **S3 Photo App** is a **browser-based application** that interacts with an **Amazon S3 bucket**, it needs permission to send and receive requests from a domain different from the bucket's origin. **This is where CORS comes in.**

Why Enable CORS for S3?

Without enabling CORS, the browser would **block requests** made by the JavaScript SDK when trying to upload, retrieve, or delete photos from the S3 bucket. Configuring **CORS on the S3 bucket** explicitly allows the browser to communicate with S3, ensuring the app functions properly.

By defining CORS rules, you specify:

1. **Which domains** (origins) are allowed to access the bucket.
2. **Which HTTP methods** (e.g., GET, POST, PUT, DELETE) are permitted.
3. **What headers** can be included in requests.
4. **Whether credentials** (such as authentication tokens) are allowed.

For this **S3 Photo App**, enabling CORS ensures that the **browser script can upload and retrieve photos** from S3 while maintaining security by allowing only specific trusted origins.

Step 2: Set Up Amazon Cognito for Authentication

Amazon Cognito: Enabling Secure and Scalable Access Control for the S3 Photo App

In any cloud-based application that interacts with **Amazon S3**, security and access control are fundamental concerns. Traditional authentication mechanisms require managing user credentials, which introduces risks associated with credential storage, unauthorized access, and scalability challenges. **Amazon Cognito** provides a **serverless, scalable, and cost-effective authentication solution** that allows applications to securely grant access to AWS services—including **Amazon S3**—without requiring a backend authentication system.

For the **S3 Photo App**, Amazon Cognito serves as the **identity broker**, managing authentication and authorization for users who need access to stored photos. This

architecture ensures **security, seamless user access management, and cost efficiency**, making it a **best practice in cloud-native applications**.

Business Case for Using Amazon Cognito in the S3 Photo App

The **S3 Photo App** allows users to **upload, view, and delete photos** stored in an **Amazon S3 bucket**. A key challenge in designing such a system is **managing access permissions** without compromising security or requiring a dedicated authentication backend. Traditional authentication models demand a separate database to store usernames and passwords, requiring encryption, access control policies, and infrastructure maintenance. This approach is expensive, difficult to scale, and introduces potential security vulnerabilities.

Amazon Cognito eliminates these complexities by providing **temporary AWS credentials**, allowing users to securely interact with the S3 bucket without direct IAM access. Cognito **Identity Pools** grant controlled access to AWS resources based on pre-configured IAM roles, making it an ideal choice for applications requiring scalable and secure access to **Amazon S3**.

How Amazon Cognito Powers the S3 Photo App

Amazon Cognito is used in two primary ways in this project:

1. **Providing Temporary AWS Credentials for Direct S3 Access**
    - Instead of hardcoding IAM credentials in the frontend, the S3 Photo App retrieves **temporary security tokens** from Cognito.
    - The application initializes **Cognito Identity Pools**, which assign an IAM role to each user, allowing them to interact with the S3 bucket securely.
2. **Supporting Both Guest (Unauthenticated) and Authenticated Users**
    - Cognito **allows guest users** (unauthenticated identities) to upload, view, and delete their own photos without requiring login credentials.
    - If authentication is later required, Cognito can seamlessly integrate with **Google, Facebook, AWS IAM, or a custom authentication provider**.

Step-by-Step Architecture Breakdown

1. **User Access and Authentication via Cognito**
    - When a user interacts with the S3 Photo App, the application first checks for an existing Cognito identity.
    - If the user is **unauthenticated** (guest), Cognito generates a **temporary identity** and grants access based on IAM policies.
    - If authentication is enabled, the user logs in via an **external identity provider (Google, Facebook, or AWS Cognito User Pools)**.
2. **IAM Role-Based Access Control**
    - The Cognito Identity Pool **maps users to IAM roles** that define access permissions for S3.
    - The IAM role allows actions such as **ListObjects, PutObject, and DeleteObject**, ensuring users can only access authorized resources.

    - Permissions are enforced **dynamically**, reducing the risk of unauthorized S3 interactions.
3. **Temporary Security Credentials**
    - Once authenticated, Cognito assigns **temporary AWS credentials** with a limited lifespan.
    - These credentials **eliminate the need for long-term access keys**, mitigating security risks.
    - Users interact with **Amazon S3 directly**, making API calls without passing through an intermediary backend.

Why Cognito is the Best Choice for This Project

1. **Security Without Hardcoding Credentials**
    - Traditionally, frontend applications required IAM user credentials, leading to exposure risks.
    - With Cognito, credentials are dynamically generated, **eliminating the need to store or expose access keys** in the frontend.
2. **Scalability Without a Backend**
    - Cognito **removes the need for a custom authentication system**, reducing development overhead and maintenance costs.
    - The service scales automatically to handle millions of users, making it a **future-proof solution**.
3. **Fine-Grained Access Control with IAM Roles**
    - Users only receive the **minimum required permissions** based on their session type (guest or authenticated).
    - Permissions can be adjusted dynamically **without modifying the frontend code**.
4. **Cost Efficiency**
    - Cognito **Identity Pools** are free to use for anonymous users, with costs only incurred when federating external identity providers.
    - Compared to managing an **authentication server**, Cognito significantly reduces operational expenses.

How This Benefits Enterprises and Startups Alike

For enterprises, Cognito provides an **enterprise-grade identity solution** that integrates with AWS IAM for compliance-driven access control. Organizations handling sensitive media files can configure **multi-factor authentication (MFA)**, enforce least-privilege policies, and apply compliance-driven security measures.

For startups, Cognito eliminates the complexity of building an authentication backend from scratch, allowing developers to focus on business logic rather than **security, access control, and credential management**. Since Cognito **scales automatically**, startups can handle rapid growth without provisioning additional infrastructure.

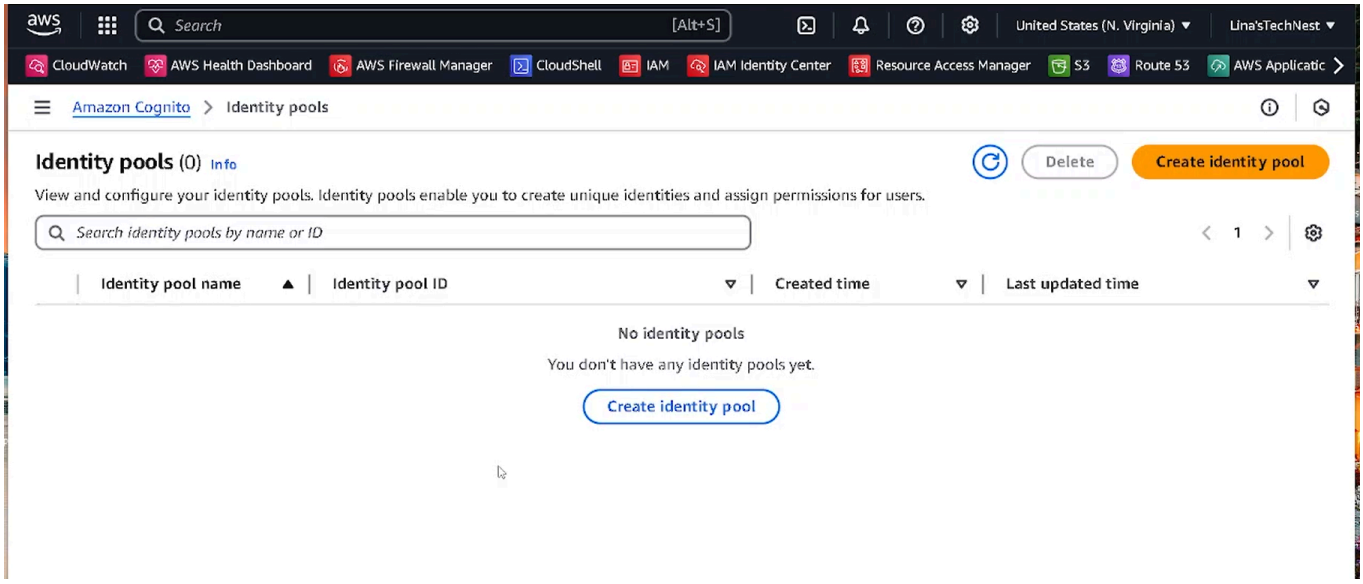
Conclusion: Why Amazon Cognito is Essential for the S3 Photo App

Amazon Cognito is the **best solution** for managing authentication and access control in the **S3 Photo App**. It ensures **security, scalability, and cost efficiency** while removing the complexity of managing IAM credentials in the frontend. By providing **temporary AWS credentials**, Cognito allows users to **interact with Amazon S3 directly** without exposing sensitive authentication information.

This **serverless, cloud-native approach** ensures that the application remains lightweight, highly available, and secure, making it a perfect example of **modern cloud architecture**. Whether for **individual developers, startups, or enterprises**, Cognito provides a **seamless authentication experience**, ensuring that users can **securely upload, view, and manage photos** in the cloud with minimal friction.

This project serves as a blueprint for how **cloud applications** should be designed—leveraging AWS services to deliver **high-performance, scalable, and secure solutions** without the overhead of traditional authentication systems

1. Open the AWS Cognito Console and select **Manage Federated Identities**.
2. Click **Create Identity Pool** and name it S3-identity-pool.



1.
2. Enable **unauthenticated identities** to allow public access.
3. Amazon Cognito provides two core mechanisms for granting access to AWS resources: **Guest Access (Unauthenticated Identities)** and **Authenticated Access**. These two access methods enable businesses to design applications that balance security, accessibility, and user experience, depending on their use case. When architecting a solution, the decision to enable guest or authenticated access is driven by business requirements, user engagement strategies, and security considerations.
4. **Guest Access (Unauthenticated Identities)**
5. Guest Access allows users to interact with AWS services without signing in. This is made possible through Cognito Identity Pools, which issue temporary AWS credentials to unauthenticated users. From a business perspective, this approach is beneficial in

scenarios where lowering friction for user interaction is a priority. For example, an application like the **S3 Photo App** can leverage guest access to enable users to upload, view, and delete photos without needing to create an account or authenticate via third-party identity providers.

6. This model is ideal for applications that:

- Provide public access to resources such as media files, product catalogs, or event listings.
- Allow seamless interaction without requiring a login, increasing user engagement.
- Need temporary access to AWS services with controlled permissions.
- Want to reduce user onboarding friction by not forcing authentication for simple interactions.

7. From a security standpoint, guest access requires **strict IAM policies** to limit what unauthenticated users can do. For instance, in an S3-based application, a role could be configured to allow unauthenticated users to upload photos but **not delete** or modify resources beyond their own uploads. Such policies enforce security while still enabling a frictionless user experience.

8. **Authenticated Access**

9. For applications that require stronger identity verification and personalized user experiences, **Authenticated Access** is the preferred approach. Cognito allows authentication via multiple identity providers:
10. **Amazon Cognito User Pools** – Users sign up and sign in using a built-in user directory managed by Cognito.
11. **Social Identity Providers** – Users can authenticate using Facebook, Google, Apple, or Twitter.
12. **Enterprise Identity Providers** – Applications can integrate OpenID Connect (OIDC) or SAML-based providers for single sign-on (SSO) capabilities.
13. Authenticated access is essential in use cases where:

- Users need to save preferences or access private data.
- Secure access control is required for different roles (e.g., admins vs. regular users).
- Organizations require corporate SSO for employees.
- Applications demand multi-factor authentication (MFA) for increased security.

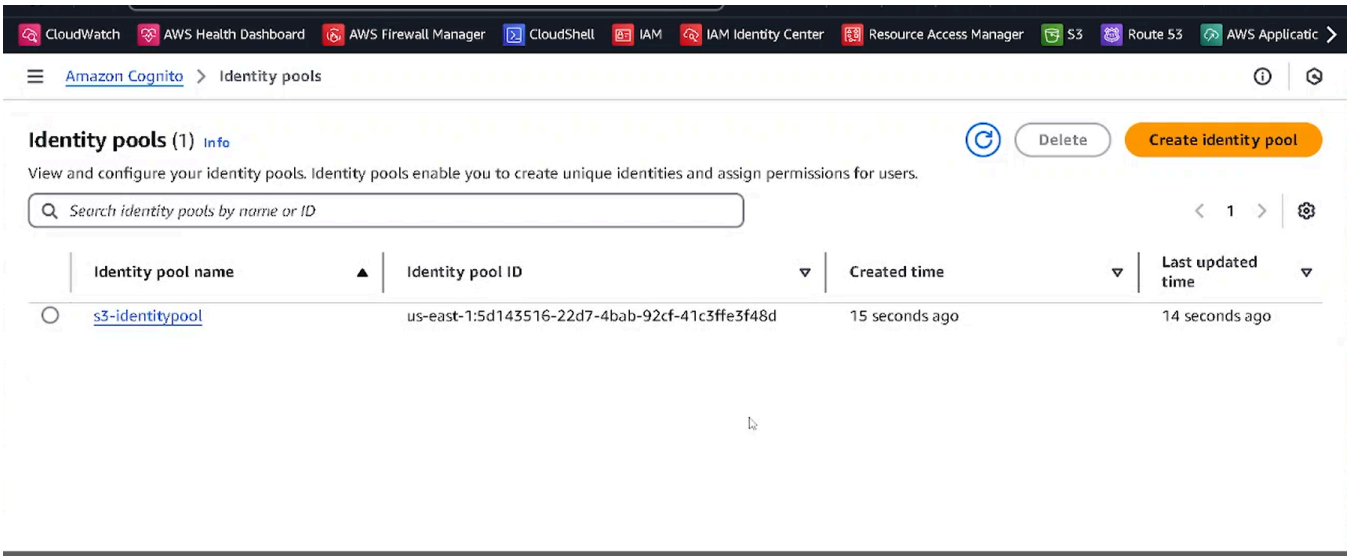
14. When a user logs in using an authentication provider, Cognito exchanges their authentication tokens for AWS credentials. This allows fine-grained access control through **IAM policies** tied to their Cognito identity. In contrast to guest access, authenticated users can have broader privileges, such as managing albums, modifying permissions, or retrieving specific resources assigned to their account.

15. **Business Implications of Choosing Guest vs. Authenticated Access**

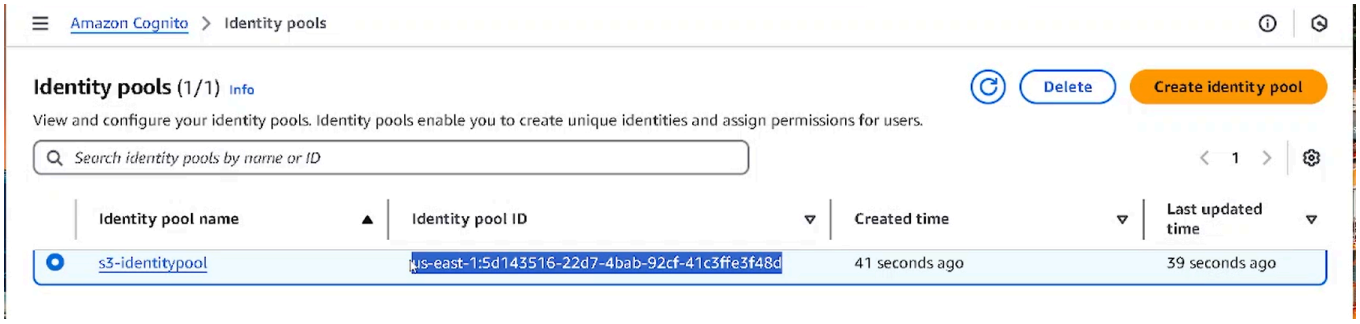
16. The choice between guest and authenticated access has direct business implications. Guest access improves accessibility and engagement by removing authentication barriers, making it well-suited for content-sharing apps, public-facing services, or platforms that want to encourage interaction without commitment. However, guest access should be **restricted using IAM policies** to prevent abuse.
17. Authenticated access, on the other hand, ensures security, enables **personalization**, and allows deeper integration with **other AWS services**, such as AWS Lambda for backend processing, DynamoDB for user-specific data storage, and API Gateway for secure

interactions. While it requires more setup and potential friction during onboarding, it offers greater **control, security, and user engagement**.

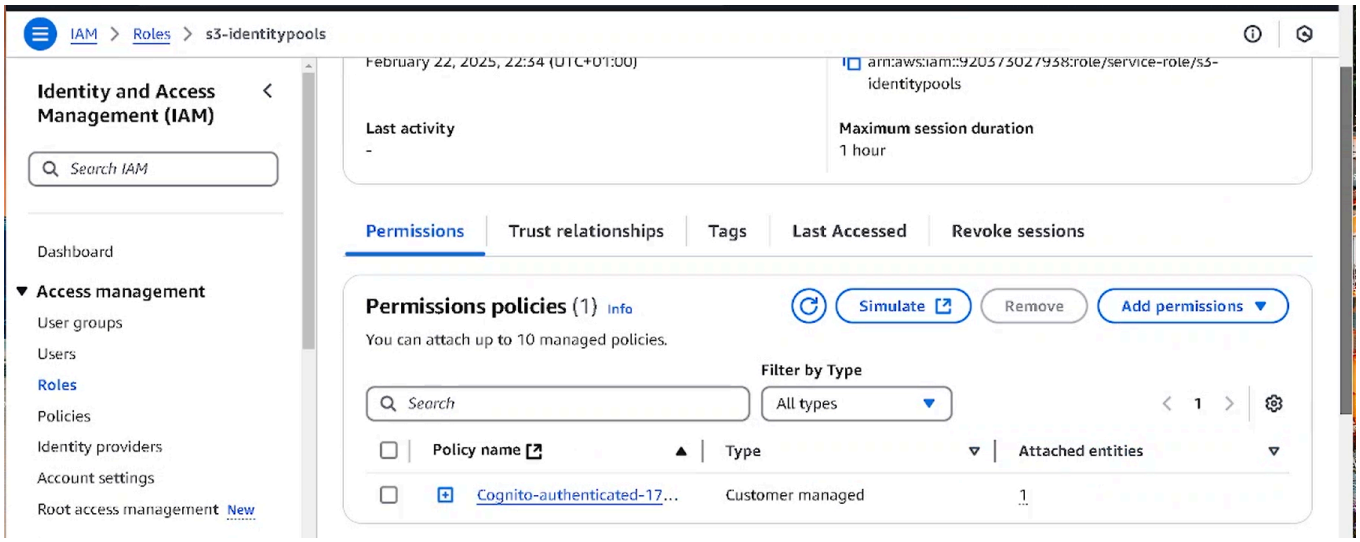
18. A hybrid approach is often the best solution, where guest access is allowed for basic interactions, and users can **optionally sign in** for enhanced functionality. For instance, an S3 Photo App might allow unauthenticated users to upload photos but require authentication to create albums, manage images, or access certain features.
19. **Final Recommendation**
20. For this project, enabling **Guest Access** is the right choice if the goal is to allow easy interaction with the S3 bucket without authentication. However, if long-term scalability and **personalization** are needed, integrating **Authenticated Access** through Cognito User Pools or social logins should be considered as a future enhancement. Implementing IAM policies that **restrict permissions based on user identity** will be critical in maintaining security, whether using guest or authenticated access.
21. In an enterprise setting, **federated authentication** with OpenID Connect or SAML could be leveraged to enable seamless single sign-on for employees, ensuring compliance and security at scale. This would be a more advanced implementation suited for corporate applications rather than public-facing consumer apps.
22. By understanding these access models, businesses can design **highly secure, scalable, and user-friendly** AWS-based applications that align with their goals and security requirements.
23.
24. Click **Create Pool**, then **Allow** to generate default IAM roles.



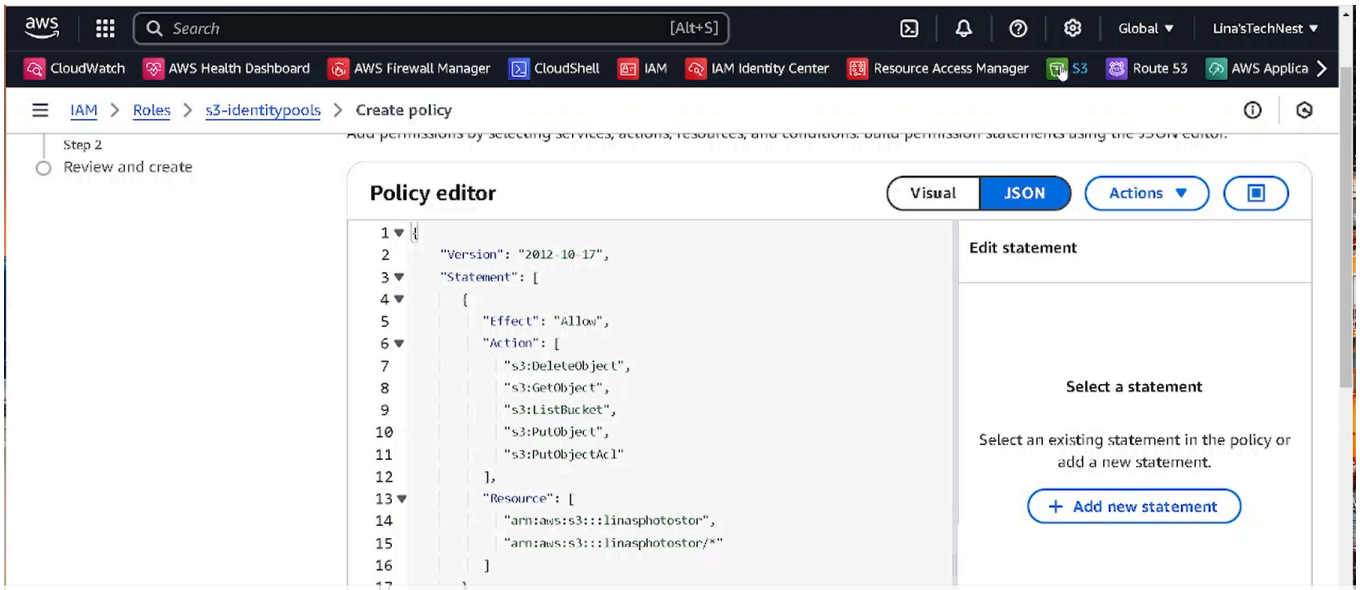
1.
2. Note the **Identity Pool ID** for later use.



1.
2. Open the AWS IAM Console and locate the IAM role for unauthenticated users (Cognito_S3identitypoolUnauth_Role).



1. Create a new inline policy and paste the JSON contents from unauth_role.json, replacing "BUCKETNAME" with your S3 bucket name.



1. Save the policy and apply it to the role.

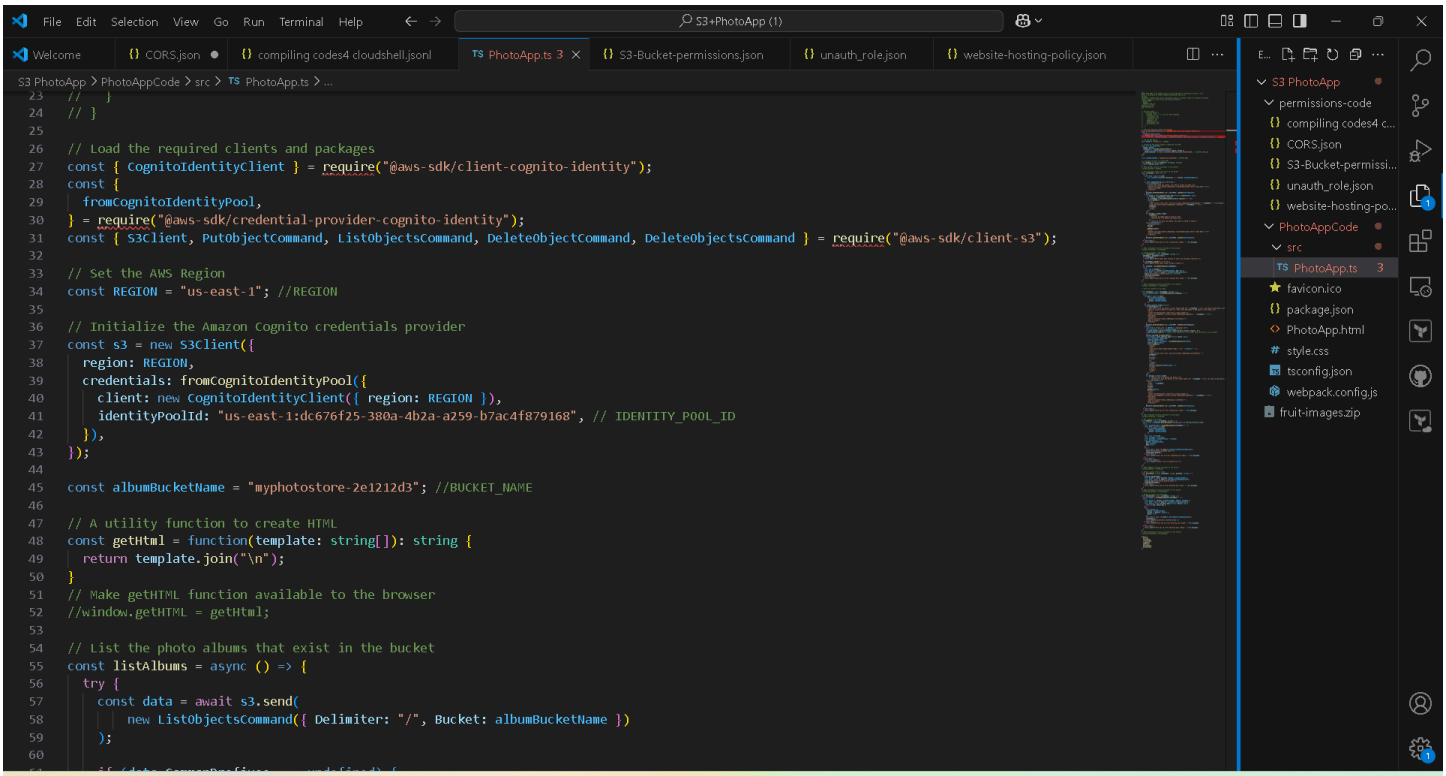
Step 3: Prepare the Application Code

You will need to configure and compile the JavaScript and HTML code.

1. Download the provided **S3 Photo App** resources.
2. Open PhotoApp.ts in **Visual Studio Code**and update:
   - **AWS Region** on line 34.

- **Identity Pool ID** on line 41.
- **S3 Bucket Name** on line 45.

3. Save the file and exit.

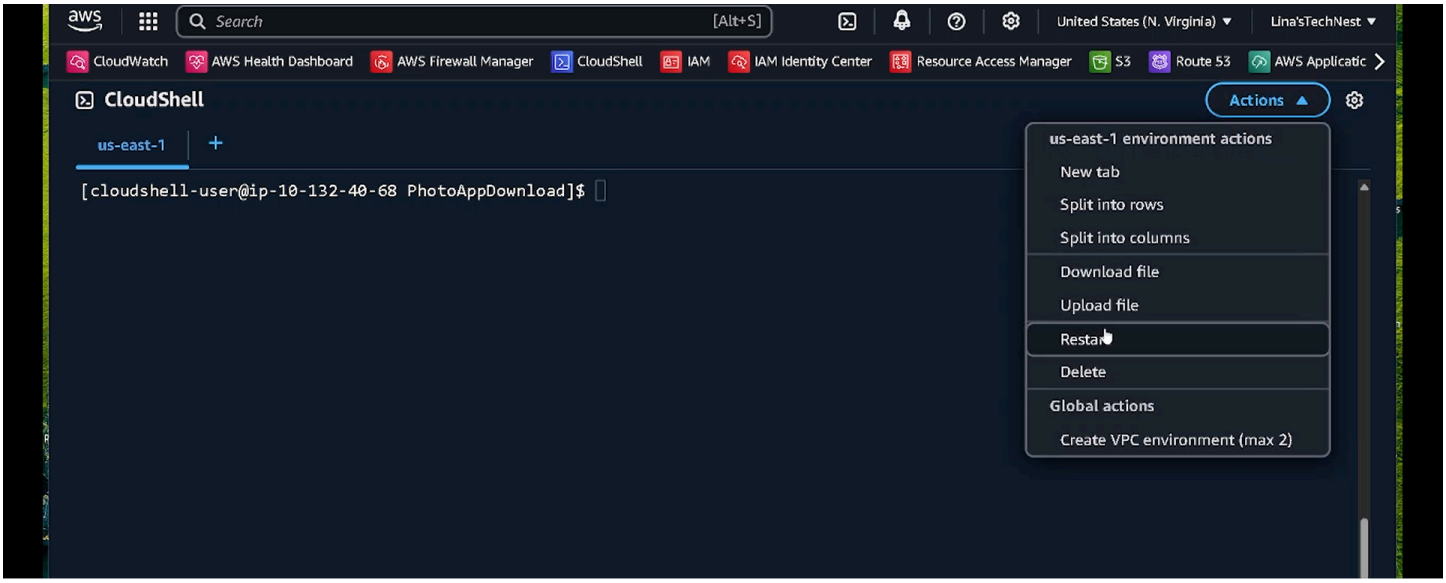


1. Open AWS CloudShell

- From the AWS Management Console, click the **CloudShell icon** in the top navigation bar.
- Wait for the terminal to load.

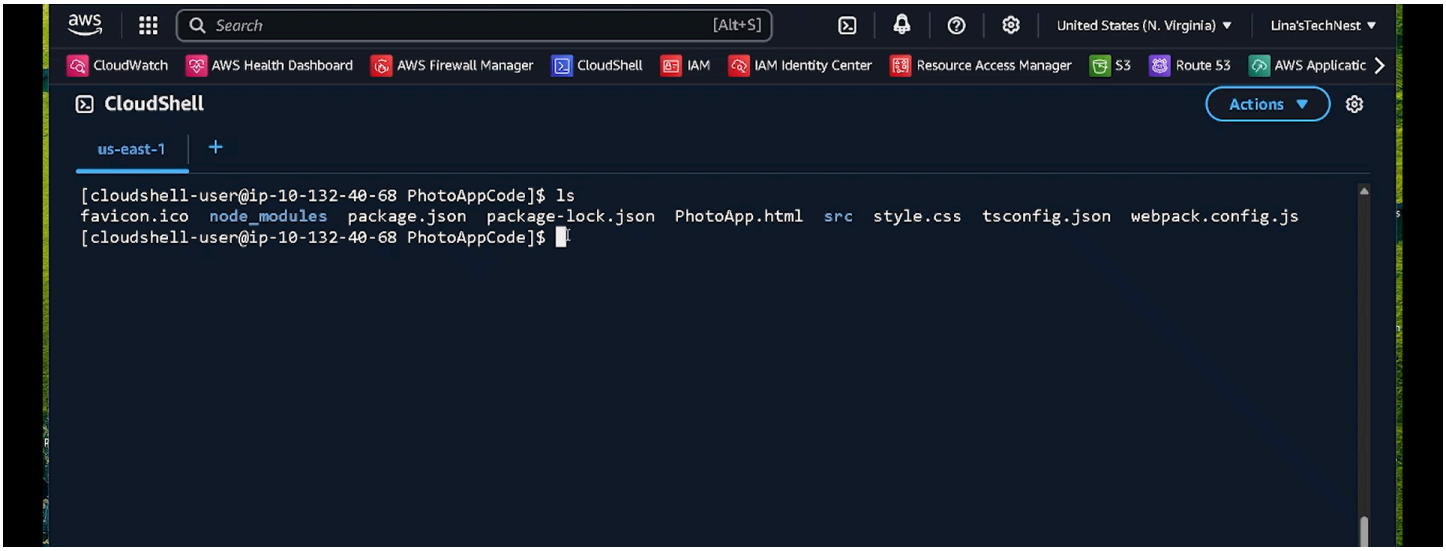
2. Upload the Code

- In CloudShell, click the **"Actions"** dropdown → choose **"Upload file"**.



-

- Select your PhotoAppCode.zip file and upload it.
- Confirm the file was uploaded using ls



-

3. Unzip the Code

Extract the zip file, by typing,

unzip PhotoAppCode.zip

Enter the extracted folder by entering,

cd PhotoAppCode

If You Encounter a Permission Denied Error:

You may see something like EACCES: permission denied, mkdir ...

To fix it, run

sudo npm install

Or fix permission issues permanently

sudo chown -R $USER:$GROUP ~/.npm

If npm still fails, delete the lock file and node_modules

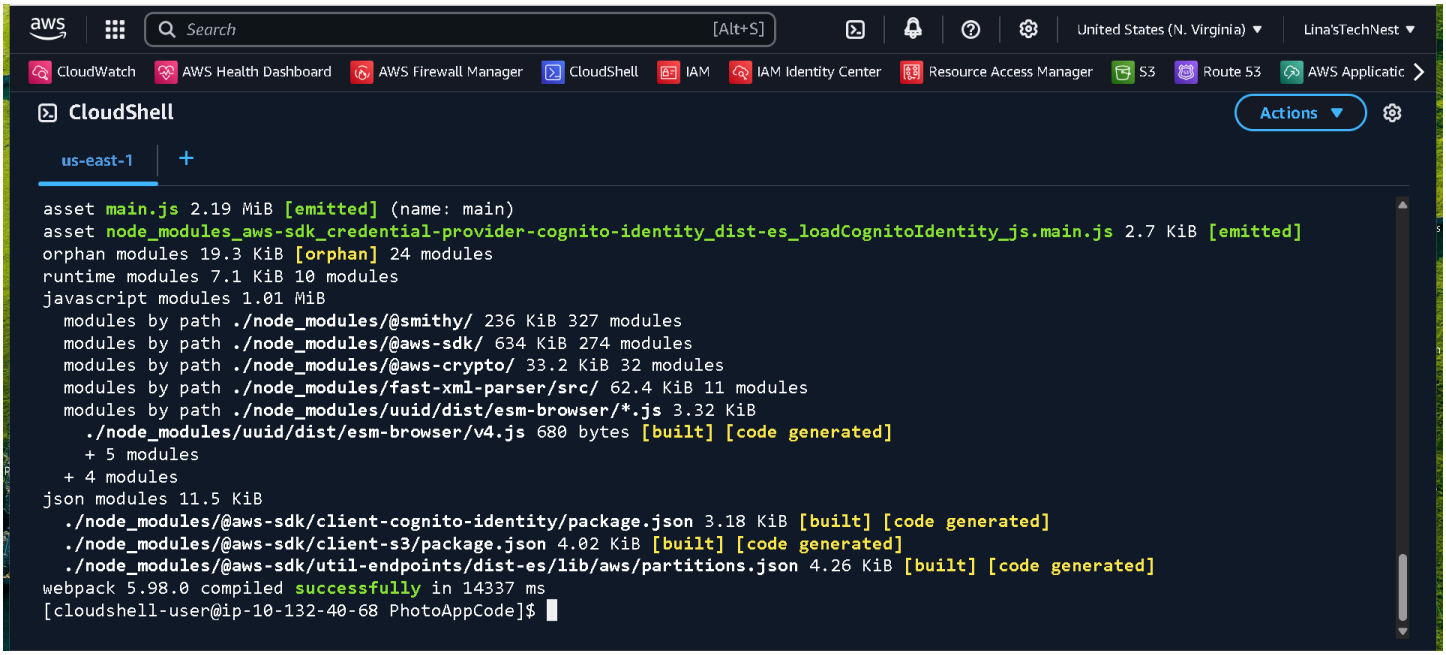
rm -rf node_modules package-lock.json

npm instal

Compile the Application

Once dependencies are installed, run the Webpack build command

npm run build



This will generate a dist/ folder with your compiled JavaScript.

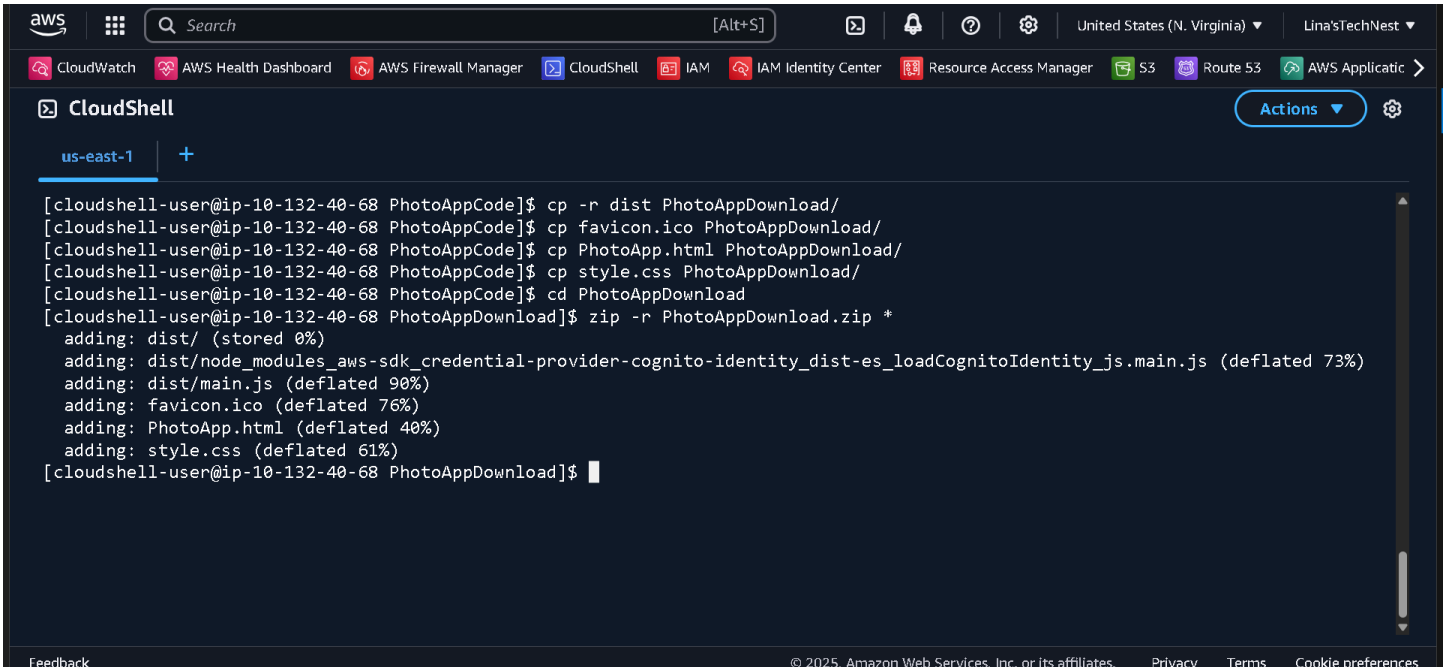
If you're preparing for production:

npm run build --mode=production

. Prepare Files for Deployment

Create a new folder for the deployable files:

mkdir PhotoAppDownload



Copy the compiled files and static assets into it

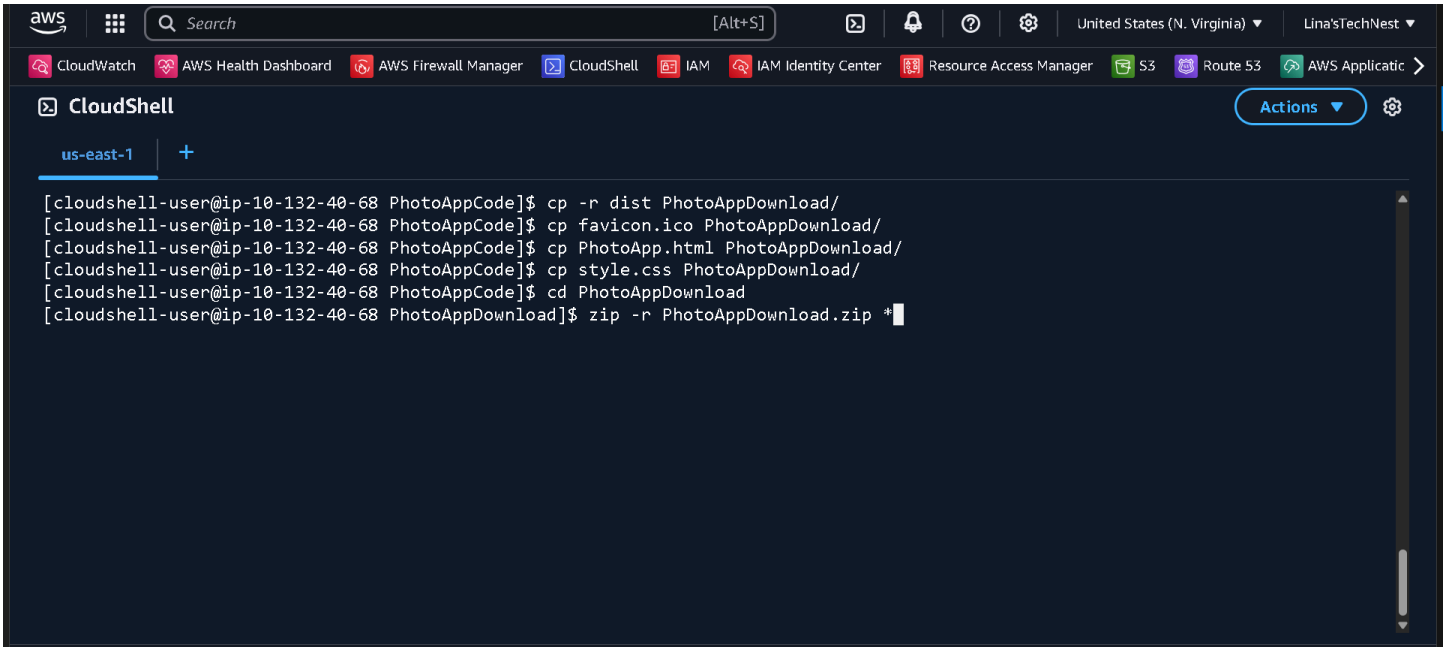
cp -r dist PhotoAppDownload/

cp favicon.ico PhotoAppDownload/

cp PhotoApp.html PhotoAppDownload/

cp style.css PhotoAppDownload/

Zip the Folder



Navigate into the deployment folder

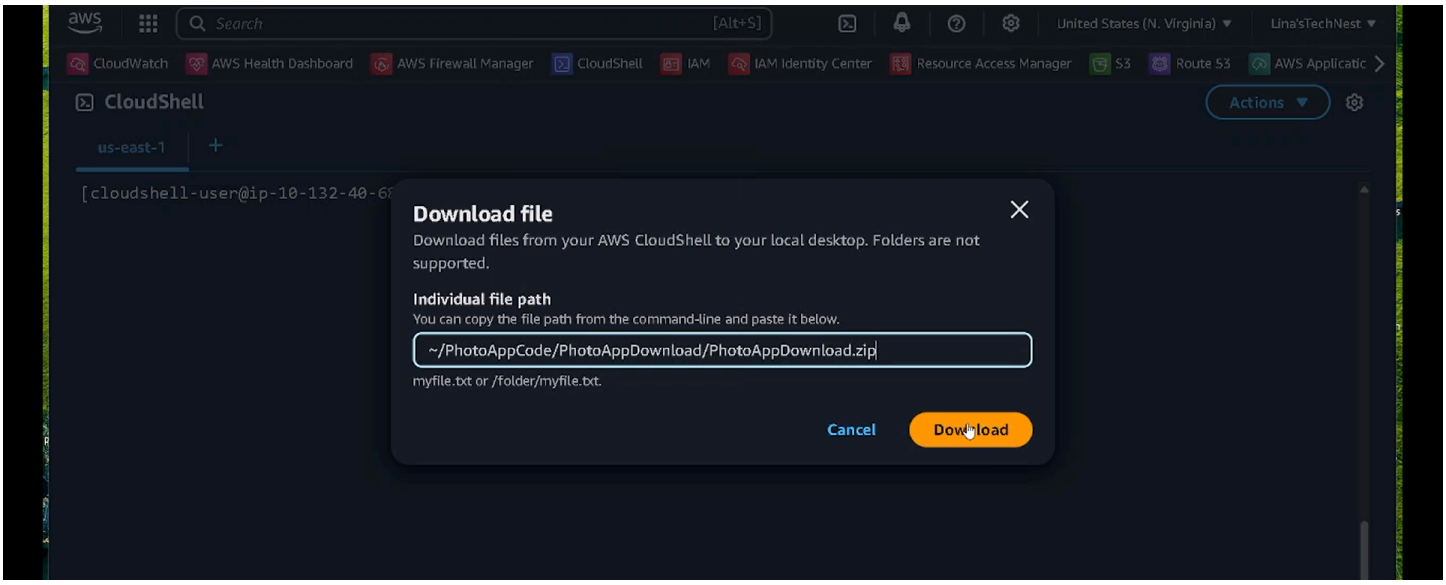
cd PhotoAppDownload

Create a zip archive

zip -r PhotoAppDownload.zip

Download the Compiled Project

To download to your local machine:



- In CloudShell, click **Actions > Download file**.
- Enter the full path to the zip file

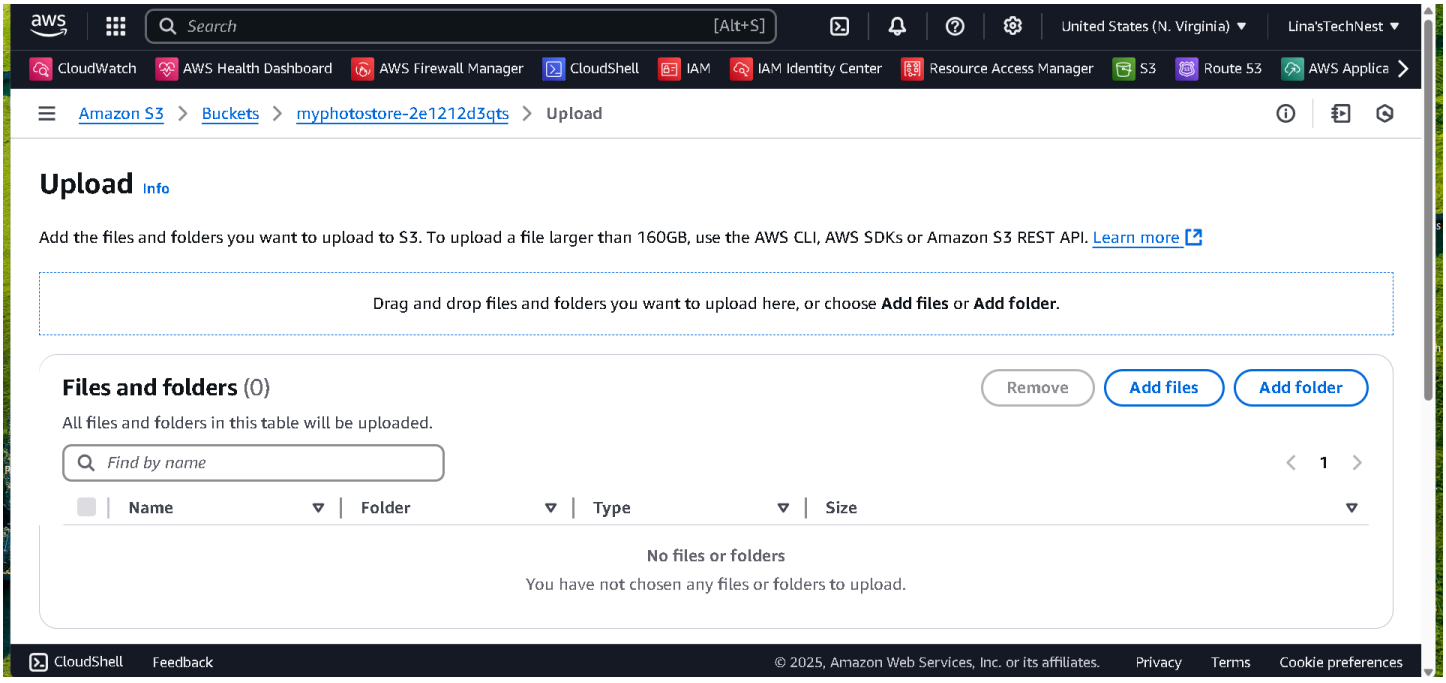
/home/cloudshell-user/PhotoAppDownload/PhotoAppDownload.zip

This downloads your compiled and ready-to-deploy app

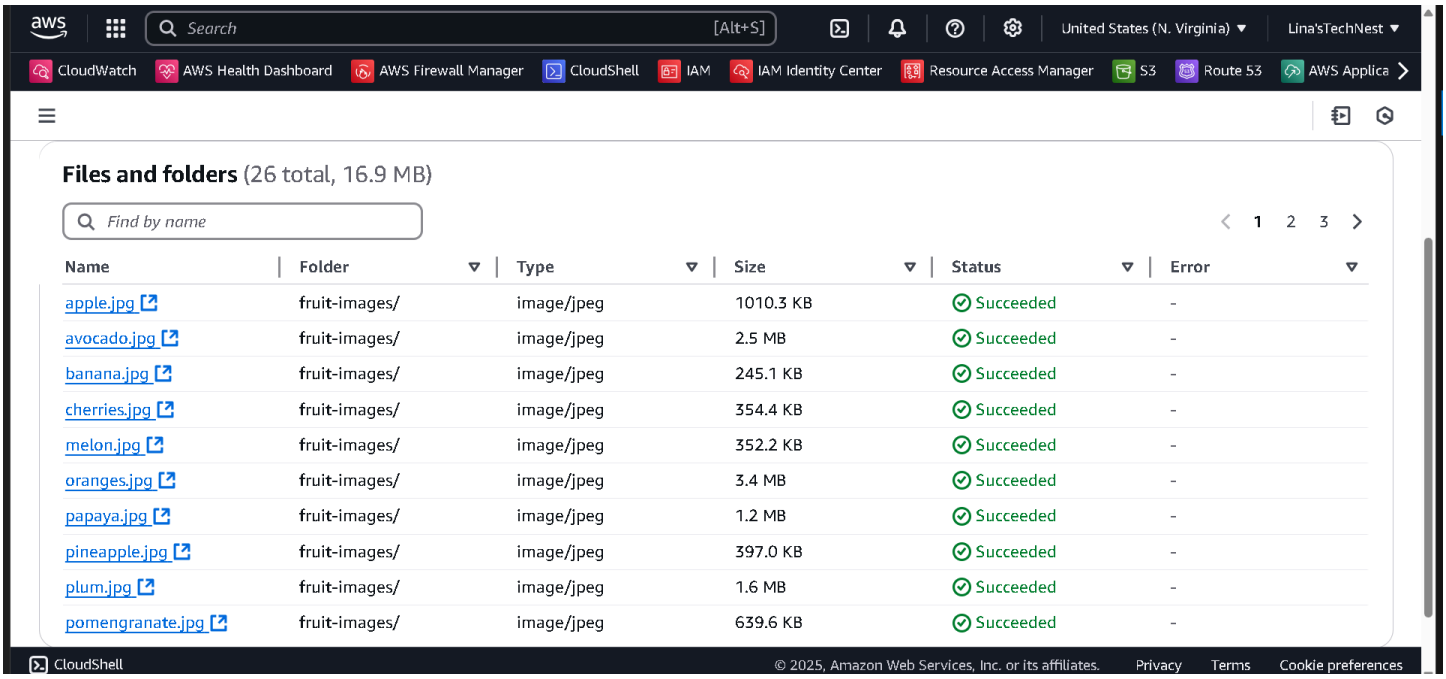
Step 5: Deploy the Application as a Static Website

To make the application accessible, host it using Amazon S3.

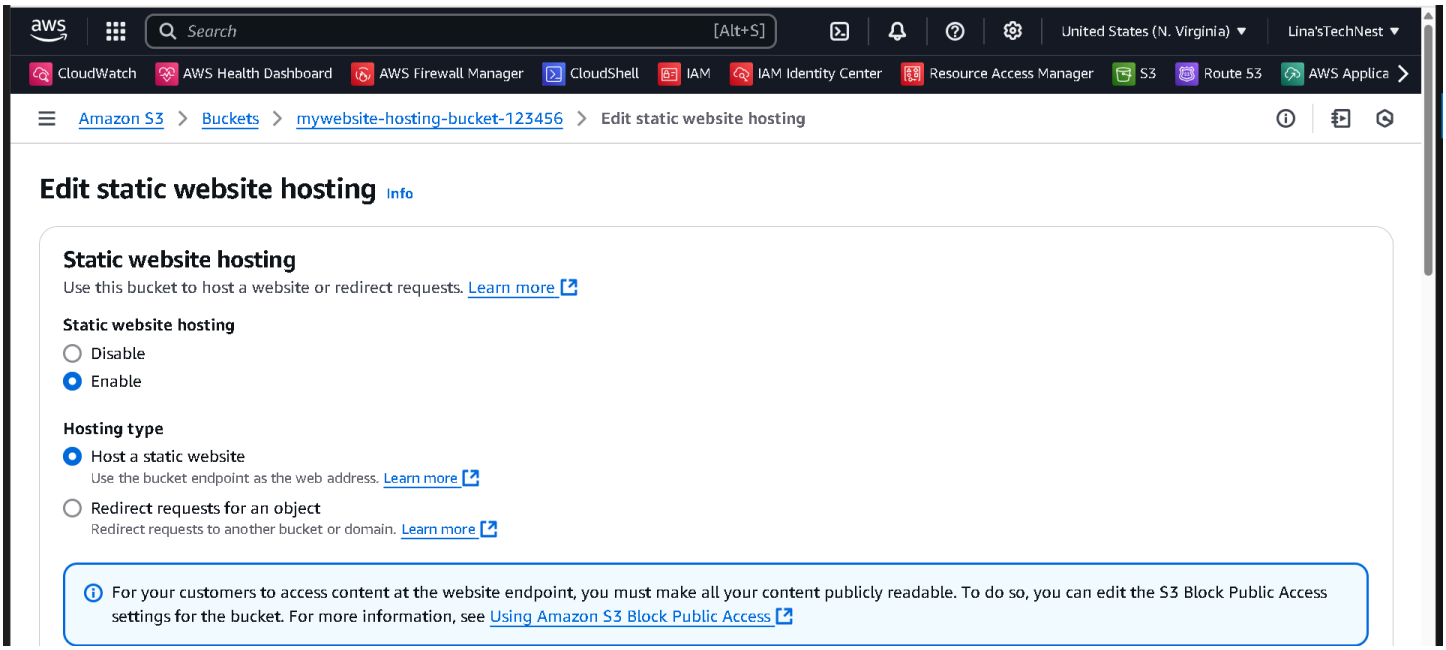
1. Open the S3 Console.



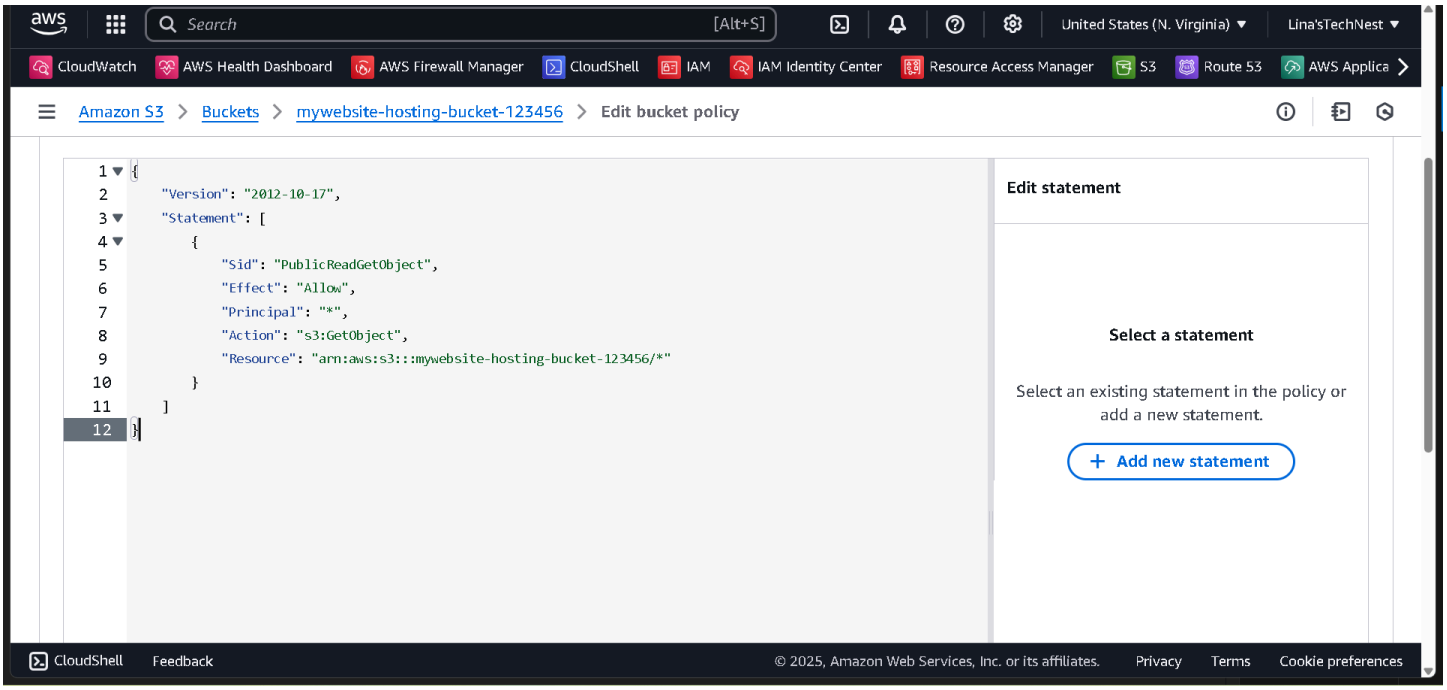
1.
2. Create a new bucket named myphotostore-2e1212d3qts.
3. Disable **Block all public access** and confirm the change.
4. Upload all files from the PhotoAppDownload folder to the bucket.



1.
2. Go to the **Properties** tab and enable **Static website hosting**.



1.
2. Set PhotoApp.html as the index document and save the changes.
3. Open the website-hosting-policy.json file from the resources and edit the bucket **ARN**.

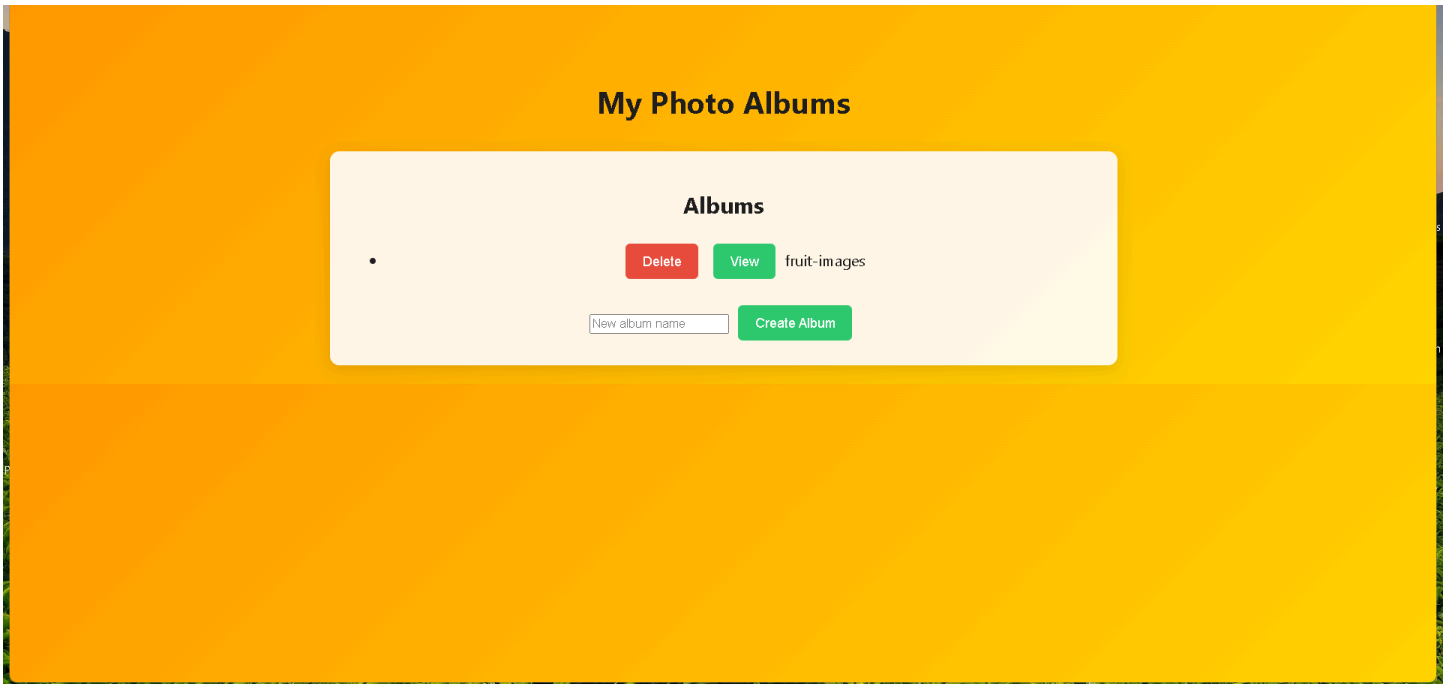


1.
2. Apply the updated policy to the S3 bucket.

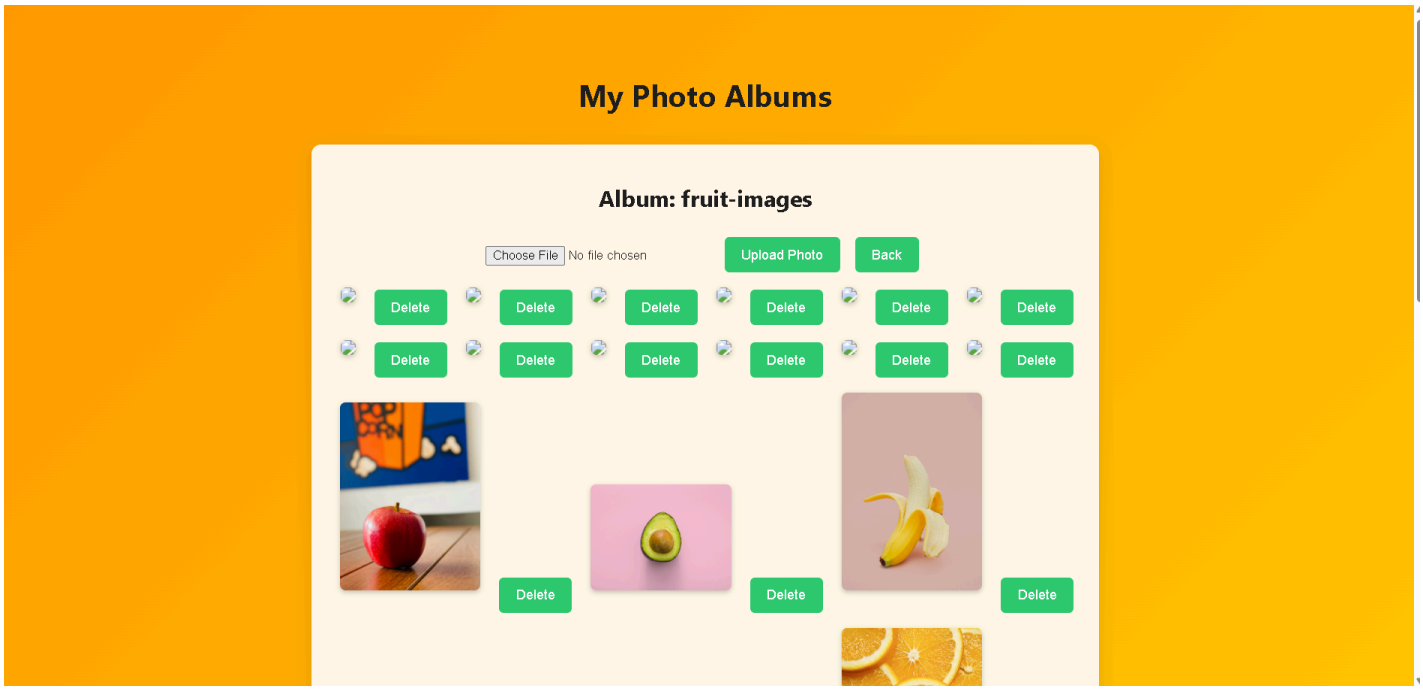
Step 6: Test the Application

Once deployed, access the **static website endpoint** from the S3 bucket properties.

1. Open the link and check if the application loads.



1.
2. Try uploading an image and verify that it appears in the S3 bucket.
3.



1. 2. **Create an Album**

- On the homepage, click **“Create Album”**.
- Enter a name like my-album and confirm.

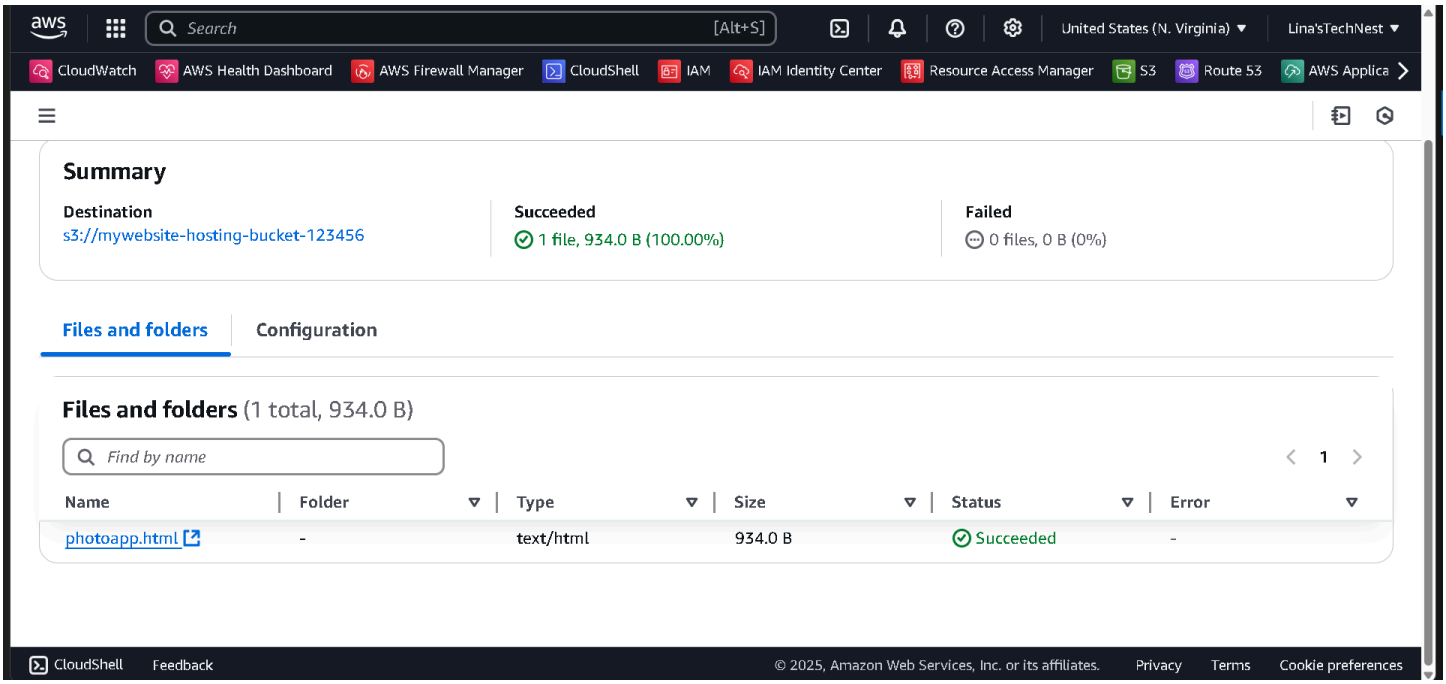
2. 3. **Add a Photo**

- After the album is created, you'll be redirected to the album page.
- Click **“Choose File”** or the file input field to upload an image from your computer.
- Click **“Add Photo”**.

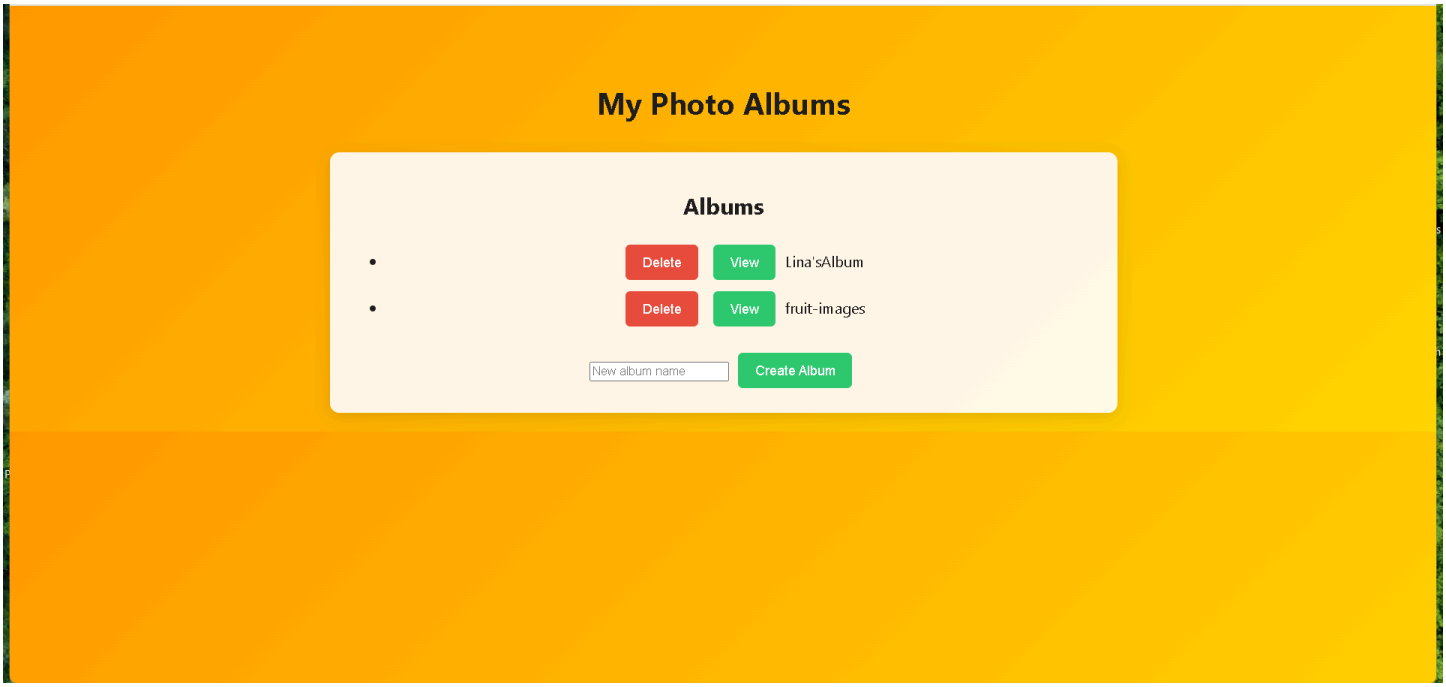
3. 4. **Verify in S3 Bucket**

- Go to the **Amazon S3 Console**.
- Open the **S3 bucket** you used to store the albums (this is the first bucket you created and enabled CORS on).
- You should see a **folder with the album name** you just created.
- Click into it — your uploaded image should be listed there.

4.



1.
2. Test the delete functionality to ensure photos can be removed.
3. **Go to your S3 website URL** and open the app in your browser.
4. **Navigate to the album** where you uploaded the photo.
5. Each image displayed should have a red **“X”** next to it.
6. Click the **“X”** on one of the photos to delete it.
7. The photo should immediately disappear from the UI.
8. To confirm, go back to the **S3 console**, open your bucket, navigate into the corresponding album folder, and ensure the photo is no longer listed
9.



1.