# Cloud-Based Smart File Storage & Sharing System

Submitted by:  Jeevan Fenittus S

Mob num : 9384804362

Github Repo : https://github.com/Jeevanfenittus/smart-drive.git

Linkedin: www.linkedin.com/in/jeevan-fenittus

Organization: Elevate Labs

project report submitted as part of the Cloud Computing Internship Program .

## Abstract

The **Cloud-Based Smart File Storage and Sharing System (Smart Drive)** is a web-based application designed to provide users with a secure, scalable, and reliable platform for storing and sharing files using **cloud computing technologies**.

This project replicates the core functionality of popular cloud storage services like Google Drive or Dropbox by enabling users to **register, log in, upload, and share files** through a simple and responsive web interface.

The system is built using **Flask (Python)** for the backend, **AWS S3** for file storage, and **AWS DynamoDB** for metadata management. **JWT (JSON Web Token)** authentication ensures secure access to user accounts and resources.

All application components are hosted on an **AWS EC2 instance**, providing real-time availability and scalability. Users can upload files to the cloud, retrieve them through dynamically generated URLs, and share them easily with others.

This project demonstrates a practical implementation of **cloud architecture, serverless data management, and security best practices**, showcasing how modern web applications can leverage cloud infrastructure for performance and reliability.

## Objectives

The main objective of the Cloud-Based Smart File Storage and Sharing System (Smart Drive) is to design and deploy a secure and efficient cloud application that enables users to store, manage, and share files seamlessly using cloud technologies.

Specific Objectives

1. To develop a web-based file storage platform that allows users to register, log in, upload, and access files from anywhere.
2. To integrate Amazon S3 for scalable and durable cloud storage of user files.

3. To use AWS DynamoDB for storing metadata such as file name, upload date, and user ownership details.
4. To implement secure authentication using JWT (JSON Web Tokens) for user access control.
5. To host and deploy the backend on AWS EC2, ensuring high availability and reliability.
6. To create a user-friendly web interface using HTML, CSS, and Bootstrap for easy interaction with the system.
7. To provide file-sharing functionality through dynamically generated public URLs.
8. To ensure data security and privacy through encryption and restricted access control.
9. To understand and demonstrate the use of cloud services for real-world web application deployment.

## System Design & Architecture

- The Smart Drive system is a cloud-based web application designed to provide users with secure file storage, access, and sharing capabilities. The architecture is built using AWS Cloud Services to ensure scalability, availability, and reliability.
- The system follows a three-tier architecture:
- Frontend (Presentation Layer) – User Interface (HTML, CSS, Bootstrap)

- Backend (Application Layer) – Flask (Python) API handling authentication, file upload, and metadata management
- Database & Storage (Data Layer) – AWS S3 for file storage and DynamoDB for metadata

2. Components Description

a. Frontend

- Built using HTML5, CSS3, and Bootstrap for a responsive and modern UI.
- Handles user interactions such as registration, login, and file upload.
- Communicates with the backend via RESTful API calls (Fetch API).

b. Backend (Flask Application)

- Developed using Python Flask framework.
- Exposes APIs for:
  - /api/auth/register → User registration
  - /api/auth/login → User authentication (JWT-based)
  - /api/upload → File upload to S3 and metadata storage
  - /files → List or download files
- Generates JWT tokens for secure session management.

c. AWS S3 (Simple Storage Service)

- Used as the main storage for uploaded files.
- Provides high durability and scalability.
- Public access is controlled through pre-signed URLs to ensure security.

d. AWS DynamoDB

- Stores metadata such as:
  - File name
  - File URL
  - Uploaded user
  - Upload timestamp
- Enables fast retrieval and indexing of files per user.

e. AWS EC2

- Hosts the Flask backend application.
- Provides public IP access for remote users.
- Configured with security groups for HTTP/HTTPS traffic and restricted SSH access.

---

4. Data Flow

1. User accesses the login/register page hosted via EC2 public IP.
2. Upon successful authentication, the user uploads a file.

3. The backend validates the JWT token and uploads the file to AWS S3.
4. File metadata is saved in DynamoDB.
5. The backend returns a public access link for file sharing.
6. Users can view or share the file via the generated link.

## Technologies Used

🌐 Frontend

- HTML5, CSS3, Bootstrap 5 → For creating a responsive and user-friendly interface.
- JavaScript (Fetch API) → For communicating with backend APIs asynchronously.

⚙️ Backend

- Python Flask → Lightweight web framework to handle API requests, authentication, and routing.
- JWT (JSON Web Token) → For secure user authentication and session management.

☁️ Cloud Platform

- Amazon Web Services (AWS)
    - EC2 → For hosting and running the Flask application.

    - S3 (Simple Storage Service) → To store uploaded files securely and provide access URLs.
    - IAM → For managing permissions and secure access between EC2 and S3.

🗄️ Database

- Amazon DynamoDB → NoSQL database used to store user and file metadata (filename, owner, date, S3 URL).

🔒 Security

- HTTPS (optional for deployment) → To secure communication between users and the server.
- JWT Tokens → To ensure only authenticated users can upload or access files.

🚀 Deployment & Version Control

- Git & GitHub → For source code management and collaboration.
- AWS EC2 / Render → For live deployment of the backend server.
- Vercel / Netlify (optional) → For hosting static frontend pages.

## Implementation Steps

Phase 1 – Project Setup

1. Installed required tools:
    - Python 3.9+
    - Flask
    - AWS CLI
    - Boto3 (for AWS integration)
2. Created a project folder structure:
3. /smart-drive
4. ├──── app.py
5. ├──── templates/
6. │ ├──── index.html
7. │ ├──── register.html
8. │ ├──── login.html
9. │ └──── dashboard.html
10. ├──── static/
11. ├──── requirements.txt
12. └──── README.md

---

Phase 2 – Cloud Environment Setup

1. Logged into AWS Management Console.
2. Created an S3 bucket to store uploaded files.
3. Configured IAM Role and Policy for the EC2 instance to allow S3 access.
4. Launched an EC2 instance (Ubuntu) and connected via SSH.
5. Installed dependencies on EC2:
6. sudo apt update
7. sudo apt install python3-pip
8. pip install flask boto3

---

Phase 3 – Backend Development (Flask)

1. Built Flask API routes for:
    - /api/auth/register – User registration
    - /api/auth/login – User login with JWT
    - /upload – Upload files to AWS S3
    - /files – List and access uploaded files
2. Used Boto3 library to integrate Flask with AWS S3 for file storage.
3. Stored metadata (filename, user, URL, timestamp) in DynamoDB.

---

Phase 4 – Frontend Development

1. Designed HTML templates:
    - index.html – Landing page
    - register.html – Signup page
    - login.html – Login page
    - dashboard.html – File upload and management page
2. Styled using Bootstrap 5 and added dynamic JavaScript for API calls.

---

Phase 5 – Integration & Testing

1. Linked frontend forms to backend routes using Fetch API.
2. Tested user registration, login, and file upload functionalities.
3. Verified that uploaded files were stored correctly in S3 and URLs worked publicly.
4. Ensured only logged-in users could upload files (using JWT token).

---

Phase 6 – Deployment

1. Deployed Flask app on AWS EC2 with public access (port 5000).
2. Accessed the app using the instance's public IP.

3. Verified that other users could register, log in, and upload files using the public URL.

---

Phase 7 – Cleanup

1. Terminated EC2 instance (to avoid costs).
2. Deleted S3 bucket and DynamoDB table after saving project results.
3. Pushed all code to GitHub for documentation and sharing.

**Security & Optimization**

1. User Authentication

- Implemented JWT (JSON Web Tokens) for secure login and session management.
- Tokens are stored in the client's local storage and verified for every API request.
- Prevents unauthorized access to the dashboard and file upload routes.

---

2. File Access Control

- Files uploaded to AWS S3 are private by default.
- Each uploaded file's access is restricted to the owner (authenticated user).

- Public URLs for file sharing are time-limited pre-signed URLs, ensuring secure temporary access.

---

3. IAM Role & Policy Management

- Created a dedicated IAM Role for the EC2 instance with limited permissions:
    - Only s3:PutObject, s3:GetObject, and s3:ListBucket allowed for the specific bucket.
- Follows the Principle of Least Privilege to minimize security risks.

---

4. Data Encryption

- All data transmission between user and server is encrypted using HTTPS (can be enabled via SSL certificate).
- Files stored in S3 are encrypted at rest using AWS SSE (Server-Side Encryption).

---

5. Input Validation & Error Handling

- Backend sanitizes all input fields (email, password, file names).
- Proper error messages are displayed without exposing server details.
- Prevents SQL Injection, XSS, and CSRF vulnerabilities.

---

6. Performance Optimization

- Implemented asynchronous file uploads using JavaScript to improve user experience.
- Files are served through AWS's global CDN (CloudFront) for faster download speeds (optional).
- Reduced Flask response overhead by using gzip compression and minimal template rendering.

---

7. Scalability & Reliability

- Backend is deployable across multiple EC2 instances with a Load Balancer.
- S3 automatically scales to store unlimited files without affecting performance.
- Can be integrated with AWS Auto Scaling Groups for future growth.

---

8. Backup & Recovery

- S3 provides built-in versioning and replication options for data durability.
- Users' uploaded files can be recovered even if accidentally deleted or overwritten.

## Conclusion

The Cloud-Based Smart File Storage & Sharing System successfully demonstrates how cloud technologies can be used to build a secure, scalable, and user-friendly file management solution similar to Google Drive.

Through the use of AWS services (EC2, S3, IAM) and a Flask backend, the project integrates authentication, file storage, and secure access control seamlessly. Users can easily register, log in, upload files, and generate shareable links — all managed through a clean and intuitive web interface.

The system ensures data confidentiality, integrity, and availability by implementing encryption, IAM-based access control, and token-based authentication. Scalability and performance are achieved through the inherent flexibility of cloud infrastructure.

In summary, this project provides a strong foundation for modern cloud-based applications and highlights key industry practices such as secure cloud architecture, API-driven design, and serverless scalability. It serves as a practical implementation of real-world cloud computing concepts and can be further enhanced with features like user roles, file versioning, and collaboration tools.