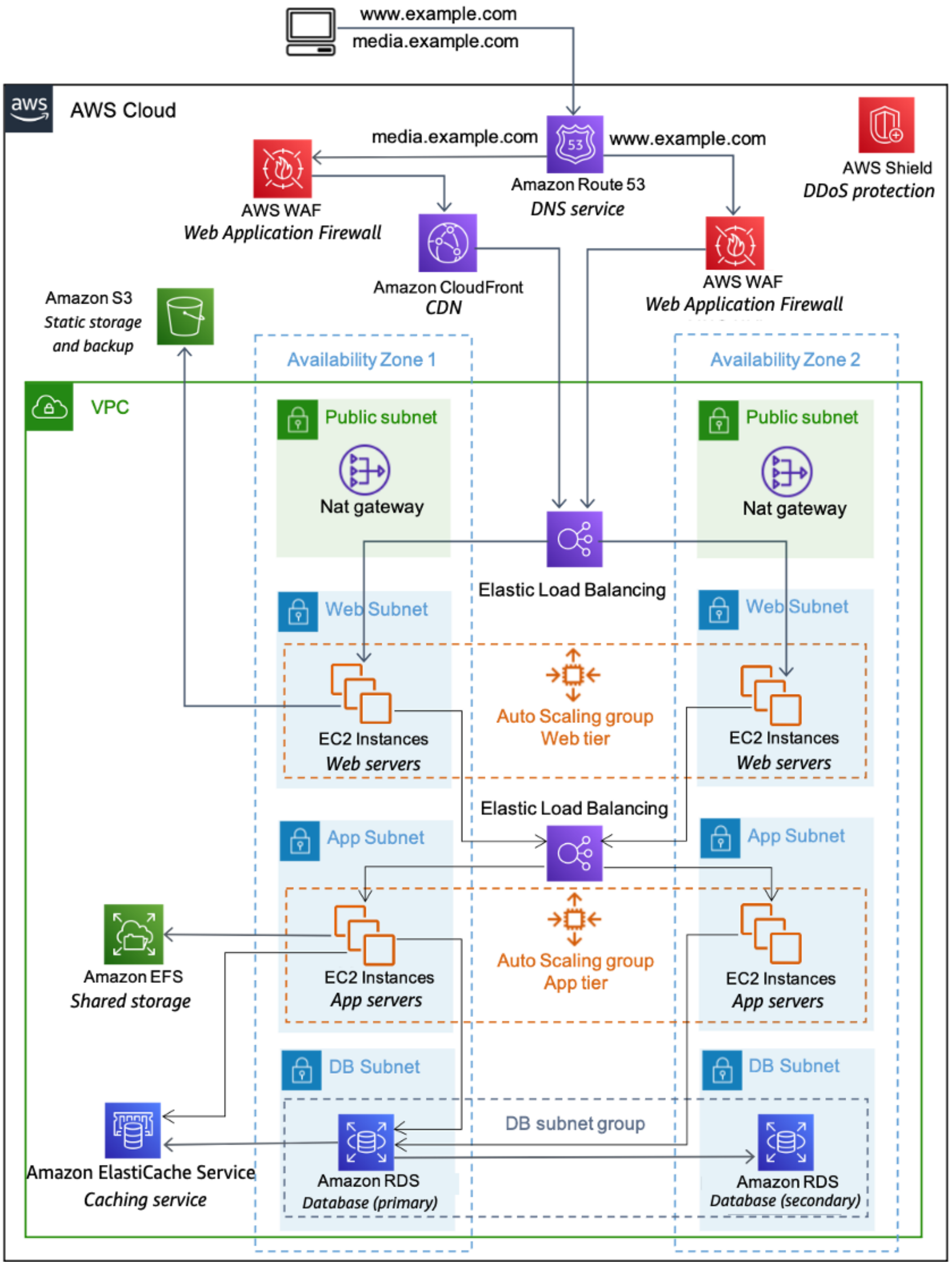
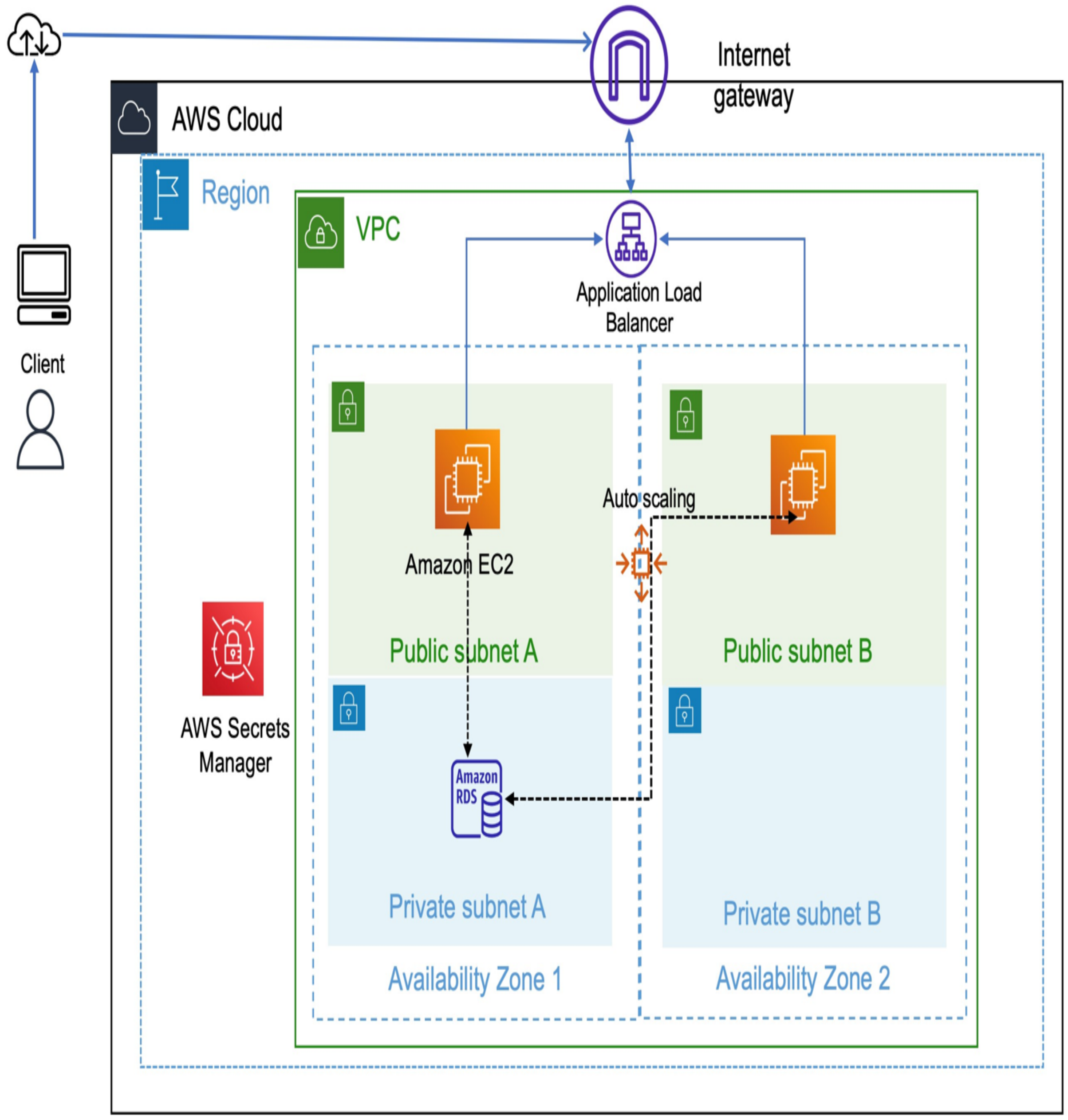
📘 PROJECT REPORT

Cloud Vault – A Secure Cloud-Based File Storage & Sharing System Using Amazon Web Services (AWS)





1. ABSTRACT

Cloud Vault is a secure, scalable, and serverless cloud-based file storage and sharing web application built using Amazon Web Services (AWS).
It allows users to upload, store, download, and share files securely.
Each user has individual login credentials and custom permissions, such as:

Admin: Full control (upload, download, delete, modify permissions)

Uploader: Upload only

Reader: View & download only

Private users: Access only their own files

The system is implemented using AWS Cognito for authentication, API Gateway, AWS Lambda, Amazon S3, and DynamoDB, ensuring high scalability, availability, and fine-grained access control.

2. INTRODUCTION

Cloud storage has become a fundamental requirement for modern digital systems. Traditional storage systems lack scalability, access control, and security.
To address this, Cloud Vault provides a serverless, cost-effective, and secure file management platform using AWS managed services.

Instead of maintaining servers manually, Cloud Vault uses serverless computing where AWS automatically handles infrastructure.

---

3. PROBLEM STATEMENT

- Traditional file storage platforms face several challenges:
- Difficulty in managing user-level file access
- Lack of fine-grained permissions (per-file visibility)
- High cost and maintenance of servers
- No scalability during high traffic
- Security vulnerabilities and data leakage risks

---

4. OBJECTIVES

The goals of Cloud Vault are:

- Build a secure file management system.
- Allow users to upload, download, and view files
- Restrict access based on user role
- Store files securely using AWS S3
- Provide user authentication using Cognito
- Use a fully serverless architecture
- Maintain metadata and permissions in DynamoDB
- Enable sharing files with specific users

---

5. SYSTEM REQUIREMENTS

5.1 Hardware Requirements (Client Side)

Laptop / PC with internet

RAM: Minimum 4 GB

Processor: i3 or above

5.2 Software Requirements

AWS Account

AWS CLI

Python 3.9+

AWS SAM CLI

Text editor / IDE

Web browser (Chrome recommended)

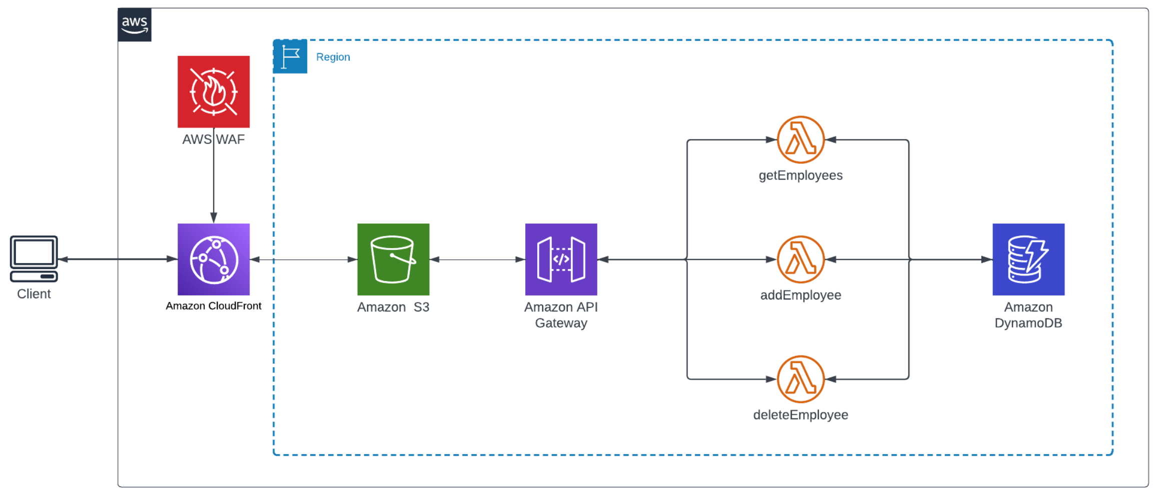
---

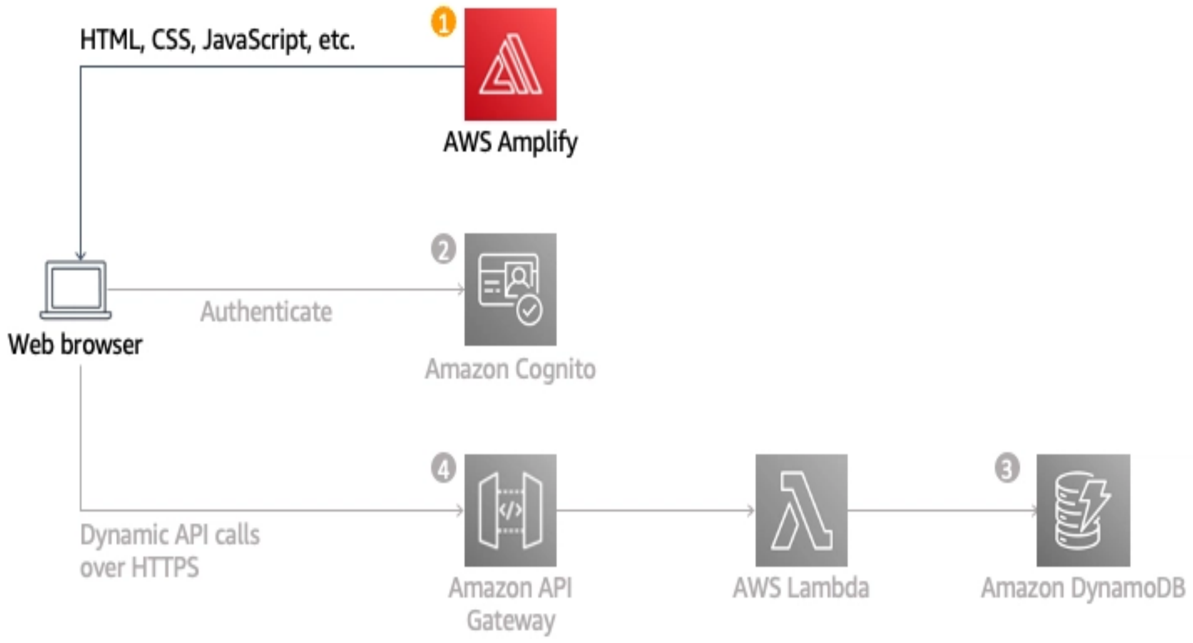
6. PROPOSED SYSTEM

- Cloud Vault follows a serverless cloud architecture providing:
- Secure user authentication
- Central file storage in encrypted S3
- Fine-grained permissions
- Automated metadata storage in DynamoDB
- Serverless API via API Gateway
- Logic implemented using AWS Lambda functions
- Web interface hosted on S3 as a static website

---

7. SYSTEM ARCHITECTURE





7.1 Workflow Overview

User Authentication
User signs in using AWS Cognito Hosted UI.

Token Retrieval
After login, the user receives an ID token (JWT).

API Requests
The frontend calls API Gateway using Authorization: Bearer <token>.

Authorization & Validation
API Gateway validates tokens using Cognito Authorizer.

Lambda Execution
Upload/Download/List/Permission logic is executed in Lambda.

File Storage
Files are stored in Amazon S3 using Presigned URLs.

Metadata Management
File metadata stored in DynamoDB:

Owner ID

Visibility

Allowed users

File name

Path

---

8. AWS SERVICES USED

8.1 Amazon Cognito – User Authentication

Manages users, passwords, and roles.

Provides Hosted UI login page.

Defines groups:

Admins

Uploaders

Readers

Editors

8.2 Amazon S3 – File Storage

Stores user files securely.

Uses presigned URLs for upload and download.

Server-side encrypted.

8.3 AWS Lambda – Backend Functions

Functions:

presign_upload

presign_download

list_files

delete_file

update_permissions

8.4 API Gateway

Acts as HTTPS entry point.

Uses Cognito Authorizer to validate tokens.

8.5 DynamoDB – Metadata Storage

Stores:

File ID

Owner ID

Visibility

AllowedUsers

AllowedGroups

8.6 AWS SAM – Infrastructure as Code

Used to deploy all resources automatically.

---

9. IMPLEMENTATION (STEP-BY-STEP)

(This is the heart of the project — EVERYTHING from start to finish)

STEP 1 — Install & Configure Tools

aws configure

Tools needed:

AWS CLI

AWS SAM CLI

Python 3.9+

STEP 2 — Create Project Folder

```
cloud-vault/
 ├── template.yaml
 ├── lambdas/
 │    ├── presign_upload.py
 │    ├── presign_download.py
 │    ├── list_files.py
 │    ├── delete_file.py
 │    └── update_permissions.py
 └── frontend/
      └── cloud_vault.html
```

Paste all code provided earlier into these files.

STEP 3 — Build & Deploy Using SAM

```
sam build
sam deploy --guided
```

Outputs contain:

API URL

Files bucket name

Frontend bucket name

User Pool ID

Client ID

STEP 4 — Setup Cognito Login (Hosted UI)

Assign a domain:

```
aws cognito-idp create-user-pool-domain ...
```

Set callback URLs to S3 website URL.

STEP 5 — Create Cognito Groups

```
aws cognito-idp create-group --group-name Admins ...
aws cognito-idp create-group --group-name Uploaders ...
aws cognito-idp create-group --group-name Readers ...
aws cognito-idp create-group --group-name Editors ...
```

STEP 6 — Create Users & Assign Groups

```
aws cognito-idp admin-create-user
aws cognito-idp admin-set-user-password
aws cognito-idp admin-add-user-to-group
```

Example:

Alice (Uploader)

Bob (Reader)

Admin (Admin)

STEP 7 — Upload Frontend to S3

```
aws s3 cp frontend/cloud_vault.html s3://<frontend-bucket>/cloud_vault.html --acl public-read
```

Open website:

https://bucket-name.s3-website.region.amazonaws.com/cloud_vault.html

---

STEP 8 — Test Upload Flow

8.1 Get ID Token (Cognito Login via Hosted UI)

User logs in → Page stores ID token.

8.2 Request Presigned Upload URL

```
curl -X POST "$API/files/upload" \
 -H "Authorization: Bearer <id_token>" \
 -d '{ "filename":"test.pdf", "visibility":"private" }'
```

8.3 Upload Using Presigned URL

```
curl -X PUT <presigned_url> --data-binary @test.pdf
```

STEP 9 — Test File Visibility Logic

9.1 List Files for Alice (Owner)

```
curl -H "Authorization: Bearer <alice_token>" $API/files
```

9.2 Update Permissions (Share with Bob)

```
{
 "visibility": "private",
 "allowedUsers": ["bob-sub-id"],
 "allowedGroups": []
}
```

Call API:

```
curl -X PUT "$API/files/<fileId>/permissions" ...
```

9.3 Bob Lists Files

```
curl -H "Authorization: Bearer <bob_token>" $API/files
```

Bob now sees the file.

STEP 10 — Test Download Flow

```
curl "$API/files/<fileId>/download" -H "Authorization: Bearer <token>"
```

Download using presigned URL.

---

10. SECURITY FEATURES

- Cognito authentication ensures secure login
- JWT tokens protect all API calls
- S3 server-side encryption for file safety
- IAM least-privilege Lambda roles
- Fine-grained per-file access control
- Secure presigned URLs (10–15 min expiry)
- No direct S3 credentials exposed

---

11. TESTING & RESULTS

Test Cases Performed

- User login using Cognito Hosted UI
- File upload via presigned URL
- File download via presigned URL
- Different users view only authorized files
- Role-based permissions verified
- Permission changes reflected instantly
- Unauthorized access returns 403 Forbidden
- Files stored and retrieved successfully from S3

Outcome

The system met all functional requirements and performed securely and efficiently.

12. LIMITATIONS

- UI is basic (can be improved with React/Angular)
- Large file uploads depend on presigned URL expiry
- DynamoDB scan used in prototype (GSI recommended for large scale)
- Sharing user lookup requires Cognito Admin permissions

---

13. FUTURE ENHANCEMENTS

Add React / Angular frontend

Add file versioning view in UI

Add folder creation & management

Add email notifications when files are shared

Add multi-factor authentication (MFA)

Analytics using AWS QuickSight

Add search functionality over metadata

---

14. CONCLUSION

Cloud Vault successfully demonstrates how modern cloud architectures can be used to build scalable, secure, and serverless applications.
Using AWS Cognito, Lambda, S3, API Gateway, and DynamoDB, the system achieves:

Secure login

Reliable file storage

Fine-grained permission control

Zero server maintenance

High scalability

This project is an excellent example of cloud-native architecture and is suitable for real-world deployment.

---

15. REFERENCES

AWS Documentation (Cognito, S3, Lambda, API Gateway, DynamoDB)

AWS Serverless Application Model (SAM) Guide

AWS Architecture Best Practices

AWS Whitepaper: Serverless Architectures
