

Serverless File Sharing Applications

A Project Report Submitted in Partial Fulfillment of the Requirements for the
Award of the Degree of Bachelor of Science (Honors) in Computer Science
with Specialization in Cloud Computing and Big data

Submitted by
Ms. Devika. G
(R21DB012)

Under the guidance of

Internal Guide
Dr. Sasi Kala. G
External Guide
Mr. Priyadarshi Sir



SCHOOL OF COMPUTER SCIENCE AND APPLICATIONS

Kattigenahalli, Yelahanka, Bengaluru-560 064
www.reva.edu.in

December 2023

CERTIFICATE

This is to certify that the Minor project work entitled “SERVERLESS FILE SHARING APPLICATIONS” submitted to the School of Computer Science and Applications, REVA University in partial fulfilment of the requirements for the award of the Degree of Bachelor of Science (Research) in Computer Science with specialization in Cloud Computing and Big data in the academic year 2023-2024 is a record of the original work done by Devika (R21DB012) and Puneeth Raj (R21DB048) under our supervision and guidance and that this Minor project work has not formed the basis for the award of any Degree / Diploma / Associate ship / Fellowship or similar title to any candidate of any University.

Place: Bangalore

Date:

Internal Guide Signature:

External Guide Signature:

Signature of the Program Co-Ordinator

Signature of the Director

Submitted for the University Examination held on _____

Internal Examiner

External Examiner

DECLARATION

I, DEVIKA.G (R21DB012) and PUNEETH RAJ. K (R21DB048) hereby declare that this Minor project work entitled “SERVERLESS FILE SHARING APPLICATION” under the guidance of Dr. Sasi Kala and MR. Priyadarshi and that this Minor project work has not formed the basis for the award of any Degree / Diploma / Associate ship / Fellowship or similar title to any candidate of any University.

Signature of the Candidate

Date:

Countersigned by

Signature of the Internal guide

Signature of the External guide

TABLE OF CONTENTS

| CHAPTERS | PAGE NO |
|----------------------------------|--------------|
| 1. INTRODUCTION | 5 |
| 1.1 INTRODUCTION TO THE PROJECT | |
| 1.2 STATEMENT OF THE PROBLEM | |
| 2. SYSTEM ANALYSIS | 6-7 |
| 2.1 EXISTING SYSTEM | |
| 2.2 PROPOSED SYSTEM | |
| 3. TOOLS AND TECHNOLOGIES | 8-9 |
| 3.1 HARDWARE REQUIREMENT | |
| 3.2 SOFTWARE REQUIREMENT | |
| 4. SYSTEM DESIGN | 10-12 |
| 4.1 ARCHITECTURE DIAGRAM | |
| 4.2 SYSTEM DEVELOPMENT STRATEGY | |
| 5. IMPLEMENTATION STEPS | 13-14 |
| 6. TESTING | 15-18 |
| 7. SNAPSHOTS | 19-50 |
| 8. CONCLUSION | 51 |
| 9. BIBILOGRAPHY | 52 |

INTRODUCTION

1.1 INTRODUCTION TO THE PROJECT

In today's fast-paced digital landscape, efficient and secure file sharing is a cornerstone of modern collaboration. Our project centers on the creation of a serverless file sharing application using AWS services. This application is designed to offer users a straightforward yet robust platform for sharing files, bolstered by the scalability and security inherent in serverless architecture. With the increasing demand for remote work and seamless data exchange, our project aims to streamline file sharing, promoting productivity and ease of use in the cloud.

1.2 STATEMENT OF THE PROBLEM

In an era characterized by remote work and digital collaboration, the existing methods for file sharing often lack the agility, security, and user-friendliness required for modern workflows. Traditional file-sharing solutions may be cumbersome to manage, prone to security vulnerabilities, and lacking in scalability. This project addresses the pressing need for an advanced file-sharing platform that leverages serverless architecture and AWS services to provide an efficient, secure, and user-centric solution for file sharing in the cloud.

SYSTEM ANALYSIS

2.1 Existing System

The current landscape for file sharing predominantly relies on traditional methods, including on-premises servers and basic cloud storage solutions. These systems often suffer from scalability issues, security concerns, complex infrastructure requirements, and limited usability. Additionally, they may not adequately support remote work and collaboration, leading to performance bottlenecks and increased costs over time. To address these limitations, there is a growing need for a more agile, secure, and user-friendly file-sharing solution. This project seeks to develop a serverless file sharing application using AWS services, offering a modern and efficient alternative to traditional systems.

2.2 Proposed System

The proposed system represents a paradigm shift in file sharing, introducing a serverless architecture underpinned by AWS services. Unlike traditional methods, this system offers a highly adaptable, secure, and user-centric approach to file sharing in the cloud. It overcomes the limitations of existing systems by providing effortless scalability, robust security features including user authentication and access control, and an intuitive and visually appealing user interface.

At its core, this system leverages the power of AWS services such as Amazon DynamoDB for efficient metadata management, Amazon S3 for reliable and scalable file storage, and AWS Lambda for serverless computing. By harnessing these capabilities, the proposed system streamlines file sharing processes and enhances user productivity, making it an ideal solution for modern work environments characterized by remote collaboration and digital interaction. With this innovative approach, the proposed system not only addresses the shortcomings of the existing systems but also sets a new standard for cloud-based file sharing, emphasizing ease of use, security, and scalability.

Advantages of the Proposed System

Scalability: The proposed system offers seamless scalability to accommodate growing data and user demands, ensuring optimal performance at all times.

Security: Robust security measures, including user authentication and access control, protect sensitive data from unauthorized access.

Usability: A modern and intuitive user interface enhances the overall user experience, making file sharing effortless and efficient.

Cost-Efficiency: By leveraging AWS serverless architecture, the system minimizes operational costs, making it a cost-effective solution.

Remote Collaboration: The system supports remote work and collaboration, allowing geographically dispersed teams to share and access files with ease.

High Availability: Utilizing AWS resources ensures high availability and redundancy, reducing the risk of downtime and data loss.

Streamlined Processes: The system streamlines file sharing processes, increasing productivity and user satisfaction.

Innovation: It sets a new standard for cloud-based file sharing, emphasizing modernization, security, and scalability.

TOOLS AND TECHNOLOGIES

3.1 HARDWARE REQUIREMENTS:

1. Computer: Your personal laptop is your primary hardware requirement. Ensure that it meets or exceeds the following specifications:

- Processor: A multi-core processor (e.g., Intel Core i5 or equivalent) for faster execution of development tasks.
- RAM: At least 8 GB of RAM to handle development tools and virtual machines.
- Storage: An SSD (Solid-State Drive) is recommended for faster read/write speeds and improved performance.
- Operating System: Your laptop should run a compatible operating system, such as Windows, macOS, or Linux.

2. Internet Connection: A reliable and high-speed internet connection is essential for accessing AWS services, downloading software updates, and collaborating with team members if applicable. A stable internet connection ensures that you can work efficiently with cloud resources.

3. External Monitor (Optional): While not a strict requirement, having an external monitor can significantly enhance your productivity by providing more screen real estate for multitasking and working with multiple AWS services and development tools simultaneously.

4. Keyboard and Mouse: Invest in a comfortable keyboard and mouse, especially if you plan to work extensively on your laptop. Ergonomic accessories can improve your overall comfort and productivity.

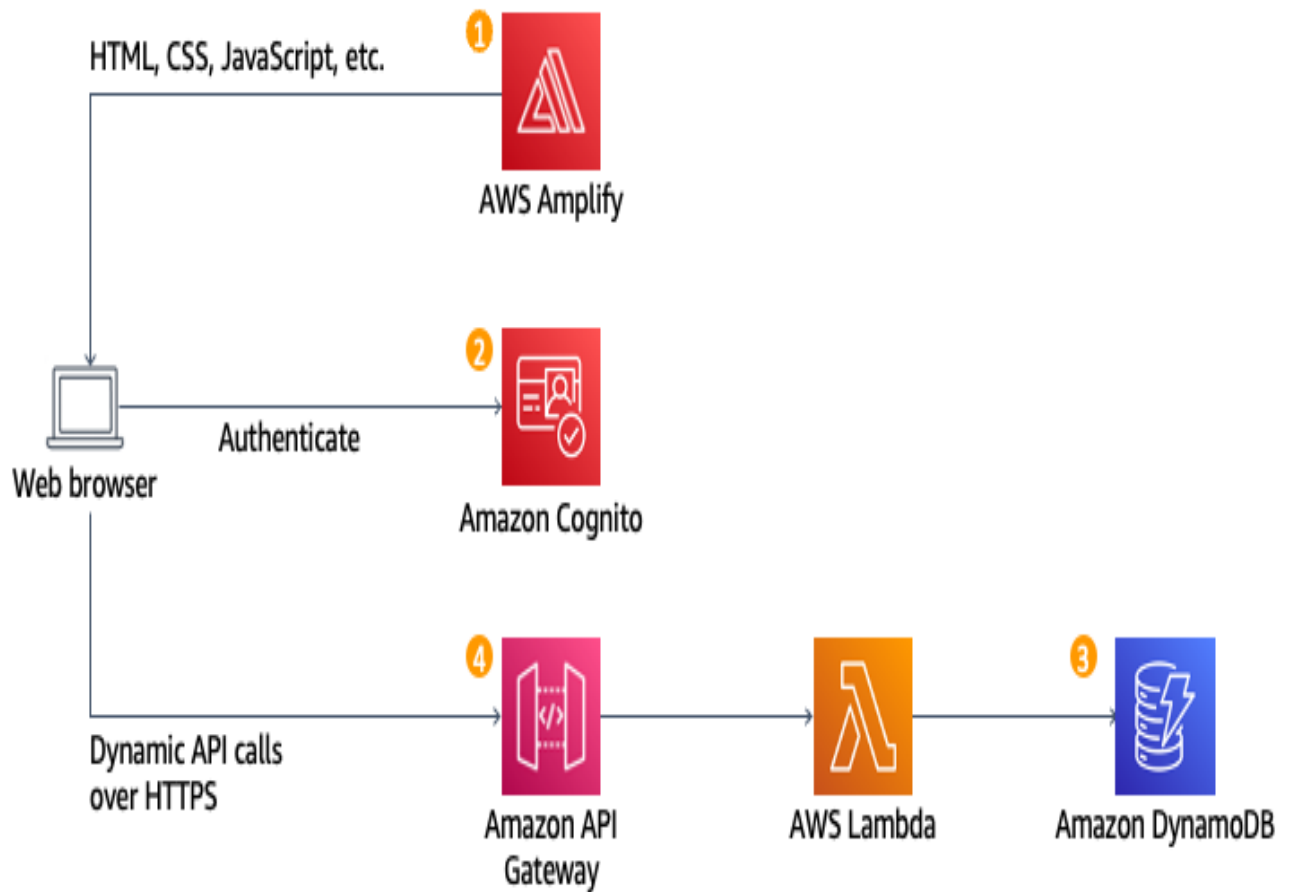
5. Power Supply: Ensure you have a reliable power supply or backup solutions, like a UPS (Uninterruptible Power Supply), to avoid unexpected interruptions during development tasks

3.2 SOFTWARE REQUIREMENTS:

1. ****AWS Account:**** You need an active AWS account to access and utilize AWS cloud services.
2. ****AWS Command Line Interface (CLI):**** Install the AWS CLI on your local development environment to interact with AWS services using command-line commands.
3. ****Integrated Development Environment (IDE) or Code Editor:**** Choose an IDE or code editor that supports your preferred programming language(s). Common choices include Visual Studio Code, IntelliJ IDEA, PyCharm (for Python), and AWS Cloud9 (a cloud-based IDE).
4. ****Git and Version Control:**** Set up Git and choose a version control platform (e.g., GitHub, GitLab, Bitbucket) for tracking changes to your project code and collaborating with team members.
5. ****Programming Languages and Frameworks:**** Depending on your project's requirements, you may need specific programming languages and frameworks. For example, Node.js (for AWS Lambda), Python (for Lambda), and relevant front-end frameworks if applicable

SYSTEM DESIGN

ARCHITECTURE DIAGRAM



SYSTEM DEVELOPMENT STRATEGY

1. Requirements Analysis:

1. Objective: Thoroughly understand and document functional and non-functional requirements for the file-sharing application.
2. Approach: Conduct stakeholder meetings, gather user stories, and analyze project documentation to define clear and comprehensive requirements.

2. Architectural Design:

1. Objective: Define the overall system architecture, emphasizing serverless components and AWS services.
2. Approach: Leverage AWS Lambda for file operations, Amazon S3 for storage, and DynamoDB for metadata management. Ensure scalability, security, and optimal performance.

3. Data Design:

1. Objective: Design data models for storing file metadata in DynamoDB and managing file storage in S3.
2. Approach: Define DynamoDB tables with appropriate attributes for file metadata. Consider S3 bucket organization and versioning for efficient data management.

4. User Interface Design:

1. Objective: Create an intuitive and responsive user interface for seamless file interactions.
2. Approach: Utilize HTML, CSS, and JavaScript for the web interface. Prioritize user experience with clear navigation, file previews, and interactive features.

5. Security Design:

1. Objective: Ensure robust security measures for data protection and user access.
2. Approach: Implement IAM roles with least privilege for Lambda functions. Enforce HTTPS for secure communication. Leverage S3 bucket policies and access control lists (ACLs) for granular control over file access.

6. Integration Design:

1. Objective: Integrate AWS services for smooth operation and data flow.
2. Approach: Establish seamless integration between Lambda functions, S3, and DynamoDB. Utilize AWS SDKs for effective communication and error handling.

7. Scalability Design:

1. Objective: Design the system to handle variable workloads and scale efficiently.
2. Approach: Leverage the serverless architecture of AWS Lambda for automatic scaling. Optimize S3 configurations for scalability and performance.

8. Testing Strategy Design:

1. Objective: Develop a comprehensive testing strategy to ensure the functionality, security, and performance of the system.
2. Approach: Implement unit testing for individual functions, integration testing for component interactions, and end-to-end testing for user scenarios. Include security testing, load testing, and user acceptance testing (UAT).

9. Documentation Plan:

1. Objective: Create detailed documentation for developers, administrators, and end-users.
2. Approach: Document system architecture, data models, API specifications, deployment procedures, and user guides. Maintain clear and up-to-date documentation throughout the development lifecycle.

10. Review and Approval:

1. Objective: Facilitate regular reviews and approvals to ensure alignment with project goals.
2. Approach: Conduct design reviews with stakeholders, development teams, and security experts. Incorporate feedback and obtain necessary approvals before moving to the implementation phase.

IMPLEMENTATION STEPS

1.Setup AWS Environment:

- Create an AWS account if you do not have one.
- Configure AWS CLI with your credentials.

2.Create an S3 Bucket:

- Go to the AWS S3 console.
- Create a new S3 bucket (e.g., filesharing-s3).

3.Create a DynamoDB Table:

- Go to the AWS DynamoDB console.
- Create a new table (e.g., FileMetadata) with the primary key as FileName (String).

4.Configure IAM Roles:

1. Ensure that Lambda functions have the necessary IAM roles and permissions to interact with S3 and DynamoDB.

5.HTML and JavaScript:

1. Use the provided HTML and JavaScript code for the frontend.
2. Update the script to interact with your Lambda functions and AWS services.

6.Testing:

1. Test your application locally before deployment.
2. Ensure that file upload, download, and deletion work as expected.

7.Deployment:

1. Deploy your Lambda functions, HTML, and associated configurations.
2. You can use AWS SAM, Serverless Framework, or the AWS Management Console for deployment.

8.Monitor and Optimize:

1. Set up monitoring using AWS CloudWatch.
2. Optimize your Lambda functions and configurations for performance.

9.User Training:

1. Provide training sessions for end-users on using the file-sharing application.

10.Documentation:

1. Update or create documentation for your application, including deployment guides and user manuals.

11.Continuous Improvement:

1. Collect user feedback and iteratively improve your application.

TESTING STRATEGY

1. Unit Testing:

- Objective: Ensure individual components (JavaScript functions, Lambda functions) work as expected.
- Approach:
 - Test JavaScript functions in script.js to handle file upload, UI updates, and interactions with AWS SDK.
 - Unit test Lambda functions for file upload, download, and deletion, validating logic and error handling.
- Tools:
 - JavaScript Unit Testing Frameworks (e.g., Jest, Mocha) for frontend code.
 - AWS SAM (Serverless Application Model) for testing Lambda functions locally.

2. Integration Testing:

- Objective: Validate the interactions between frontend components and AWS services (S3, DynamoDB, Lambda).
- Approach:
 - Test the integration of HTML, JavaScript, and AWS SDK components.
 - Verify that Lambda functions interact correctly with S3 and DynamoDB.
- Tools:
 - Selenium or Cypress for end-to-end testing of the user interface.
 - AWS SDK provides mocks for simulating AWS service interactions during testing.

3. Security Testing:

- Objective: Identify and address security vulnerabilities in the application.
- Approach:
 - Check for proper implementation of pre-signed URLs for secure file interactions.

- Review AWS IAM roles and permissions for Lambda functions.
- Tools:
 - Manual code reviews for security best practices.
 - AWS IAM Policy Simulator for testing IAM roles.

4. User Acceptance Testing (UAT):

- Objective: Validate that the application meets user expectations and business requirements.
- Approach:
 - Conduct testing with real users or stakeholders.
 - Ensure that users can successfully perform file upload, download, and deletion operations.
- Tools:
 - User feedback and testing sessions.

5. Performance Testing:

- Objective: Evaluate the application's performance under different scenarios.
- Approach:
 - Measure the time taken for file upload and download operations.
 - Simulate concurrent user actions to assess system scalability.
- Tools:
 - Apache JMeter or Gatling for load testing.
 - AWS CloudWatch for monitoring performance metrics.

6. Deployment Testing:

- Objective: Ensure a smooth deployment process without disrupting the live application.
- Approach:
 - Test the deployment of Lambda functions, HTML, and associated configurations.

- Verify that the new version works seamlessly with existing data.
- Tools:
 - AWS Code Deploy for automated deployments.

7. Backup and Recovery Testing:

- Objective: Verify the ability to recover from data loss or system failures.
- Approach:
 - Test backup and restore procedures for S3 and DynamoDB.
 - Simulate scenarios like accidental file deletion and ensure recovery mechanisms work.
- Tools:
 - AWS S3 Versioning for file backup.
 - DynamoDB backups and restores.

8. Documentation Testing:

- Objective: Ensure that all documentation, including user manuals and system architecture, is accurate and up-to-date.
- Approach:
 - Review and verify documentation for completeness.
 - Ensure that deployment guides are clear and follow best practices.
- Tools:
 - Manual review and validation against the deployed application.

9. Cross-Browser Testing:

- Objective: Confirm the application functions correctly across different web browsers.
- Approach:
 - Test the application on popular browsers (Chrome, Firefox, Safari).
 - Verify consistency in UI and functionality.
- Tools:

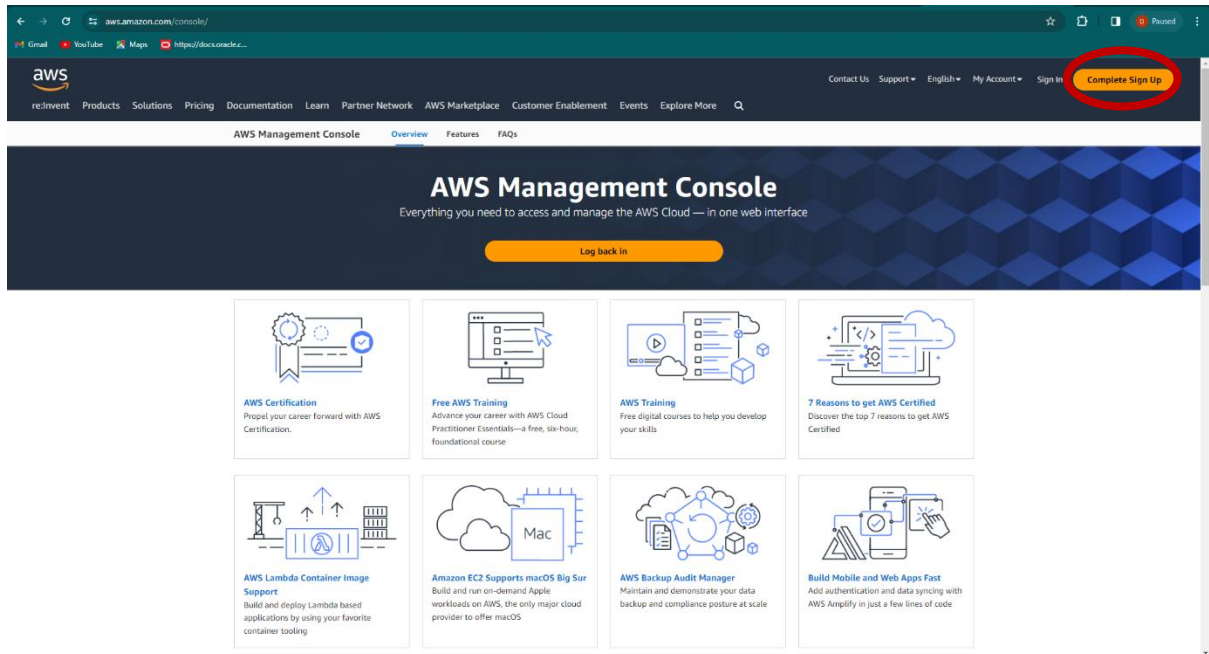
- Cross-browser testing tools like Browser Stack or CrossBrowserTesting.

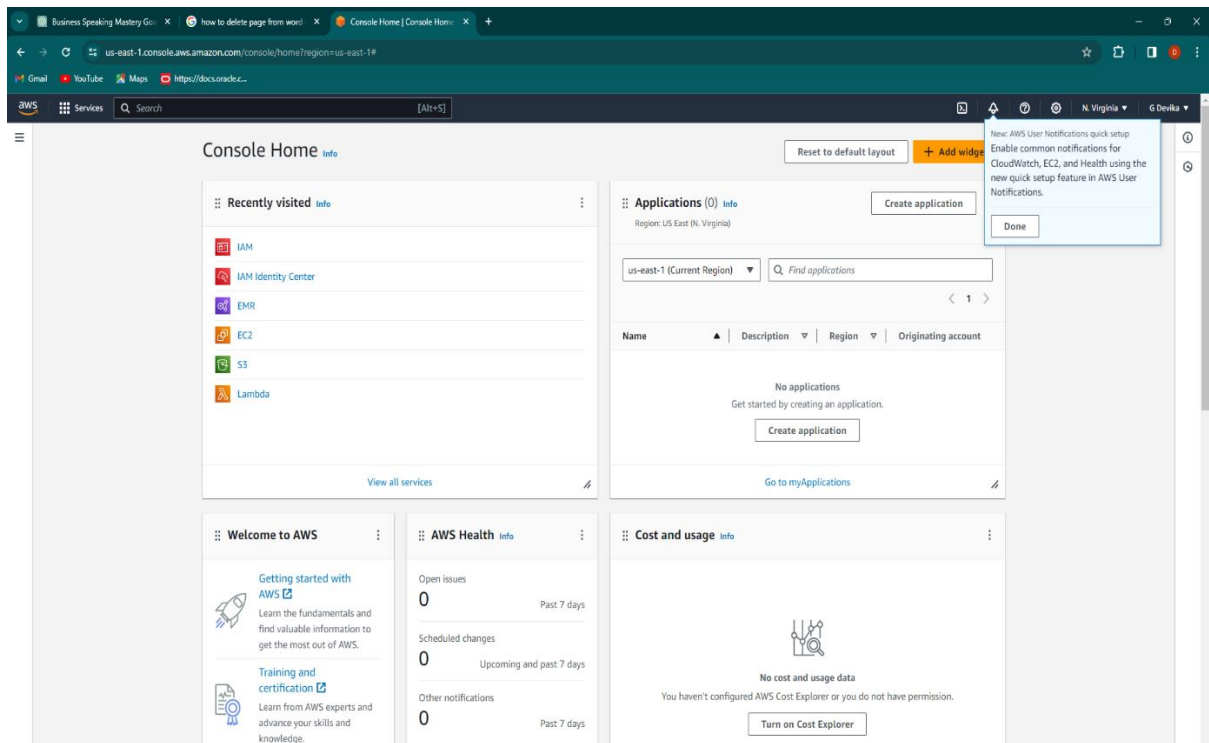
10. Accessibility Testing:

- Objective: Ensure the application is accessible to users with disabilities.
- Approach:
 - Verify that UI elements are labelled correctly.
 - Test keyboard navigation and screen reader compatibility.
- Tools:
 - Accessibility testing tools like Axe or Lighthouse

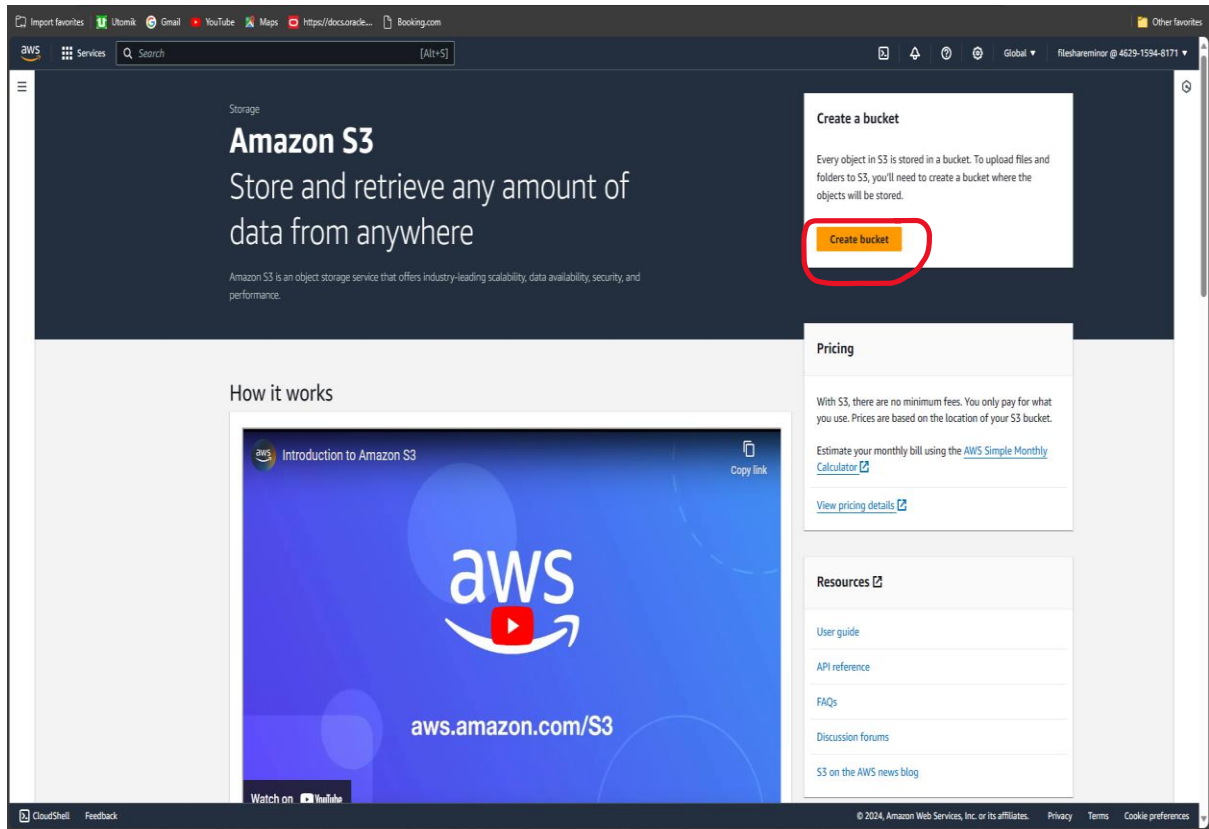
SNAPSHOTS

STEP 1: -Creating an AWS account and logging into AWS Management console.





STEP 2: -Create S3Bucket and upload the file of source code and html file.



Give name for your Bucket

The screenshot shows the AWS Management Console interface for creating a new S3 bucket. The page title is 'Create bucket' with an 'Info' link. Below the title, a brief description states: 'Buckets are containers for data stored in S3. [Learn more](#)'. The 'General configuration' section includes a dropdown for 'AWS Region' set to 'Asia Pacific (Sydney) ap-southeast-2'. The 'Bucket name' field is highlighted with a red circle and contains the text 'fileshareminor'. Below this field, a note states: 'Bucket name must be unique within the global namespace and follow the bucket naming rules. [See rules for bucket naming](#)'. There is also a section for 'Copy settings from existing bucket - optional' with a 'Choose bucket' button and a format example 'Format: s3://bucket/prefix'. The 'Object Ownership' section shows two options: 'ACLs disabled (recommended)' (selected) and 'ACLs enabled'. The 'Block Public Access settings for this bucket' section is partially visible at the bottom.

Create bucket [Info](#)

Buckets are containers for data stored in S3. [Learn more](#)

General configuration

AWS Region
Asia Pacific (Sydney) ap-southeast-2

Bucket name [Info](#)
fileshareminor

Bucket name must be unique within the global namespace and follow the bucket naming rules. [See rules for bucket naming](#)

Copy settings from existing bucket - optional
Only the bucket settings in the following configuration are copied.
[Choose bucket](#)
Format: s3://bucket/prefix

Object Ownership [Info](#)
Control ownership of objects written to this bucket from other AWS accounts and the use of access control lists (ACLs). Object ownership determines who can specify access to objects.

☒ **ACLs disabled (recommended)**
All objects in this bucket are owned by this account. Access to this bucket and its objects is specified using only policies.

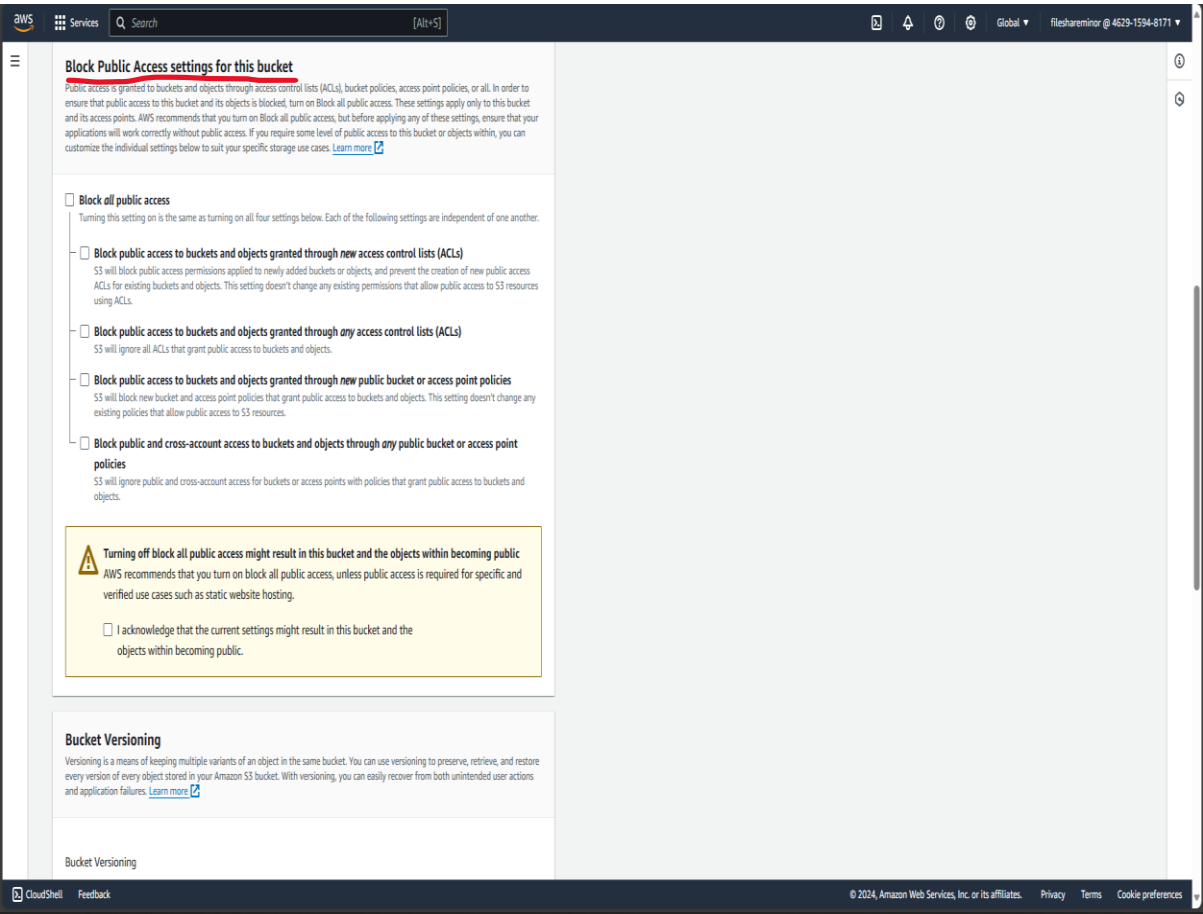
☐ **ACLs enabled**
Objects in this bucket can be owned by other AWS accounts. Access to this bucket and its objects can be specified using ACLs.

Object Ownership
Bucket owner enforced

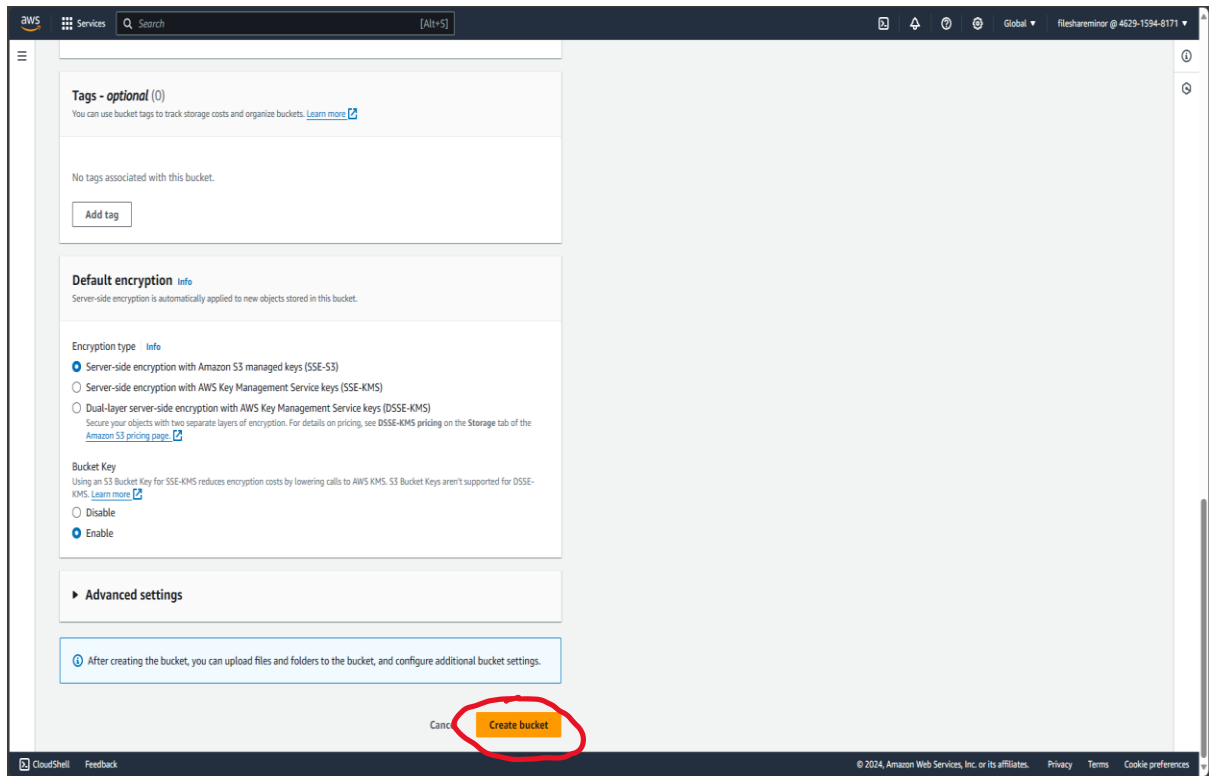
Block Public Access settings for this bucket
Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. These sections apply only to this bucket.

© 2024, Amazon Web Services, Inc. or its affiliates. [Privacy](#) [Terms](#) [Cookie preferences](#)

Give public access for your Bucket



Click on Create button



Uploaded the file successfully:

Upload succeeded

View details below.

Upload: status

Close

The information below will no longer be available after you navigate away from this page.

Summary

Destination

s3://fileshareminor

Succeeded

2 files, 4.6 KB (100.00%)

Failed

0 files, 0 B (0%)

Files and folders

Configuration

Files and folders (2 Total, 4.6 KB)

Find by name

< 1 >

| Name | Folder | Type | Size | Status | Error |
|-----------|-----------------|-----------------|--------|-----------|-------|
| aws.html | FileSharingApp/ | text/html | 1.3 KB | Succeeded | - |
| script.js | FileSharingApp/ | text/javascript | 3.2 KB | Succeeded | - |

CloudShell

Feedback

© 2023, Amazon Web Services, Inc. or its affiliates. [Privacy](#) [Terms](#) [Cookie preferences](#)

Create an IAM user and login to IAM user

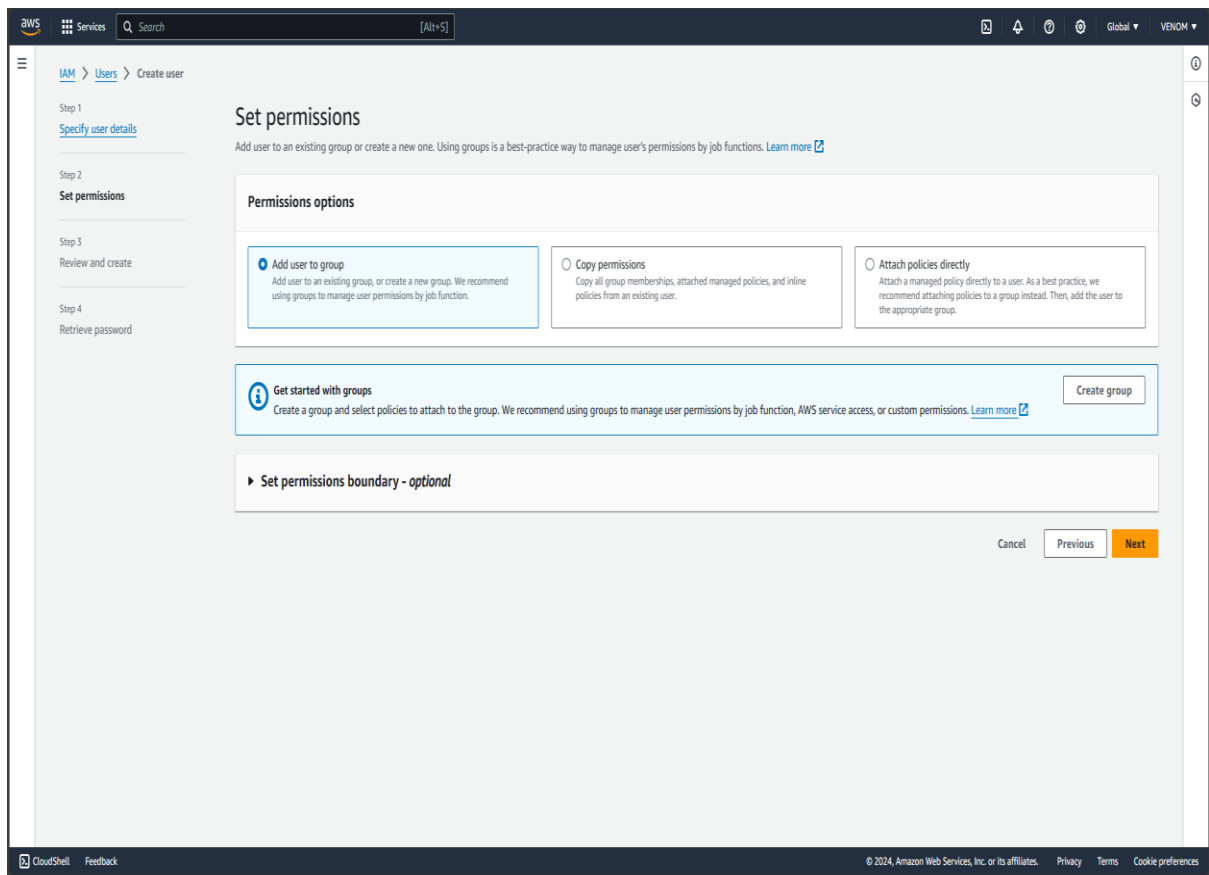
The screenshot displays the AWS IAM Dashboard for the user 'fileshareminor' in the 'Global' region. The dashboard is divided into several sections:

- Security recommendations:** Contains two warnings: 'Add MFA for root user' and 'Add MFA for yourself', both with an 'Add MFA' button. A green checkmark indicates that the user 'fileshareminor' does not have any active access keys that have been unused for more than a year.
- IAM resources:** A table showing the count of resources in the AWS Account:

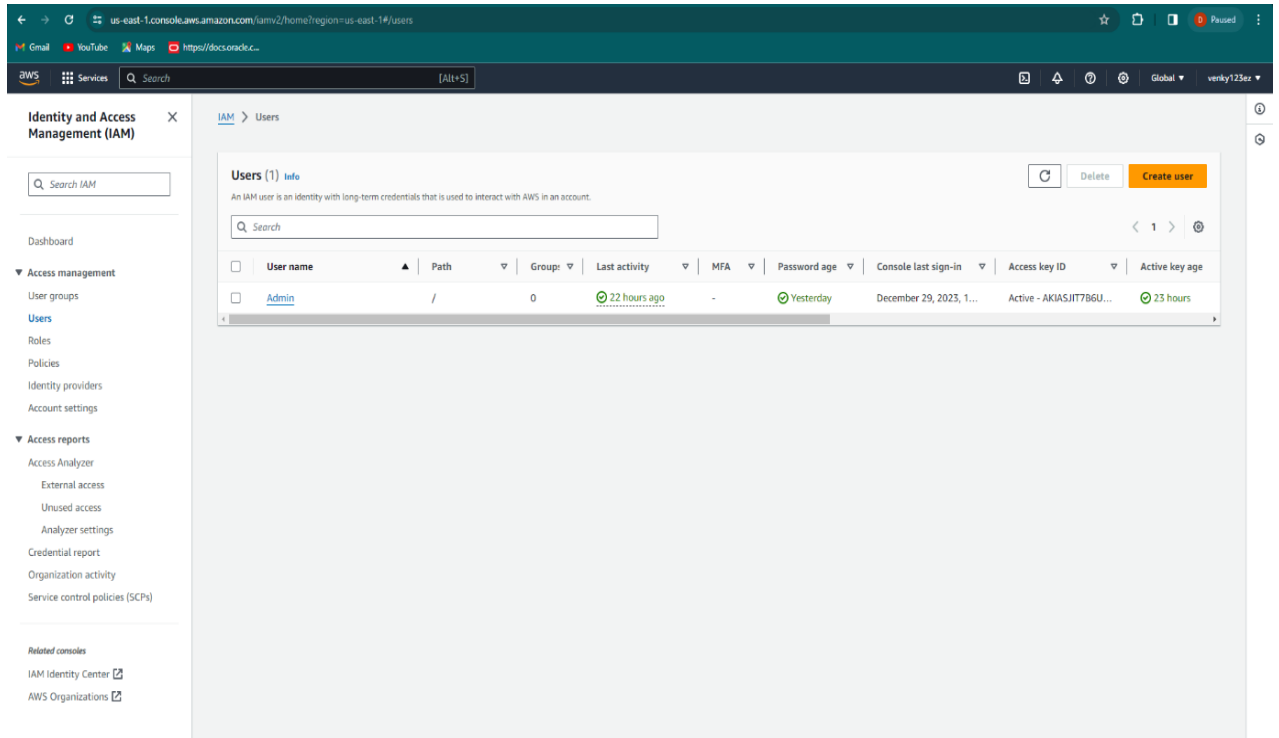
| User groups | Users | Roles | Policies | Identity providers |
|-------------|-------|-------|----------|--------------------|
| 0 | 1 | 2 | 0 | 0 |
- What's new:** A section titled 'Updates for features in IAM' with a 'View all' link. It lists three updates:
 - IAM Access Analyzer now simplifies inspecting unused access to guide you toward least privilege. 1 month ago
 - IAM Access Analyzer introduces custom policy checks powered by automated reasoning. 1 month ago
 - Announcing AWS IAM Identity Center APIs for visibility into workforce access to AWS. 1 month ago
 - New organization-wide IAM condition keys to restrict AWS service-to-service requests. 1 month ago
- AWS Account:** Displays the Account ID (462915948171), Account Alias (Create), and the Sign-in URL for IAM users in this account (https://462915948171.signin.aws.amazon.com/console).
- Quick Links:** A link to 'My security credentials' with the description: 'Manage your access keys, multi-factor authentication (MFA) and other credentials.'
- Tools:** A link to 'Policy simulator' with the description: 'The simulator evaluates the policies that you choose and determines the effective permissions for each of the actions that you specify.'
- Additional information:** A link to 'Security best practices in IAM'.

The left sidebar shows the 'Identity and Access Management (IAM)' section with a search bar and a list of navigation links: Dashboard, Access management (User groups, Users, Roles, Policies, Identity providers, Account settings), Access reports (Access Analyzer, External access, Unused access, Analyzer settings, Credential report, Organization activity, Service control policies (SCPs)), and Related consoles (IAM Identity Center, AWS Organizations).





Successfully Created an IAM user



The screenshot displays the AWS IAM console interface. The left-hand navigation pane is titled "Identity and Access Management (IAM)" and includes sections for "Access management" (User groups, Users, Roles, Policies, Identity providers, Account settings) and "Access reports" (Access Analyzer, External access, Unused access, Analyzer settings, Credential report, Organization activity, Service control policies (SCPs)). Below these are "Related consoles" for IAM Identity Center and AWS Organizations.

The main content area is titled "Users (1) Info" and contains a search bar and a table of users. The table has columns for selection, User name, Path, Group, Last activity, MFA, Password age, Console last sign-in, Access key ID, and Active key age. One user, "Admin", is listed with a path of "/", group of "0", last activity of "22 hours ago", MFA status of "-", password age of "Yesterday", console last sign-in of "December 29, 2023, 1...", access key ID of "Active - AKIASJIT786U...", and active key age of "23 hours".

| <input type="checkbox"/> | User name | Path | Group | Last activity | MFA | Password age | Console last sign-in | Access key ID | Active key age |
|--------------------------|-----------------------|------|-------|---------------|-----|--------------|-------------------------|--------------------------|----------------|
| <input type="checkbox"/> | Admin | / | 0 | 22 hours ago | - | Yesterday | December 29, 2023, 1... | Active - AKIASJIT786U... | 23 hours |

Add the Policies permissions for the IAM user to get full access.

The screenshot shows the AWS IAM console interface. At the top, a green banner indicates "10 policies added to fileshareminor". Below this, the user's details are shown: ARN (arn:aws:iam::157345058728:user/fileshareminor), Console access (Enabled without MFA), and Access key 1 (Create access key). The user was created on December 30, 2023, and last signed in today. The "Permissions" tab is selected, showing a list of 10 policies attached to the user. The policies are all AWS managed and attached directly. The left sidebar contains navigation links for Identity and Access Management (IAM), Access management, Access reports, and Related consoles.

| Policy name | Type | Attached via |
|--|----------------------------|--------------|
| AdministratorAccess | AWS managed - job function | Directly |
| AmazonDynamoDBFullAccess | AWS managed | Directly |
| AmazonDynamoDBFullAccesswithDataPipeline | AWS managed | Directly |
| AmazonDynamoDBReadOnlyAccess | AWS managed | Directly |
| AmazonS3FullAccess | AWS managed | Directly |
| AmazonS3ObjectLambdaExecutionRolePolicy | AWS managed | Directly |
| AmazonS3OutpostsFullAccess | AWS managed | Directly |
| AmazonS3OutpostsReadOnlyAccess | AWS managed | Directly |
| AWSLambda_FullAccess | AWS managed | Directly |
| AWSLambda_ReadOnlyAccess | AWS managed | Directly |

Generate the access key for your IAM to include it in the Code

The screenshot displays the AWS IAM console interface. On the left, the 'Identity and Access Management (IAM)' sidebar is visible, with a search bar and a list of navigation items including Dashboard, Access management, User groups, Users, Roles, Policies, Identity providers, Account settings, Access reports, Access Analyzer, External access, Unused access, Analyzer settings, Credential report, Organization activity, Service control policies (SCPs), and Related consoles. The main content area is titled 'fileshareminior' and includes a 'Delete' button. Below the title, a 'Summary' section shows the user's ARN, creation date, console access status (Enabled without MFA), and last console sign-in. The 'Security credentials' tab is highlighted with a red circle. This tab contains sections for 'Console sign-in' (with a link to the console and a password update status) and 'Multi-factor authentication (MFA)' (showing 0 devices and an 'Assign MFA device' button). The bottom of the console shows the footer with 'CloudShell', 'Feedback', and copyright information.

Summary

| | | |
|---|---------------------------------------|---|
| ARN arn:aws:iam::462915948171:user/fileshareminior | Console access Enabled without MFA | Access key 1 Create access key |
| Created January 01, 2024, 13:23 (UTC+05:30) | Last console sign-in Never | |

Security credentials

Console sign-in

| | |
|---|---|
| Console sign-in link https://462915948171.signin.aws.amazon.com/console | Console password Updated Now (2024-01-01 13:23 GMT+5:30) |
| | Last console sign-in Never |

Multi-factor authentication (MFA) (0)

Use MFA to increase the security of your AWS environment. Signing in with MFA requires an authentication code from an MFA device. Each user can have a maximum of 8 MFA devices assigned. [Learn more](#)

| Device type | Identifier | Certifications | Created on |
|--|------------|----------------|------------|
| No MFA devices. Assign an MFA device to improve the security of your AWS environment | | | |

[Assign MFA device](#)

Click on Create access key

Access keys (0)

Use access keys to send programmatic calls to AWS from the AWS CLI, AWS Tools for PowerShell, AWS SDKs, or direct AWS API calls. You can have a maximum of two access keys (active or inactive) at a time. [Learn more](#)

No access keys. As a best practice, avoid using long-term credentials like access keys. Instead, use tools which provide short term credentials. [Learn more](#)

Create access key

SSH public keys for AWS CodeCommit (0)

Use SSH public keys to authenticate access to AWS CodeCommit repositories. You can have a maximum of five SSH public keys (active or inactive) at a time. [Learn more](#)

| SSH Key ID | Uploaded | Status |
|--------------------|----------|--------|
| No SSH public keys | | |

Upload SSH public key

HTTPS Git credentials for AWS CodeCommit (0)

Generate a user name and password you can use to authenticate HTTPS connections to AWS CodeCommit repositories. You can have a maximum of 2 sets of credentials (active or inactive) at a time. [Learn more](#)

| User name | Created | Status |
|----------------|---------|--------|
| No credentials | | |

Generate credentials

Credentials for Amazon Keyspaces (for Apache Cassandra) (0)

Generate a user name and password you can use to authenticate to Amazon Keyspaces. You can have a maximum of two sets of credentials (active or inactive) at a time. [Learn more](#)

| User name | Created | Status |
|-----------|---------|--------|
|-----------|---------|--------|

© 2024, Amazon Web Services, Inc. or its affiliates. [Privacy](#) [Terms](#) [Cookie preferences](#)

Give name for access key

The screenshot shows the AWS IAM console interface for creating an access key. The breadcrumb trail is IAM > Users > fileshareminior > Create access key. The left sidebar shows the progress: Step 1 (Access key best practices & alternatives), Step 2 - optional (Set description tag), and Step 3 (Retrieve access keys). The main content area is titled 'Set description tag - optional' with an info icon. It explains that the description will be attached to the user as a tag. A text input field labeled 'Description tag value' contains the text 'filesharing'. Below the field, a note states: 'Maximum 256 characters. Allowed characters are letters, numbers, spaces representable in UTF-8, and: _ - / = + - @'. At the bottom right, there are three buttons: 'Cancel', 'Previous', and 'Create access key'.

Step 1
[Access key best practices & alternatives](#)

Step 2 - optional
Set description tag

Step 3
Retrieve access keys

Set description tag - optional [Info](#)

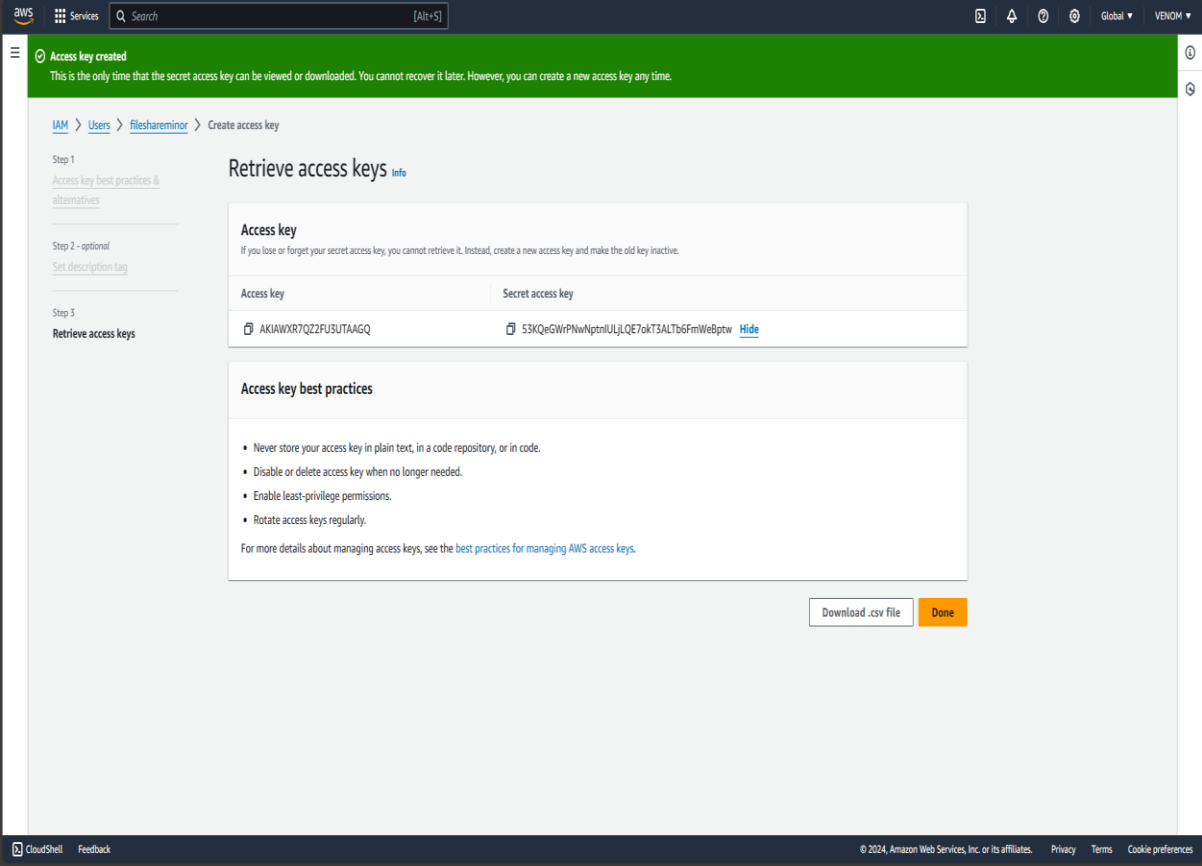
The description for this access key will be attached to this user as a tag and shown alongside the access key.

Description tag value
Describe the purpose of this access key and where it will be used. A good description will help you rotate this access key confidently later.

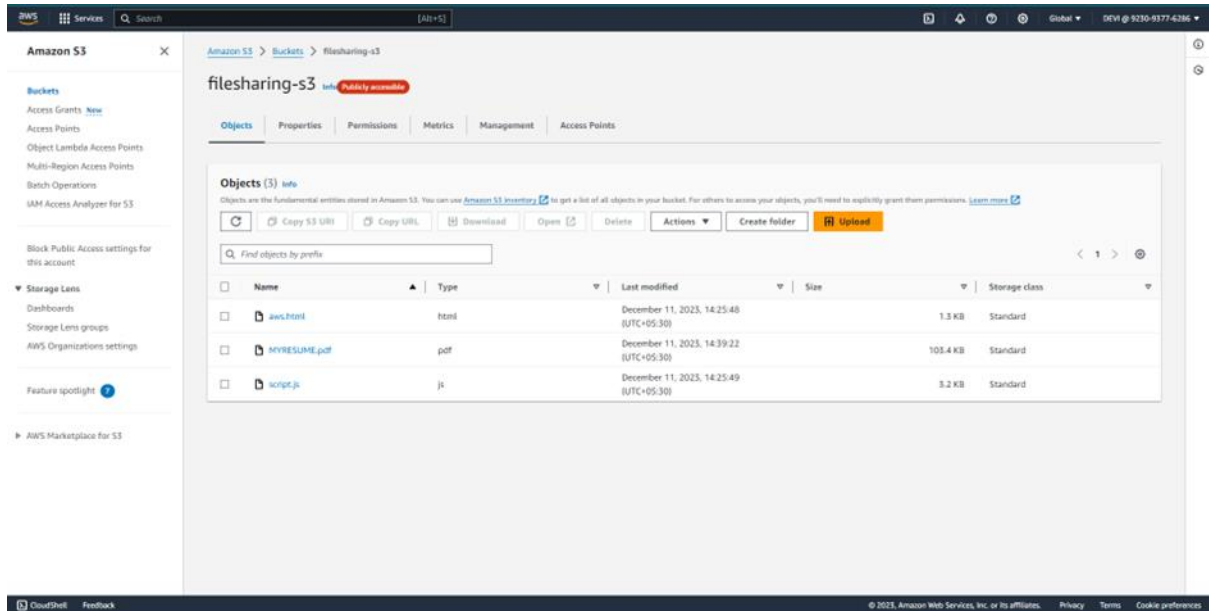
filesharing

Maximum 256 characters. Allowed characters are letters, numbers, spaces representable in UTF-8, and: _ - / = + - @

Cancel Previous **Create access key**



Now Upload the Script.js and html file to Your S3 Bucket.



The following File contains the Code in it.

Pseudo-code included in the aws.html file is given below:

Segment-1: AWS SDK Configuration

```
// AWS SDK Configuration
var region = "us-east-1";
var accessKeyId = "your-access-key-id";
var secretAccessKey = "your-secret-access-key";

AWS.config.update({
    region: region,
    credentials: new AWS.Credentials(accessKeyId, secretAccessKey)
});
|
```

Segment-2: S3 Service Instance Creation.

```
// Create an instance of the S3 service  
var s3 = new AWS.S3();
```

Segment-3: File List Refresh Function

```
// Function to refresh the file list in the table
function refreshFileList(bucketname) {
  // Implementation details for fetching file list from S3
  s3.listObjectsV2({ Bucket: bucketname }, (err, data) => {
    if (err) {
      console.log("Error fetching file list", err);
    } else {
      // Update the fileTable with the retrieved file data
      updateFileTable(data.Contents);
    }
  });
}
```

Segment-4: File Upload Function

```
// Function to upload files to S3
function uploadFiles(bucketname) {
  // Implementation details for handling file upload to S3
  let files = document.getElementById('fileInput').files;

  for (let i = 0; i < files.length; i++) {
    let file = files[i];
    let params = {
      Bucket: bucketname,
      Key: file.name,
      Body: file
    };

    // Use the S3 upload method to upload the file
    s3.upload(params, (err, data) => {
      if (err) {
        console.log("Error uploading file", err);
      } else {
        console.log("File uploaded successfully");
        // Refresh the file list after successful upload
        refreshFileList(bucketname);
      }
    });
  }
}
```


Segment-5: File Delete Function:

```
// Function to delete a file from S3
function deleteFile(bucketname, objectKey) {
  // Implementation details for deleting a file from S3
  let params = {
    Bucket: bucketname,
    Key: objectKey
  };

  // Use the S3 deleteObject method to delete the file
  s3.deleteObject(params, (err, data) => {
    if (err) {
      console.log("Error deleting file", err);
    } else {
      console.log("File deleted successfully");
      // Refresh the file list after successful deletion
      refreshFileList(bucketname);
    }
  });
}
```

Segment-6: Initial file List Refresh

```
// Initial file list refresh for the default bucket  
refreshFileList("filesharing-s3");  
|
```

Source Code

```
var region = "us-east-1";
var accessKeyId = "AKIA5N3FNG6PEVCXD2MR";
var secretAccessKey = "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";

AWS.config.update({
  region: region,
  credentials: new AWS.Credentials(accessKeyId, secretAccessKey)
});

var s3 = new AWS.S3();

function refreshFileList(bucketname) {
  var tableBody = document.querySelector("#fileTable tbody");
  tableBody.innerHTML = "";

  s3.listObjectsV2({ Bucket: bucketname }, (err, data) => {
    if (err) {
      console.log("Error fetching file list", err);
    } else {
      data.Contents.forEach((object) => {
        var fileRow = document.createElement('tr');

        var fileNameCell = document.createElement('td');
        fileNameCell.textContent = object.Key;
        fileRow.appendChild(fileNameCell);

        var fileSizeCell = document.createElement("td");
        fileSizeCell.textContent = object.Size;
        fileRow.appendChild(fileSizeCell);

        var downloadCell = document.createElement('td');
        var downloadLink = document.createElement('a');

        // Generate a pre-signed URL with the 'response-content-disposition' parameter
        var params = {
          Bucket: bucketname,
          Key: object.Key,
          ResponseContentDisposition: 'attachment; filename="' + object.Key + '"'
        };

        downloadLink.href = s3.getSignedUrl("getObject", params);
        downloadLink.textContent = "Download";

        downloadCell.appendChild(downloadLink);
        fileRow.appendChild(downloadCell);

        var deleteCell = document.createElement('td');
        var deleteButton = document.createElement('button');
        deleteButton.textContent = "Delete";
        deleteButton.addEventListener('click', () => {
          deleteFile(bucketname, object.Key);
        });
      });
    }
  });
}
```

```

        deleteCell.appendChild(deleteButton);
        fileRow.appendChild(deleteCell);

        tableBody.appendChild(fileRow);
    });
}
});
}

function uploadFiles(bucketname) {
    let files = document.getElementById('fileInput').files;

    for (let i = 0; i < files.length; i++) {
        let file = files[i];
        let params = {
            Bucket: bucketname,
            Key: file.name,
            Body: file
        };

        s3.upload(params, (err, data) => {
            if (err) {
                console.error("Error uploading file", err);
            } else {
                console.log("File uploaded successfully");
                refreshFileList(bucketname);
            }
        });
    }
}

function deleteFile(bucketname, objectKey) {
    var params = {
        Bucket: bucketname,
        Key: objectKey
    };

    s3.deleteObject(params, (err, data) => {
        if (err) {
            console.log("Error deleting file", err);
        } else {
            console.log("File deleted successfully");
            refreshFileList(bucketname);
        }
    });
}

// Initial file list refresh
refreshFileList("filesharing-s3");

```

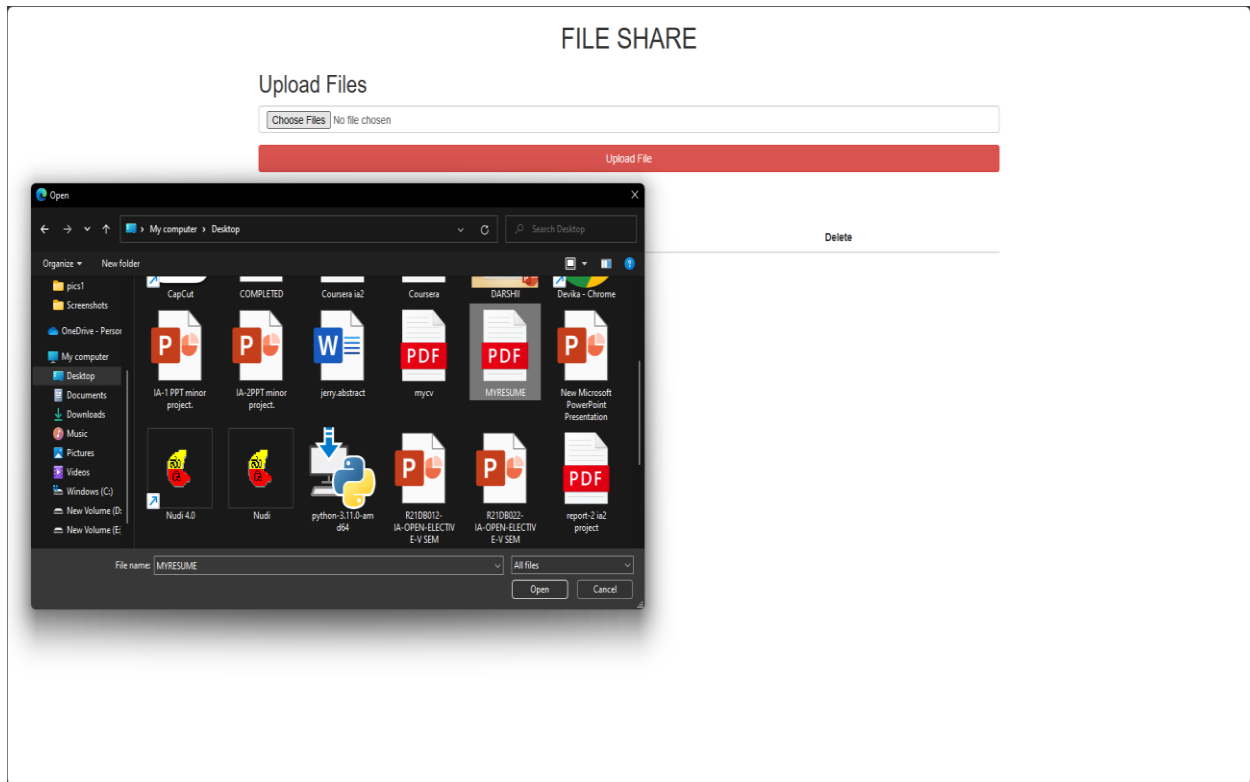
Direct to the application website now

The screenshot shows the AWS Management Console interface for an Amazon S3 bucket named 'fileshareminor1'. The breadcrumb navigation path is 'Amazon S3 > Buckets > fileshareminor1 > FileSharingApp/ > aws.html'. The object 'aws.html' is selected, and its details are displayed under the 'Properties' tab. The 'Object overview' section shows the following information:

| Property | Value |
|----------------------------|---|
| Owner | 2107446 |
| AWS Region | Asia Pacific (Sydney) ap-southeast-2 |
| Last modified | January 1, 2024, 14:12:29 (UTC+05:30) |
| Size | 1.3 KB |
| Type | html |
| Key | FileSharingApp/aws.html |
| S3 URI | s3://fileshareminor1/FileSharingApp/aws.html |
| Amazon Resource Name (ARN) | arn:aws:s3::fileshareminor1/FileSharingApp/aws.html |
| Entity tag (ETag) | 96dc2b885c383b8554aab91dc03eb0f9 |
| Object URL | https://fileshareminor1.s3.ap-southeast-2.amazonaws.com/FileSharingApp/aws.html |

At the top right of the object details section, there are three buttons: 'Copy S3 URI', 'Download', and 'Open'. The 'Open' button is highlighted with a red circle. To the right of the 'Open' button is a dropdown menu labeled 'Object actions'. Below the 'Object overview' section, there is an 'Object management overview' section with a note about bucket properties and object management configurations. The footer of the console shows the year 2024 and links to Privacy, Terms, and Cookie preferences.

Now You can Upload the file you need to share



FILE SHARE

Upload Files

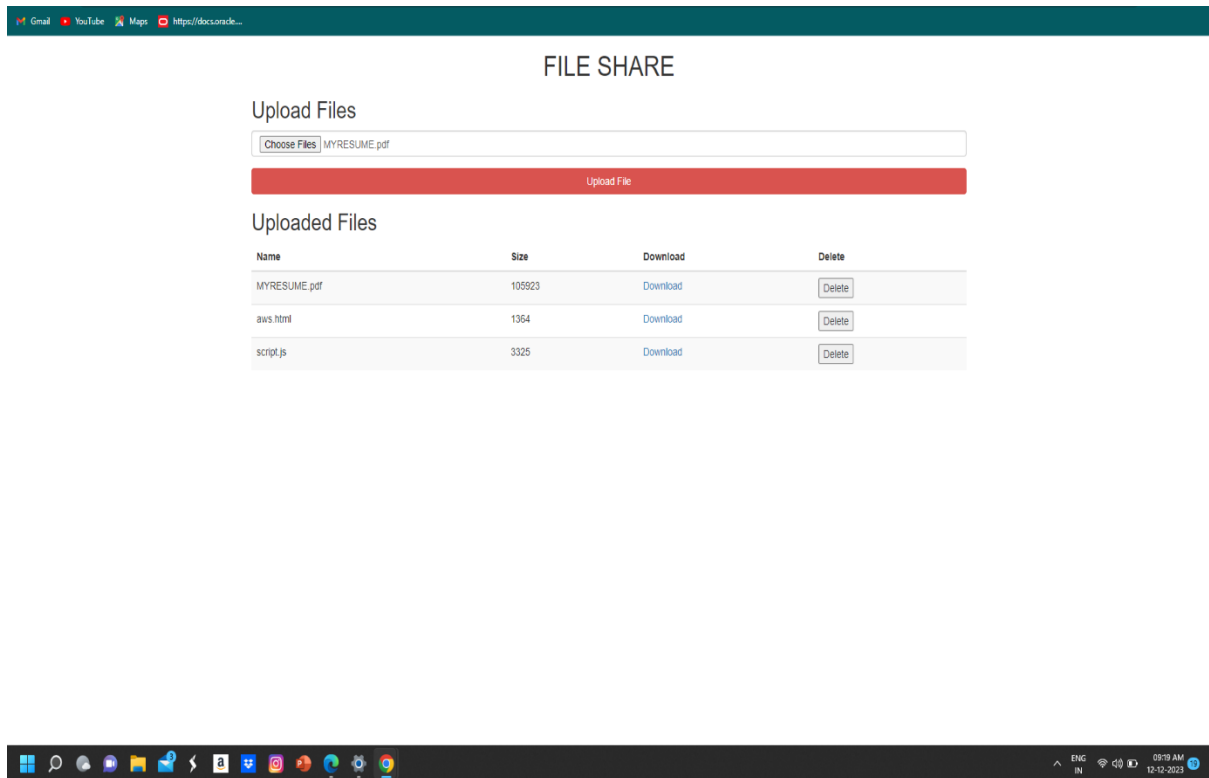
Choose Files MYRESUME.pdf

Upload File

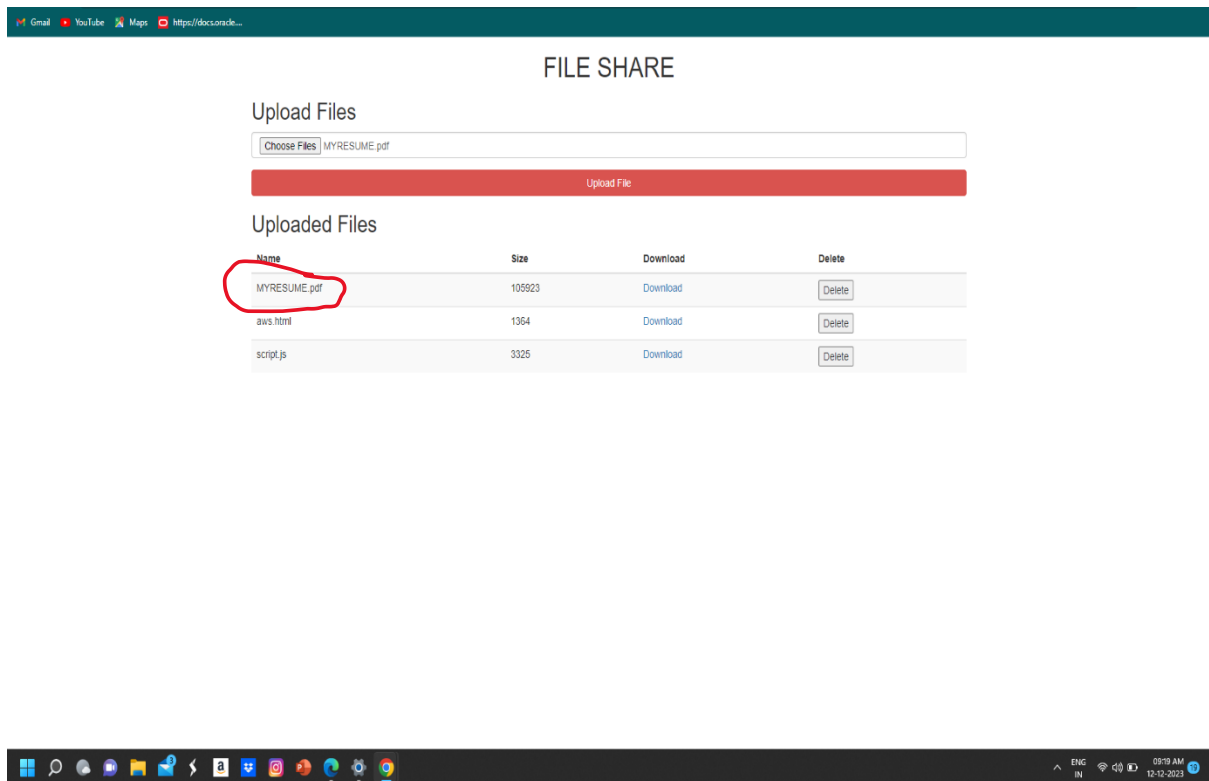
Uploaded Files

| Name | Size | Download | Delete |
|--------------|--------|--------------------------|-------------------------|
| MYRESUME.pdf | 105923 | Download | <button>Delete</button> |
| aws.html | 1364 | Download | <button>Delete</button> |
| script.js | 3325 | Download | <button>Delete</button> |

Final Image of the output:



I have Uploaded the 'MYRESUME' file to share



The uploaded file is shown in your AWS S3 bucket too.

The screenshot shows the AWS S3 console interface. The left sidebar contains navigation links for Amazon S3, Buckets, Access Grants, Access Points, Object Lambda Access Points, Multi-Region Access Points, Batch Operations, IAM Access Analyzer for S3, Storage Lens, Dashboards, Storage Lens groups, AWS Organizations settings, Feature spotlight, and AWS Marketplace for S3. The main content area displays the 'filessharing-s3' bucket, which is publicly accessible. The 'Objects' tab is selected, showing a list of three objects: 'aws.html' (1.3 KB, Standard), 'MYRESUME.pdf' (103.4 KB, Standard), and 'script.js' (3.2 KB, Standard). The 'MYRESUME.pdf' file is circled in red. The console also includes a search bar, a list of actions (Copy S3 URI, Copy URL, Download, Open, Delete), and a 'Create folder' button.

| Name | Type | Last modified | Size | Storage class |
|--------------|------|---|----------|---------------|
| aws.html | html | December 11, 2023, 14:25:48 (UTC+05:30) | 1.3 KB | Standard |
| MYRESUME.pdf | pdf | December 11, 2023, 14:39:22 (UTC+05:30) | 103.4 KB | Standard |
| script.js | js | December 11, 2023, 14:25:49 (UTC+05:30) | 3.2 KB | Standard |

Conclusion:

1. The development of the serverless file sharing application using Amazon DynamoDB, Amazon S3 bucket, and AWS Lambda has been successfully accomplished. The project leverages the power of AWS cloud services to provide a seamless and scalable file sharing experience. Key functionalities include file upload, download, and deletion, all orchestrated through AWS Lambda functions and DynamoDB for efficient metadata management.
2. The application is designed to be serverless, ensuring cost-effectiveness, scalability, and high availability. Users can easily upload files through a user-friendly interface, and the system dynamically updates the file list using S3 and DynamoDB. The solution adheres to AWS best practices, ensuring security and optimal performance.
3. Overall, the project achieves its objectives by delivering a robust and user-friendly file sharing platform powered by AWS cloud services. Future enhancements and optimizations can be explored to further enhance the application's capabilities and meet evolving user requirements.

BIBLIOGRAPHY

1. Amazon S3 Documentation: <https://docs.aws.amazon.com/s3/>
2. Amazon IAM user guide:
https://docs.aws.amazon.com/IAM/latest/UserGuide/id_users.html
3. User guide to build System design and java script:-
<https://aws.amazon.com/blogs/compute/building-scalable-serverless-applications-with-amazon-s3-and-aws-lambda/>