REVA UNIVERSITY
Bengaluru, India

SCHOOL OF COMPUTER SCIENCE AND APPLICATIONS

A Project Report
On

Optimizing Data Integrity: A Cloud Based Redundancy Removal System

Submitted in Partial fulfillment of the requirements for the award of the Degree of

Bachelor of Science (Honors) in Computer Science –
Cloud Computing and Big Data

Submitted by

Bhavya S
Tamanna
Srn
R22DB009
R22DB052

Under the guidance of
Dr.Prof. S. Manju Priya

January 2025

Rukmini Knowledge Park, Kattigenahalli, Yelahanka, Bengaluru-560064
www.reva.edu.in

REVA UNIVERSITY
Bengaluru, India

SCHOOL OF COMPUTER SCIENCE AND APPLICATIONS

CERTIFICATE

The project work titled – '**Optimizing Data Integrity: A Cloud Based Redundancy Removal System',** is being carried out under our guidance by **Bhavya S (R22DB009), Tamanna (R22DB052)**, a bonafide students of REVA University, and is submitting the project report in partial fulfillment, for the award of **Bachelor of Science (Honors) in Computer Science – Cloud Computing and Big Data** during the academic year **2024–25**. The project report has been approved, as it satisfies the academic requirements with respect to the Project Work prescribed for the aforementioned Degree.

Signature with date **Signature with date**

Dr. S. Manju Priya
Internal Guide

Dr. G. Sasikala
Head of the Department-BSc

Signature with Date

Dr C K Lokesh

Director

Name of the Examiner with Affiliation **Signature with Date**

1.

2.

DECLARATION

We, Ms. Bhavya S (R22DB009), Ms. Tamanna (R22DB052), pursuing our **Bachelor of Science (Cloud Computing and Big Data)**, offered by School of Computer Science and Applications, REVA University, declare that this Project title - **"Optimizing Data Integrity: A Cloud Based Redundancy Removal System"**, is the result of the Project Work done by our under the supervision of **Dr.Prof. S. Manju Priya,** at Reva University.

We are submitting this Project Work in partial fulfillment of the requirements for the award of the degree of **Bachelor of Science (CC & BD)** by REVA University, Bengaluru, during the Academic Year 2024-25.

We further declare that this Project Report or any part of it has not been submitted for the award of any other Degree / Diploma of this University or any other University/ Institution.

(Signature of the candidate)

Signed by *me* on:

Certified that this project work submitted by Ms. Bhavya S, Ms. Tamanna has been carried out under our guidance and the declaration made by the candidate is true to the best of our knowledge.

Signature of Internal Guide

Date :……..

Signature of Director of the School

Date :……..
Official Seal of the School

ACKNOWLEDGEMENT

We hereby acknowledge all those, under whose support and encouragement, I have been able to fulfill all our academic commitments successfully. In this regard, I take this opportunity to express our deep sense of gratitude and sincere thanks to School of Computer Science and Applications which has always been a tremendous source of guidance.

We express our sincere gratitude to **Dr. P. SHYAMA RAJU**, Honorable Chancellor, REVA University, and Bengaluru for providing us the state-of-the-art facilities.

We are thankful to **Dr. SANJAY CHITNIS**, Vice Chancellor, REVA University, **Dr K S NARAYANASWAMY,** Registrar, REVA University, Bengaluru for their support and encouragement.

We take this opportunity to express our heartfelt thanks to **Dr C K LOKESH**, Director, School of CSA, REVA University and our sincere thanks to **Dr. G. SASIKALA** Associate Professor, Head of the Department for BSc Program, School of CSA, REVA University, whose encouragement and best wishes provided impetus for the Project Work carried out.

Also, our sincere gratitude goes to our internal guide, **Dr. S. MANJU PRIYA** Professor, School of CSA for the valuable suggestion and constant encouragement towards the completion of this project

Last, but not the least, I thank our parents for their incredible support and encouragement throughout.

ABSTRACT

In the era of cloud computing and big data, ensuring data integrity while managing the ever-growing volume of information has become a critical challenge. Redundant data storage not only increases storage costs but also affects system efficiency and performance. This project, titled "Optimizing Data Integrity: A Cloud-Based Redundancy Removal System," aims to design and implement an efficient system for identifying and removing redundant data in cloud environments. The proposed system leverages advanced algorithms for data deduplication and integrity verification, ensuring that only unique, non-duplicative information is stored across the cloud infrastructure. By integrating cloud-native tools and big data processing frameworks, the system will optimize storage utilization, enhance data integrity, and improve overall system performance. The solution is particularly beneficial for enterprises dealing with large-scale data and helps reduce storage costs while maintaining the accuracy and reliability of critical information. This project will also explore various redundancy detection techniques, such as hash-based , to ensure scalability and efficiency across distributed cloud storage systems.

TABLE OF CONTENTS

1. INTRODUCTION

1.1. Introduction to Project

1.1.1 Statement of the Problem

In cloud computing environments, storing large volumes of data can lead to challenges such as redundancy, inefficient use of storage, and increased costs. Duplicate files uploaded to cloud storage systems not only waste resources but also complicate data management and processing workflows.

This project aims to develop an efficient system for detecting and handling duplicate file uploads in real-time. By leveraging AWS services such as Lambda, S3, and DynamoDB, the solution will ensure optimal storage utilization and maintain the integrity of the data repository. The system will compute unique identifiers for uploaded files, check for existing duplicates in the database, and provide feedback to users if a file is a duplicate or processed successfully.

1.1.2 Brief Description of the Project

The Data Redundancy Removal System is a cloud-based solution designed to optimize data storage by eliminating duplicate files and ensuring efficient file management. The system leverages Amazon Web Services (AWS), utilizing S3 for file storage, DynamoDB for metadata management, and Lambda for automation and computation.

If a duplicate file is detected, the system logs the event in AWS CloudWatch, returns a "Duplicate file detected" message to the user, and avoids storing redundant data.

If the file is unique, its metadata, including the hash, file name, size, and S3 key, is stored in DynamoDB. The system logs the successful upload and sends a "File processed successfully" confirmation to the user.

This project is designed to offer a scalable, reliable, and cost-effective solution by leveraging modern cloud technologies such as AWS S3, AWS Lambda, and DynamoDB to identify and eliminate redundant data in real time. By implementing this system, organizations can optimize their storage costs, enhance performance, and improve overall data management processes.

At its core, the system employs a hash-based deduplication mechanism to detect duplicate files. When a file is uploaded to an AWS S3 bucket, an event is triggered, automatically invoking an AWS Lambda function. This hash serves as a digital signature for the file. The generated hash is then checked against an existing database of hashes stored in AWS DynamoDB.

If no match is detected, the metadata of the file—such as its name, hash, and size—is recorded in the DynamoDB table, and the file is considered unique.

This project also incorporates real-time processing capabilities, ensuring that files are analyzed and managed as soon as they are uploaded. This eliminates latency and enables organizations to maintain up-to-date storage systems without manual intervention.

1.1.3 Software and Hardware Specifications

Software Specifications :

1. Operating System:
    - Compatible with OS for development.
    - AWS Cloud environment for deployment.
2. Programming Language:
    - Python (for Lambda function and backend logic).
3. Development Tools and IDEs:
    - AWS EC2 instance for backend and computation.
    - AWS Management Console for resource configuration.
4. AWS Services:
    - S3: For file storage.
    - DynamoDB: For metadata storage and duplicate detection.
    - Lambda: For serverless computation.
    - CloudWatch: For monitoring logs and debugging.
    - IAM: For setting appropriate permissions.
5. Libraries and Dependencies:
    - boto3: For AWS service interaction.
    - hashlib: For hash calculation of files.
    - json: For parsing and structuring data.

Hardware Specification:

1. Development System:

- Processor: Intel i5 or higher / AMD equivalent.
- RAM: Minimum 8 GB (16 GB recommended for smoother development).
- Storage: Minimum 256 GB (SSD recommended).

2. Internet Connectivity:

- A stable broadband connection for accessing AWS services.

3. Cloud Infrastructure:

- AWS account with sufficient credits or payment setup for resource deployment.
- Region availability for services like S3, Lambda, and DynamoDB.

1.2 Functional and Non-Functional Reqiurement :

Functional Requirements:

1. File Upload: The system should allow users to upload files (e.g., CSV, PDF) via an API.
2. Duplicate Detection: The system should detect and prevent the upload of duplicate files based on file hash.
3. Metadata Storage: The system should store file metadata (file name, size, hash) in DynamoDB.
4. Event Handling: The system should use AWS Lambda to trigger actions upon file upload to S3.
5. API Interaction: The frontend interface (e.g., using React or HTML) should send requests to the API Gateway to trigger Lambda for file processing.

Non-Functional Requirements:

1. Scalability: The system should scale automatically to handle large file uploads and user traffic (using AWS EC2, S3, and Lambda).
2. Security: The system should ensure secure communication using HTTPS and protect sensitive data with encryption.
3. Performance: The system should process files and detect duplicates quickly with minimal latency.
4. Availability: The system should be highly available, using AWS services to ensure minimal downtime.
5. Usability: The user interface should be easy to use, allowing simple file uploads and real-time feedback on status.

2. LITERATURE SURVEY

- Managing Event Ordering and Duplicate Events with Amazon S3 Event Notifications: This AWS blog published by AWS(Amazon Web Services) in the year 2023 explores strategies for handling and avoiding duplicate or outdated Amazon S3 event notifications. It emphasizes the use of a combination of Amazon S3, Amazon EventBridge, AWS Lambda, and Amazon DynamoDB to efficiently manage event ordering and deduplication in event-driven applications. The focus is on ensuring that only the most recent event notifications are processed, preventing the system from acting on stale or duplicate data.
- Data Duplication Removal in Cloud Computing Based on File Checksum : This research proposes a method for data deduplication in cloud computing environments by using a combination of fuzzy algorithms and MD5 checksum algorithms. The approach aims to accurately identify and remove duplicate files, thereby reducing storage costs and enhancing overall system performance. The focus is on leveraging file checksums to detect duplicates efficiently, which can improve the management of data within cloud storage systems , published by James Adegboye and Folahan Jiboku in the year 2022.
- Eliminating Duplicates in Real-Time with AWS Kinesis and Lambda : This research by Ravi Rajendra published in the year 2021 focuses on real-time data deduplication using AWS services such as Kinesis, Lambda, S3, and Redshift.It proposes a cache-based approach where data streams are managed by Kinesis and Lambda is used for processing and deduplication.AWS Kinesis allows for handling large-scale data streams, while AWS Lambda provides the serverless computing environment for quick, scalable data processes.Real-time deduplication is achieved by checking for duplicates as data flows in, which makes it highly suitable for high-volume and dynamic environments.
- Techniques of Data Deduplication for Cloud Storage : This research by Naseer Ali Hussein published in the year 2022 provides a comprehensive review of various data deduplication techniques tailored for cloud storage.It also evaluates the trade-offs between speed, efficiency, and storage savings, providing a deeper understanding of how to select the appropriate deduplication strategy based on specific needs.The paper highlights the cost-effectiveness of deduplication in cloud storage, reducing the amount of storage required and improving management efficiency.
- A Similarity Clustering-Based Deduplication Strategy in Cloud Storage : This paper published by Sangyoon Oh in the year 2020 proposes a similarity clustering-based deduplication strategy to improve the process of detecting and eliminating duplicate data in cloud storage systems.The strategy leverages clustering algorithms to group similar data, making it easier and faster to identify duplicates within these clusters.The paper highlights improved performance in deduplication by minimizing unnecessary comparisons between data blocks, which is a significant issue in traditional methods that compare data across the entire dataset.

3. SYSTEM ANALYSIS

3.1 Existing System

1. Cloud Data Deduplication with Amazon S3 and AWS Lambda : This system leverages Amazon S3 for storage, AWS Lambda for event-driven execution, and DynamoDB for maintaining metadata and tracking duplicates. It works by calculating file hashes and checking for duplicates in real time when new data is uploaded to S3(Aws documentation – 2022).

2. Data Deduplication in Cloud Storage Providers : Popular cloud storage providers like Google Drive and Dropbox implement deduplication techniques at the file or block level. They focus on ensuring that identical data uploaded multiple times is stored only once, improving storage utilization(Dropbox – 2018).

3. Hadoop Distributed File System (HDFS) Deduplication: HDFS employs block-level deduplication in large-scale distributed systems. This reduces redundancy across nodes and optimizes storage in big data applications. Algorithms like MapReduce are often used to manage and identify duplicates efficiently(Apache Hadoop – 2016).

3.2 Limitations of Existing System

1. Cost Efficiency

- The financial implications of using specific tools or services for deduplication.
- Proprietary services like AWS can incur significant costs for high-frequency operations.

2. Privacy and Security

- Whether user data is handled securely and transparently during the deduplication process.
- Centralized metadata management raises privacy concerns, especially in shared or distributed environments.

3. Real-Time vs Batch Processing

- The ability of the system to handle real-time data streams versus batch deduplication.
- Real-time systems often face computational overhead, while batch systems lack immediacy.

4. Ease of Use and Adaptability

- The flexibility of the system to adapt to different user needs or environments.
- Vendor lock-in and rigid configurations limit adaptability in proprietary systems.

5. Performance Overhead

- The additional resources (e.g., CPU, memory) required to perform deduplication.
- High resource usage reduces efficiency, especially in distributed systems like Hadoop.

These criteria form the basis for identifying limitations in existing systems, helping to pinpoint the need for a more effective and adaptable Data Redundancy Removal System.

3.3 Proposed System

The proposed Data Redundancy Removal System is designed to detect and eliminate redundant data in cloud storage environments efficiently. The system aims to optimize storage utilization, improve performance, and reduce costs while ensuring data integrity and security.

1. Hash-Based Deduplication :Uses advanced cryptographic algorithms (e.g., SHA-256) to generate unique file hashes for detecting duplicate files.Ensures accuracy and reliability in identifying redundant data.

2. Integration with AWS Services :Amazon S3 for file storage.

- AWS Lambda for real-time data processing.
- Amazon DynamoDB for storing metadata like file hashes and references.

3. Real-Time Detection :Automatically detects duplicate files upon upload using AWS S3 event triggers.Responds instantly by validating file hashes and preventing unnecessary duplication.

4. Metadata Storage :Maintains a record of file metadata (e.g., hash, file size, upload date) in DynamoDB for quick lookup and management.

5. Scalable Architecture :Built on cloud-native technologies to ensure scalability and reliability for growing datasets.

6. Cost Optimization :Minimizes storage costs by eliminating duplicate files at the source.Reduces computational overhead using lightweight, serverless functions like Lambda.

3.4 Advantages of Proposed System

1. Storage Efficiency : By removing redundant data at both the point of entry (inline) and retrospectively (post-process), the system significantly reduces storage requirements.This ensures lower costs for cloud storage and physical infrastructure.

2. Improved System Performance : Inline deduplication minimizes the data footprint early on, reducing the burden on network bandwidth and storage I/O, improving overall system performance.Post-process deduplication ensures that older, legacy data is also optimized without interrupting active workloads.

3. Dynamic and Adaptive : The integration of machine learning allows the system to adapt and improve over time, making future deduplication efforts more efficient based on past patterns. It adjusts deduplication policies dynamically, depending on the organization’s evolving data needs.

4. Enhanced Data Security : Built-in encryption ensures that even deduplicated data remains secure, addressing common security concerns with data deduplication systems.

5. Data Integrity Assurance : The indexing and data integrity module ensures that all unique data is accurately referenced and accessible, preventing any issues that could arise from deduplication, such as data loss or corruption.

This proposed system addresses storage efficiency, dynamic adaptability, and security concerns, making it a robust solution for organizations dealing with large-scale data.

3.5 Feasibility study

3.5.1 Technical Feasibility :

- Cloud Infrastructure :The project relies on AWS services like S3, Lambda, and DynamoDB, which are reliable, scalable, and industry-standard cloud solutions.
- Hashing Algorithms:The use of secure hashing algorithms like SHA-256 ensures accurate detection of redundant files. These algorithms are computationally efficient and widely supported.
- Programming Tools :The system is implemented in Python, a versatile and well-supported programming language, with libraries like boto3 for AWS service Technical integration.
- Scalability :The architecture is serverless, allowing it to handle increased workloads without significant changes or infrastructure management.

3.5.2 Economic Feasibility:

- Cost of Services:AWS services operate on a pay-as-you-go model, meaning costs are incurred only for usage.DynamoDB storage and Lambda invocation charges are minimal for moderate data volumes.
- Implementation Costs:Initial development costs include programming, testing, and integration.No need for dedicated hardware, reducing upfront investment.
- Cost Savings:By eliminating redundant files, the system significantly reduces storage costs.Efficient storage management translates into long-term savings.

3.5.3 Operational Feasibility :

- Ease of Use:The system integrates seamlessly with existing AWS workflows.Automated deduplication ensures minimal manual intervention.
- Maintenance:AWS services handle infrastructure management, reducing the operational burden.Regular monitoring and updates to algorithms and metadata tables ensure smooth operation.
- User Training:Minimal training is required for users familiar with AWS and basic cloud concepts.

In conclusion , The system is operationally feasible and easy to maintain, with high usability for the target users.

4. SYSTEM DESIGN AND DEVELOPMENT

4.1 High Level Design (Architectural) :

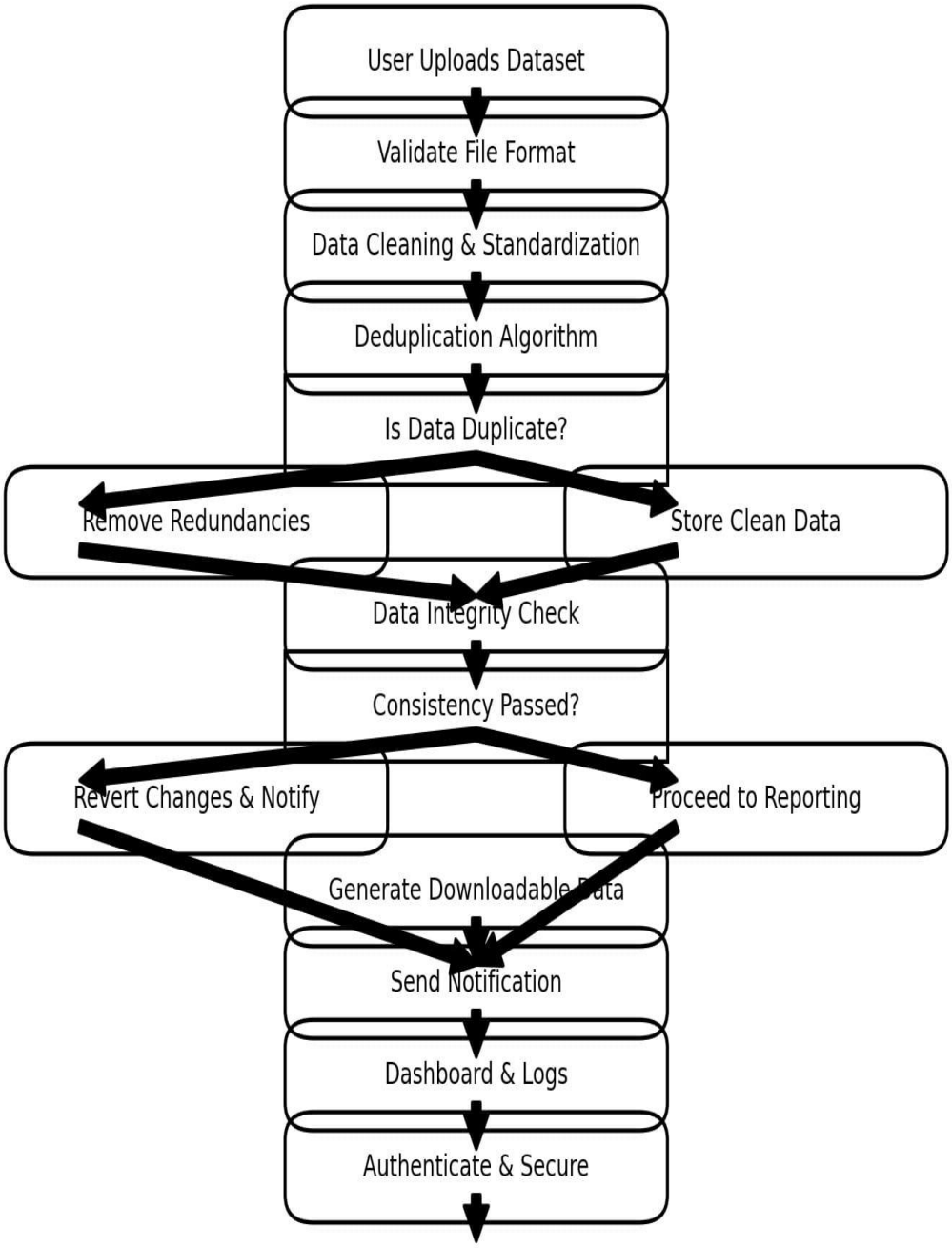


Fig : 1.0

Figure description:

- User Uploads Dataset: The process begins with the user uploading a dataset.
- Validate File Format: The system validates the format of the uploaded file to ensure it meets the requirements.
- Data Cleaning & Standardization: The data is cleaned and standardized to ensure uniformity and remove inconsistencies.
- Deduplication Algorithm: A deduplication algorithm is applied to identify duplicate records.
- Is Data Duplicate?: A decision point checks if the data contains duplicates:
- Data Integrity Check: The system performs an integrity check on the data to verify its accuracy.
- Consistency Passed?: Another decision point checks if the data consistency checks are passed:
- Generate Downloadable Data: After passing consistency checks, the system generates a downloadable version of the data.
- Send Notification: The user is notified about the data status.
- Dashboard & Logs: The process logs activities and provides a dashboard for user access.
- Authenticate & Secure: The workflow concludes by securing and authenticating the processed data.

4.2 Usecase Diagram :

- User : Represents the individual or entity interacting with the system to perform file-related tasks such as uploading, processing, and retrieving data.
- System Administrator :Responsible for maintaining and monitoring the system to ensure its reliability and efficiency.
- Remove Redundancy :The user initiates this process to detect and eliminate duplicate files from the system.The system uses file hashes to identify duplicates and removes redundant data, ensuring storage efficiency.
- Upload Data :The user uploads files to the system for redundancy checks and storage.The system validates the uploaded data and prepares it for further processing.
- Store Processed Data :The system stores non-redundant data in cloud storage (e.g., AWS S3).Metadata about the files is also saved in a database (e.g., DynamoDB) for quick retrieval and management.
- Retrieve Processed Data : The user retrieves the processed, non-redundant files from cloud storage.This use case allows access to the stored data for download or further usage.

- Monitor System Performance : The system administrator monitors system logs, redundancy removal efficiency, storage usage, and performance metrics.This ensures the system operates optimally and identifies any potential issues.
- System Administrator : Focuses solely on "Monitor System Performance" to ensure the smooth functioning of the system.Ensures the user experience is seamless by addressing technical issues and optimizing system performance.

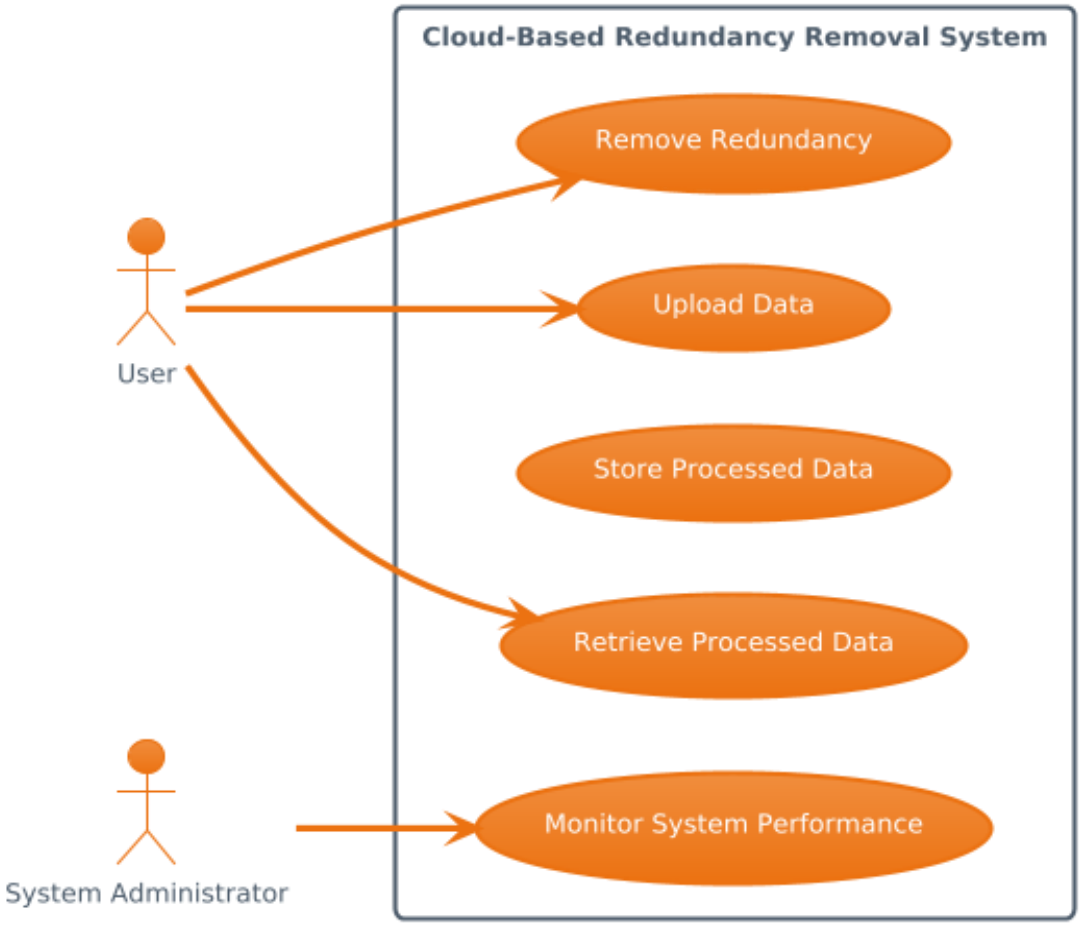


Fig : 2.0

4.3 Sequence Diagram :

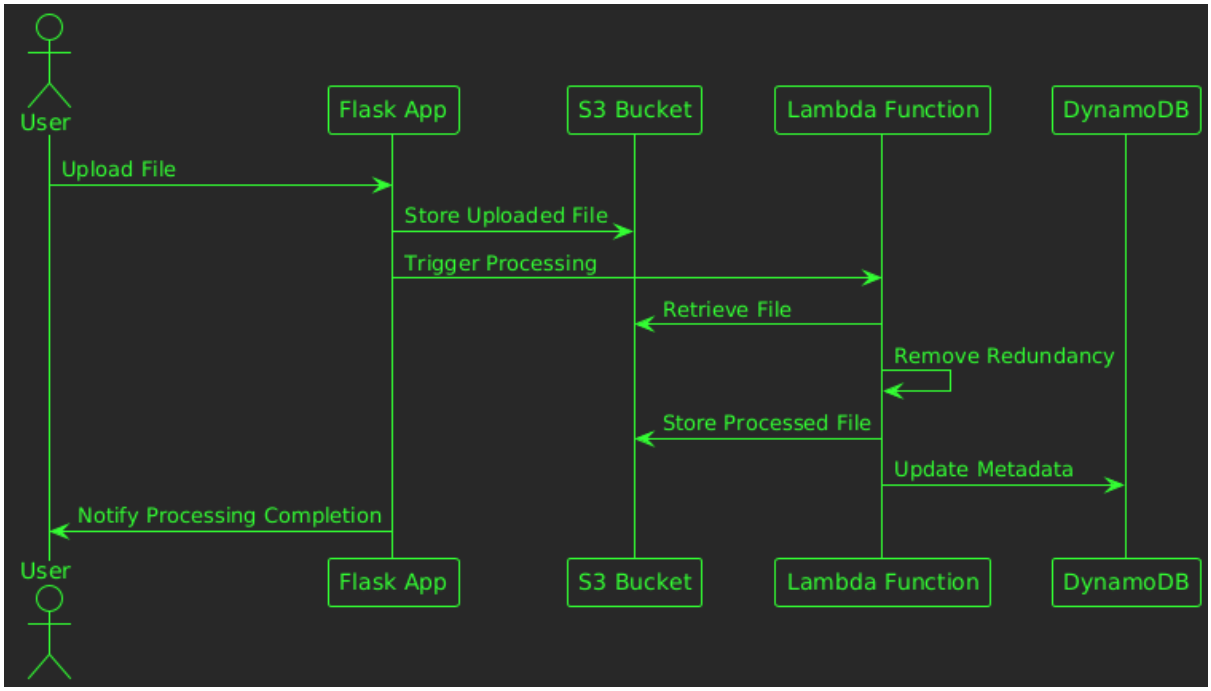


Fig : 3.0

Figure Description :

Actors and Components:

User: The end-user interacting with the system via the web interface (Flask App).

Flask App: The web application responsible for handling user requests and coordinating the process.

S3 Bucket: AWS Simple Storage Service (S3) used for storing uploaded and processed files.

Lambda Function: AWS Lambda, which processes the uploaded file to remove redundancy.

DynamoDB: AWS DynamoDB used to store metadata or logs related to the processed files.

Workflow :

- User : Upload File,the user initiates the process by uploading a file via the Flask application interface.
- S3 Bucket: Store Uploaded File ,the Flask app sends the uploaded file to the S3 bucket for temporary storage.
- Lambda Function: Trigger Processing,the Flask app triggers the Lambda function to begin processing the file.
- S3 Bucket: Retrieve File ,the Lambda function retrieves the uploaded file from the S3 bucket for processing.

- Lambda Function: Remove Redundancy ,the Lambda function executes the redundancy removal algorithm on the retrieved file.
- S3 Bucket: Store Processed File ,after processing, the redundancy-free file is saved back into the S3 bucket.
- DynamoDB: Update Metadata ,the Lambda function updates the metadata or logs related to the processed file in DynamoDB.
- User: Notify Processing Completion , once the process is complete, the Flask app notifies the user and provides a link to download the processed file.

4.4 Class Diagram :

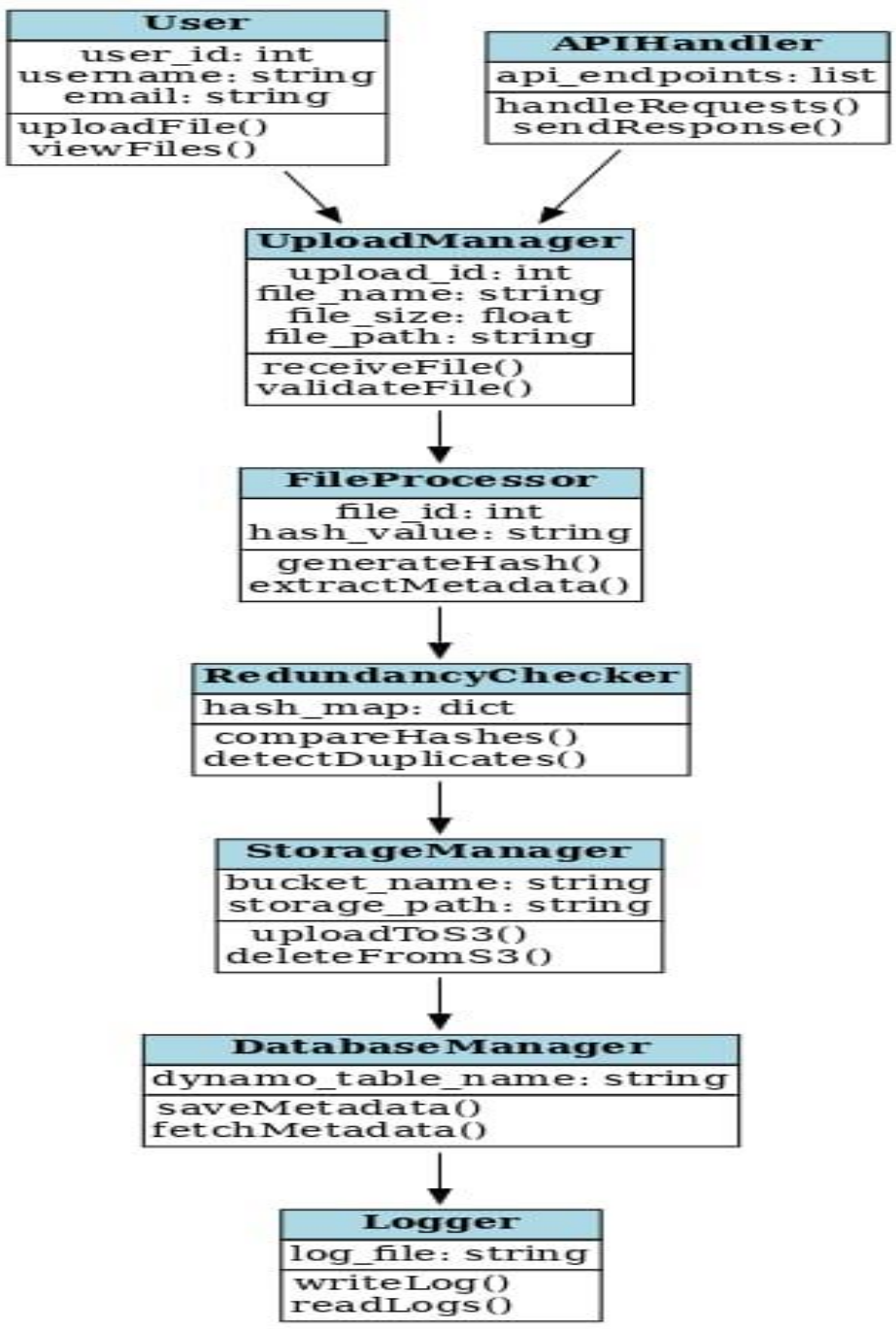


Fig : 4.0

Figure Description :

- User : Represents an end-user who interacts with the system.
  Attribute : user_id , username, email
  uploadFile() - Allows users to upload files to the system.

viewFiles() - Enables users to view their uploaded files.

- APIHandler : Serves as the interface for handling external requests from the user to the system.
  Attribute : api_endpoints
  handleRequests() - Parses and routes incoming API requests to the appropriate modules.
  sendResponse() - Sends back responses (success, failure, or data) to the user.

- UploadManager : Manages file uploads and validates the files before processing.
  Attributes : upload_id , file_name ,file_size , file_path .
  receiveFile() - Accepts the file from the user/API and stores it for processing.
  validateFile() - Checks the file's format, size, and integrity.

- FileProcessor : Processes the uploaded file to generate metadata and check redundancy.
  Attributes: file_id ,hash_value
  generateHash() - Creates a hash value for the file based on its content.
  extractMetadata() - Extracts metadata such as file type, size, and creation date.

- RedundancyChecker : Identifies duplicate files by comparing hashes.
  Attribute:hash_map
  compareHashes() - Compares the hash of the newly uploaded file with existing hashes.
  detectDuplicates() - Flags duplicate files and provides the associated metadata.

- StorageManager : Handles storage and retrieval of files in AWS S3.
  Attributes: bucket_name ,storage_path
  uploadToS3() - Uploads a validated and non-duplicate file to the S3 bucket.
  deleteFromS3() - Deletes a file from the S3 bucket, if required.

- DatabaseManager : Manages metadata storage and retrieval from AWS DynamoDB.
  Attribute : dynamo_table_name
  saveMetadata() - Saves file metadata (e.g., hash, size, user information) in the database.
  fetchMetadata() - Retrieves file metadata based on queries like user ID or file hash.

- Logger : Maintains logs of system events, errors, and file processing activities.
  Attribute : log_file
  writeLog() - Writes a new log entry for significant system events or errors.
  readLogs() - Allows reading log entries for debugging or monitoring purposes.

Interaction and Workflow :

- File Upload : A user initiates a file upload via uploadFile() in the User class.The APIHandler routes the request to the UploadManager, which validates the file.

- File Processing : The FileProcessor generates a hash for the file and extracts metadata.The RedundancyChecker uses the hash to check for duplicates. If the file is unique, it proceeds; otherwise, the duplicate is flagged.

- Storage : Unique files are sent to the StorageManager, which uploads them to AWS S3.The DatabaseManager stores metadata in DynamoDB for tracking and retrieval.

- Logging:Throughout the process, the Logger logs important events, such as uploads, duplicates detected, or storage operations.

4.5 Input/Output Interface Design :

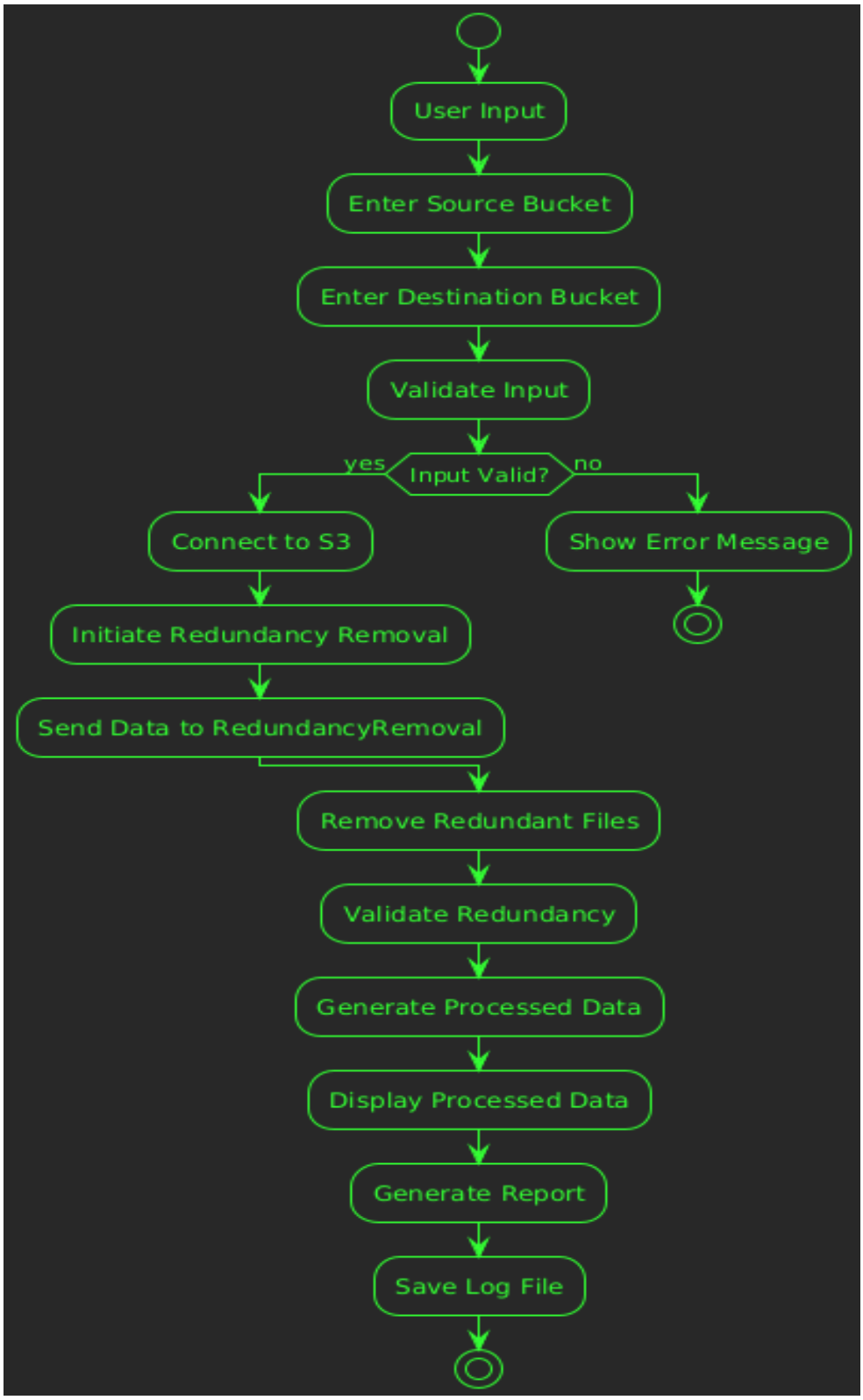


Fig : 5.0

Figure description :

- User Input:The process begins with the user providing necessary input details.
- Enter Source Bucket:The user specifies the source bucket from which data will be accessed.
- Enter Destination Bucket:The user specifies the destination bucket where the processed data will be stored.
- Validate Input:The system checks whether the provided inputs are valid.
- Connect to S3:Establishes a connection to the specified AWS S3 buckets.
- Initiate Redundancy Removal:The redundancy removal process is initialized.
- Send Data to Redundancy Removal:The system sends the necessary data to the redundancy removal component.
- Remove Redundant Files:The system identifies and removes redundant files from the dataset.
- Validate Redundancy:Ensures that redundancy has been successfully eliminated.
- Generate Processed Data:The cleaned and processed data is generated.
- Display Processed Data:The system displays the processed data for user review.
- Generate Report:A report summarizing the process and results is generated.
- Save Log File:A log file containing details of the operation is saved for future reference.
- End:The process concludes.

4.6 Module Description :

- User Interface (UI) Module : This module provides a user-friendly interface for uploading files to the system. Users can upload data through a web application or CLI, and the module validates the input format and size. It also displays processing results, such as whether a file is new or duplicate.
- File Processing : It handles the extraction of file details and computes a unique checksum for each file. The checksum serves as a unique identifier to detect duplicate files.
- Cloud Storage Integration Module : This module manages the integration with Amazon S3 to upload and retrieve files. It ensures files are stored securely and retrieves file metadata for processing.
- Deduplication Management Module : This module compares the generated file checksum against existing records in a database (e.g., DynamoDB). If a match is found, it flags the file as a duplicate; otherwise, it stores the file metadata and checksum
- Logging and Monitoring Module : This module captures logs of all system activities and monitors performance. Logs are stored in Amazon CloudWatch and used for debugging and performance optimization.
- Security and Access Control Module:This module ensures that the system is secure, with proper authentication and access control mechanisms for data uploads and retrievals.
- System Administration Module:This module provides administrative tools for managing the system, including manual uploads, metadata corrections, and maintenance tasks .

5.CODING

5.1 Pseudo Code

Install Dependencies on EC2:

sudo su

sudo apt update && sudo apt upgrade -y

Install and Configure AWS CLI :

Curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip"-o "awscliv2.zip"

unzip awscliv2.zip

sudo ./aws/install

aws --version

aws configure

Install and creating environment for backend development:

Install python and pip :

sudo apt install python3 python3-pip -y

Create virtual environment :

apt install python3.12-venv

python3 -m venv myenv

source myenv/bin/activate

Install flask and boto3:

pip3 install flask boto3

Create a project directory :

mkdir ~/flask-app

cd ~/flask-app

Create the app.py :

nano app.py

Flask Code:

```
from flask import Flask, render_template, request, jsonify
import boto3
import os

app = Flask(__name__)

# AWS Configuration
s3_client = boto3.client('s3')
dynamodb = boto3.resource('dynamodb')
s3_bucket_name = 'your-s3-bucket-name'
table = dynamodb.Table('FileMetadata')

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/upload', methods=['POST'])
def upload_file():
    if 'file' not in request.files:
        return jsonify({'message': 'No file uploaded'}), 400

    file = request.files['file']
    if file.filename == '':
```

```
        return jsonify({'message': 'No file selected'}), 400

    # Save file temporarily
    tmp_file_path = os.path.join('/tmp', file.filename)
    file.save(tmp_file_path)

    # Upload file to S3
    s3_client.upload_file(tmp_file_path, s3_bucket_name, file.filename)

    # Return success message
    return jsonify({'message': 'File uploaded successfully'}), 200

@app.route('/files', methods=['GET'])
def get_files():
    response = table.scan()
    files = response.get('Items', [])
    return jsonify({'files': files})

if __name__ == '__main__':
    app.run(debug=True)
```

Creating HTML Template :

Create template folder:

mkdir template

Create the index.html :

nano templates/index.html

Html code :

```
<!DOCTYPE html>
<html>
<head>
    <title>Data Redundancy Removal</title>
</head>
<body>
    <h1>Upload File</h1>
    <form id="uploadForm" method="POST" enctype="multipart/form-data" action="/upload">
        <input type="file" name="file">
        <button type="submit">Upload</button>
    </form>

    <h1>Uploaded Files</h1>
    <button onclick="fetchFiles()">Refresh List</button>
    <ul id="fileList"></ul>

    <script>
        async function fetchFiles() {
            const response = await fetch('/files');
            const data = await response.json();
            const fileList = document.getElementById('fileList');
            fileList.innerHTML = '';
            data.files.forEach(file => {
                const li = document.createElement('li');
                li.textContent = `${file.FileName} (${file.FileHash ? 'Unique' : 'Duplicate'})`;
                fileList.appendChild(li);
```

```
        });
    }
  </script>
</body>
</html>
```

Run the Flask app :

```
# python3 app.py
```

Integration and deployment :

Deploy your Flask application:

```
# pip3 install gunicorn
```

Run the flask with gunicorn :

```
# gunicorn -w  4  app:app
```

Configure NGINX :

```
# sudo apt-get install nginx
# Edit : /etc/nginx/sites-available/your-app
```

Nginx Code:

```
server {
  listen 80;
  server_name <your-ec2-public-ip-or-domain>;

  location / {
    proxy_pass http://127.0.0.1:8000;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
```

```
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

Monitoring and Testing Frontend development :

S3 operations :

```
# aws s3 ls s3://your-bucket-name
```

Test the lambda function :

```
# aws s3 cp yourfile.pdf s3://my-file-storage-bucket/yourfile.pdf
```

Test the workflow :

```
# aws s3 cp testfile.pdf s3://your-bucket-name/
```

6. SOFTWARE TESTING

AWS S3 :

1. Log In to AWS Management Console

2. Sign in using your credentials :

- Navigate to the S3 Service
- In the AWS Management Console, search for S3 in the search bar.
- Click on the Amazon S3 service.
- Create a New Bucket
- On the S3 dashboard, click the "Create bucket" button.

3.Bucket Settings :

- Enter a unique name for your bucket (e.g., my-data-redundancy-bucket).
- Bucket names must be globally unique and should not contain spaces or uppercase letters.

4. Region :

- Choose the AWS Region where the bucket will be created (e.g., us-east-1).
- Pick a region close to your users or application to reduce latency.

5. Configure Bucket Options :

- Block Public Access :Leave the "Block all public access" option enabled unless you want to make the bucket public.

6.Review and Create :

- Review all the configurations to ensure they meet your requirements.
- Click "Create bucket"

7. Verify Bucket Creation :

- Once the bucket is created, you will see it listed in the S3 dashboard.
- Click on the bucket name to configure further settings or upload files.

AWS DYNAMODB :

1.Log in to the AWS Management Console

2. Navigate to DynamoDB :

- Search for DynamoDB in the AWS services search bar.
- Click on DynamoDB to access the service dashboard.

3. Create a New Table :

- On the DynamoDB dashboard, click "Create table".
- Enter a unique name for your table (e.g., DataRedundancyTable).
- Specify the primary key to uniquely identify each item in the table.
- Choose one of the following:

- Partition Key (Hash Key): A unique attribute for partitioning data (e.g., FileID).
- Partition Key + Sort Key (Hash + Range Key): Use an additional key for sorting (e.g., FileID and Version).

4. Index Settings:

- By default, the primary key is indexed.
- Optionally, add Global Secondary Indexes (GSI) or Local Secondary Indexes (LSI) to optimize queries on non-primary key attributes.

5.Configure Capacity Settings :

Read/Write Capacity Mode:

- On-Demand: Pay per request (recommended for unpredictable workloads).
- Provisioned: Specify the number of read and write capacity units for predictable workloads.

- Example: 5 RCU (Read Capacity Units) and 5 WCU (Write Capacity Units).

6. Review and Create :

- Review all the configurations.
- Click "Create table".

7. Verify Table Creation :

- Wait for the table status to change to Active in the DynamoDB dashboard.
- Click on the table name to view and configure additional settings.

AWS LAMBDA :

1.Go to the AWS Lambda Console:

- Open the AWS Lambda Console.

2.Create a New Lambda Function:

- Click on Create function.
- Choose Author from scratch.
- Provide a Function name (e.g., FileRedundancyCheck).
- Select the Runtime (e.g., Python 3.x).
- Under Permissions, choose to create a new role with basic Lambda permissions or use an existing role if you already have one.
- Click Create function to create the Lambda

3.Create an S3 Trigger for Lambda:

- Configure S3 to Trigger the Lambda Function
- In the Lambda function dashboard, go to the Triggers section and click Add trigger.
- Select S3 as the trigger source.
- Choose the S3 bucket you want to use for file uploads.
- Set the event type to Object Created (All Events) so that the Lambda function is triggered when a file is uploaded.

4.Save and Test:

- Click Add to create the trigger.
- Now, when a file is uploaded to the S3 bucket, the Lambda function will automatically be triggered to process the file.

AWS EC2 :

Configure the EC2 instance with appropriate security groups and IAM roles for accessing AWS services.

1. Launch an EC2 Instance:

- Go to the AWS EC2 Console.
- Click Launch Instance.
- Select an Amazon Linux 2 or Ubuntu AMI (Amazon Machine Image).
- Choose an appropriate instance type (e.g., t2.micro for testing).
- Configure your instance (use the default settings or customize them as needed).
- Security Group: Open port 80 (HTTP) and 22 (SSH) for web and SSH access.
- Click Launch and download the key pair (if you don't have one already).

2. SSH into the EC2 Instance:

- In the EC2 Console, find your instance.
- Click Connect, and follow the SSH instructions.
- For example, use the following command (assuming you're using Linux/macOS):

```
# ssh -i /path/to/your-key.pem ec2-user@<EC2-PUBLIC-IP>
```

AWS API GATEWAY :

1.Create an API Gateway:

- Go to API Gateway in the AWS Console.
- Click Create API and select REST API.
- Choose New API and give it a name (e.g., FileUploadAPI).
- Click Create API.

2.Create a POST Method:

- Click Actions > Create Resource.
- Name the resource (e.g., /upload), and click Create Resource.
- Select the /upload resource and click Create Method > POST.
- Select Lambda Function as the integration type, and choose your Lambda function.
- Click Save, then OK.

3.Deploy the API:

- Click Actions > Deploy API.
- Select a deployment stage (e.g., prod) and click Deploy.

4.Test the API:

- Get the Invoke URL from the Stages section.
- Use Postman or a web client to send a POST request to your API endpoint.

AWS IAM :

1. Sign in to the AWS Management Console :

- Go to the AWS Management Console and sign in with your admin credentials.

2. Navigate to IAM Service :

- In the AWS Console, type "IAM" in the search bar and select IAM under Services to open the Identity and Access Management (IAM) console.

3. Create a New IAM User :

- In the IAM dashboard, click on Users in the left sidebar, then click the Add user button at the top.

4. Set User Details :

- Enter the User name for the new IAM user.
- Choose the Access type:
  Programmatic access: Allows access through the AWS CLI, SDKs, or API.
  AWS Management Console access: Provides access to the AWS Console. You'll need to set a password for console access.
- Click Next: Permissions.

5. Attach Policies to the User :

- You can attach permissions to the user in multiple ways:
- Attach policies directly: Choose from predefined AWS policies such as AdministratorAccess, ReadOnlyAccess, etc.
- Add user to a group: If you have predefined IAM groups (with attached policies), add the user to an appropriate group.
- Copy permissions from existing user: Copy permissions from another user in your AWS account.
- Attach customer managed policies: If you've created custom policies, attach them here.
- Select the appropriate policies or group for the user and click Next: Tags.

6. Add Tags (Optional) :

- Tags are optional metadata that can be used to organize and track your users. Add key-value pairs for tags if necessary, and click Next: Review.

7. Review the User and Create :

- Review the settings for the user (name, permissions, etc.).
- After reviewing, click Create user.

8. Save the Credentials :

- After the user is created, you'll see a success screen with the user's credentials:
- Access key ID and Secret access key (for programmatic access)
- Password (if console access was enabled)
- Login URL for console access.

AWS CLOUDWATCH :

1. Access AWS CloudWatch :

- Sign in to AWS Management Console and navigate to the CloudWatch Dashboard.
- From the dashboard, you can manage metrics, logs, and alarms.

2. Create a Log Group :

- In the CloudWatch dashboard, go to the Logs section.
- Click on Log Groups > Create log group.
- Enter a name for your log group, such as /data-redundancy-system/logs.
- Choose a retention period for your logs (e.g., 1 week, 1 month, or indefinite). Logs older than the retention period will be automatically deleted.

3. Enable Logging in AWS Lambda :

- Go to the AWS Lambda service in the AWS Management Console.
- Select your Lambda function used for data redundancy removal.
- Scroll to the Monitor tab and ensure the Amazon CloudWatch Logs section is enabled.
- AWS Lambda automatically creates a log group in CloudWatch Logs when it runs. You can view it in the CloudWatch console.

4. Verify Logs :

- After running your Lambda function, return to the CloudWatch Logs section.
- Click on the appropriate Log Group (e.g., /aws/lambda/<lambda-function-name>).
- Open a Log Stream to view specific logs generated by your Lambda function. Each stream corresponds to an execution.

5. Create Metrics to Monitor System Performance :

- In the CloudWatch Metrics section, select Create metric.
- Choose relevant AWS services, such as Lambda or DynamoDB, to monitor. Examples include:
    1. Lambda: Invocation count, duration, errors.
    2. DynamoDB: Read/write throughput, throttling.
- Define a metric filter if you want to track custom log entries, such as "File processed successfully."

6. Set Up Alarms for Notifications :

- Go to the Alarms section in CloudWatch.
- Click Create Alarm.
- Choose a metric to monitor, such as Errors or Throttles for Lambda functions.
- Set a threshold (e.g., if errors exceed 5 in 1 minute).
- Configure a notification action using Amazon SNS (Simple Notification Service) to send alerts via email or SMS.

7. Automate Insights Using CloudWatch Logs Insights :

- In the Logs Insights section, write queries to analyze logs
- Use these queries to analyze patterns or troubleshoot issues.

8. Integrate CloudWatch with Other AWS Services :

- S3 Event Notifications: Enable S3 to log events into CloudWatch for file uploads.
- DynamoDB: Monitor table activity using CloudWatch metrics.
- Custom Logs: Use boto3 in your Python code to send custom logs directly to CloudWatch using PutLogEvents.

9. Test and Validate :

- Upload test files to S3 or trigger your Lambda function.
- Check CloudWatch logs for expected outputs:
    1. Success: "File processed successfully."
    2. Duplicate: "Duplicate file detected."
- Verify alarms and notifications are triggered under predefined conditions.

10. Optimize and Scale :

- Archive Logs: Set up log archiving to Amazon S3 for long-term storage.
- Cost Optimization: Monitor CloudWatch usage to avoid unnecessary costs for log storage or metric queries**.**

7.CONCLUSION AND SCOPE FOR FUTURE ENHANCEMENT

Conclusion :

The Data Redundancy Removal System presented in this project demonstrates a practical approach to optimizing cloud storage and computational efficiency by reducing redundant data. Through the integration of advanced data deduplication algorithms and cloud computing resources such as Amazon EC2, the system achieves significant storage savings, thereby enhancing resource utilization and lowering operational costs.

This project highlights the potential of leveraging big data techniques for real-world cloud computing challenges. By implementing robust methods for identifying and removing redundancy, the system ensures data integrity and seamless accessibility while addressing the scalability needs of modern cloud infrastructure.

The results obtained affirm the viability of this approach in mitigating the challenges associated with data storage overhead in a cloud environment. Further enhancements could include extending the system to support real-time redundancy detection, integrating machine learning models for predictive analysis, and ensuring compatibility with diverse cloud service providers.

In conclusion, this project serves as a foundation for future research and development in data optimization for cloud computing, contributing to sustainable and efficient cloud solutions.

Future Enhancements :

1. Advanced Error Handling:Improve error handling with retries, better logging, and alerting for specific failure scenarios using AWS CloudWatch Alarms or SNS.
2. Versioning in S3:Enable S3 versioning to track different versions of the same file. This helps in maintaining a history of file changes.
3. File Validation:Implement validation for file formats or data integrity checks before processing the file (e.g., CSV schema validation, file size checks).
4. Scaling and Optimization: - Optimize Lambda function performance for large files or high-frequency uploads, such as splitting file processing into smaller chunks or optimizing DynamoDB read/write throughput.
5. User Authentication & Authorization: Add a user interface or API where users can upload files securely. Use Amazon Cognito or API Gateway for user authentication.
6. Cost Optimization:Monitor and optimize the AWS resources being used (S3, Lambda, DynamoDB) to reduce costs, especially when handling large amounts of data.
7. 9.Frontend Interface: Add a user-friendly web interface (using React or Vue.js) to allow users to interact with the system, upload files, and check if duplicates exist.

8. BIBILIOGRAPHY

Books:

1. "Amazon Web Services in Action" , Michael Wittig and Andreas Wittig - This book offers a comprehensive guide to AWS services, including S3, DynamoDB, Lambda, and more. It provides practical examples and in-depth explanations that would be valuable for thisproject.

2. "AWS Certified Solutions Architect Official Study Guide" ,Joe Baron, Hisham Baz, and Tim Bixler - A great resource for understanding AWS architecture and services, which could be useful when integrating your application with AWS components like EC2, S3, and Lambda.

3. "Cloud Computing: Concepts, Technology & Architecture" ,Thomas Erl - This book covers cloud computing fundamentals, which will give you a deeper understanding of the cloud concepts used in this project.

Websites:

1. AWS Documentation *(https://docs.aws.amazon.com/)-* This is the official AWS documentation, providing details about all AWS services, including Lambda, S3, DynamoDB, and more. This is the most reliable and up-to-date resource.

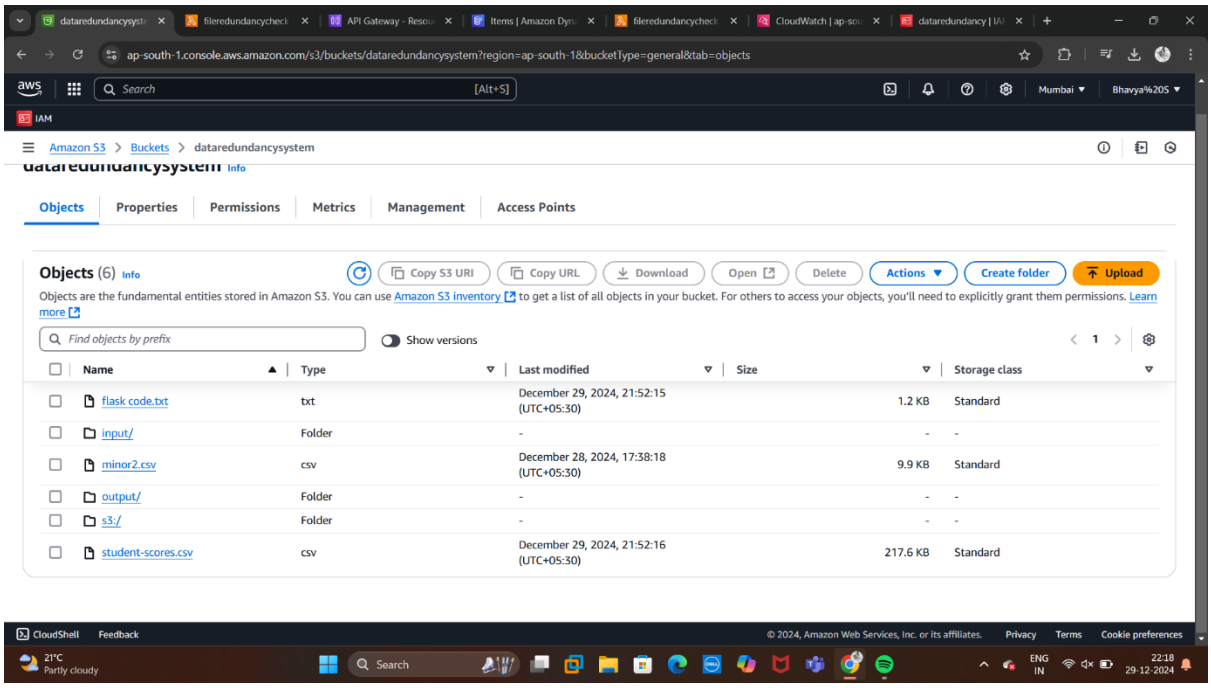
2. AWS Lambda Best Practices *(https://aws.amazon.com/lambda/faqs/)-* AWS offers best practices and guidelines for working with Lambda, which will help you optimize your Lambda functions.

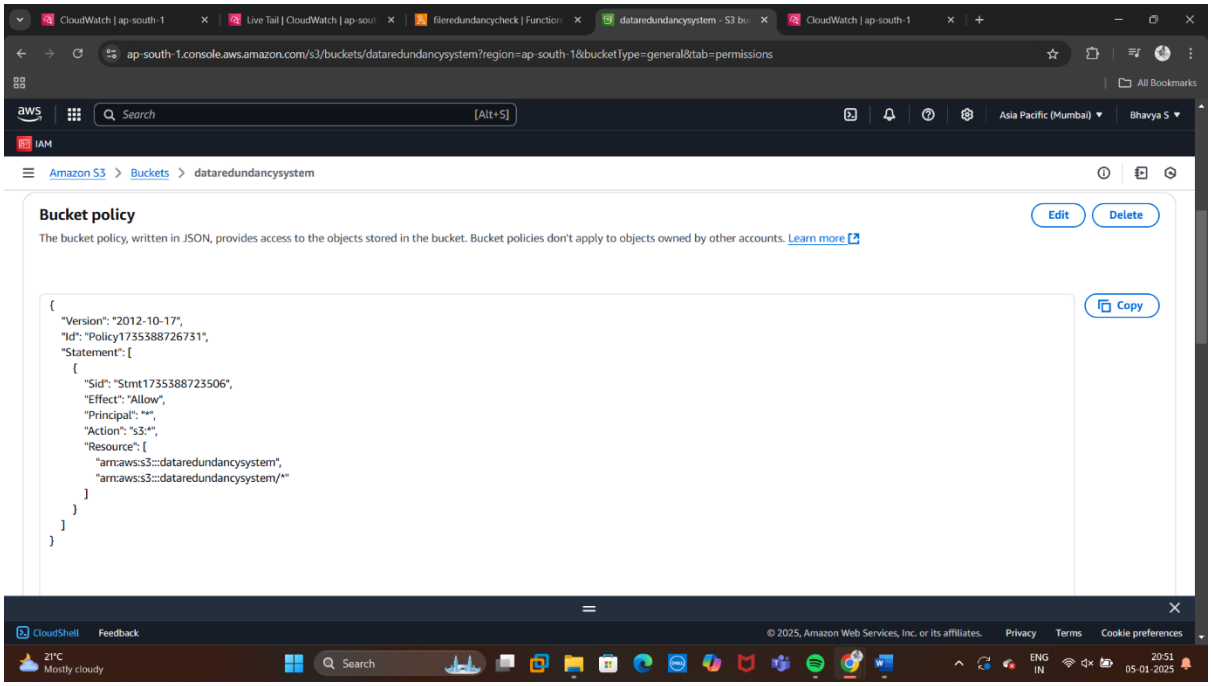
3. Stack Overflow*(https://stackoverflow.com/)-* A popular site for developers where you can find solutions to common coding problems. You can search for issues related to AWS services, Lambda functions, DynamoDB, etc.

9.APPENDIX

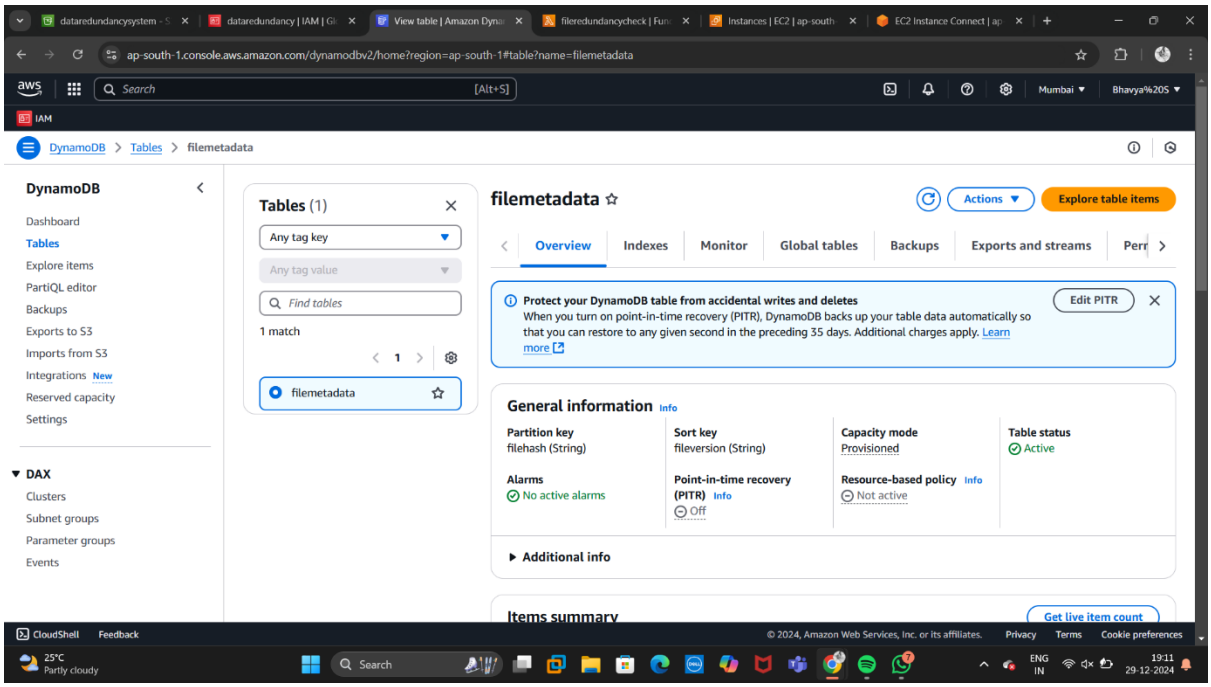
9.1 SNAPSHOTS:

AWS S3 :

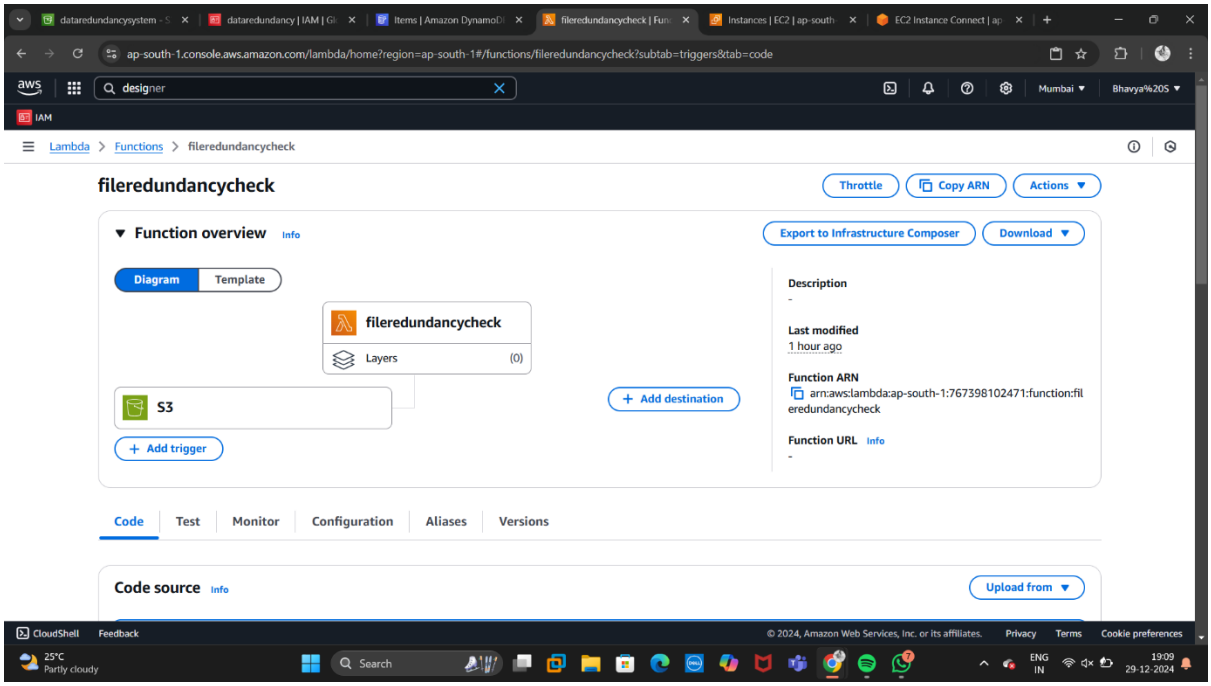




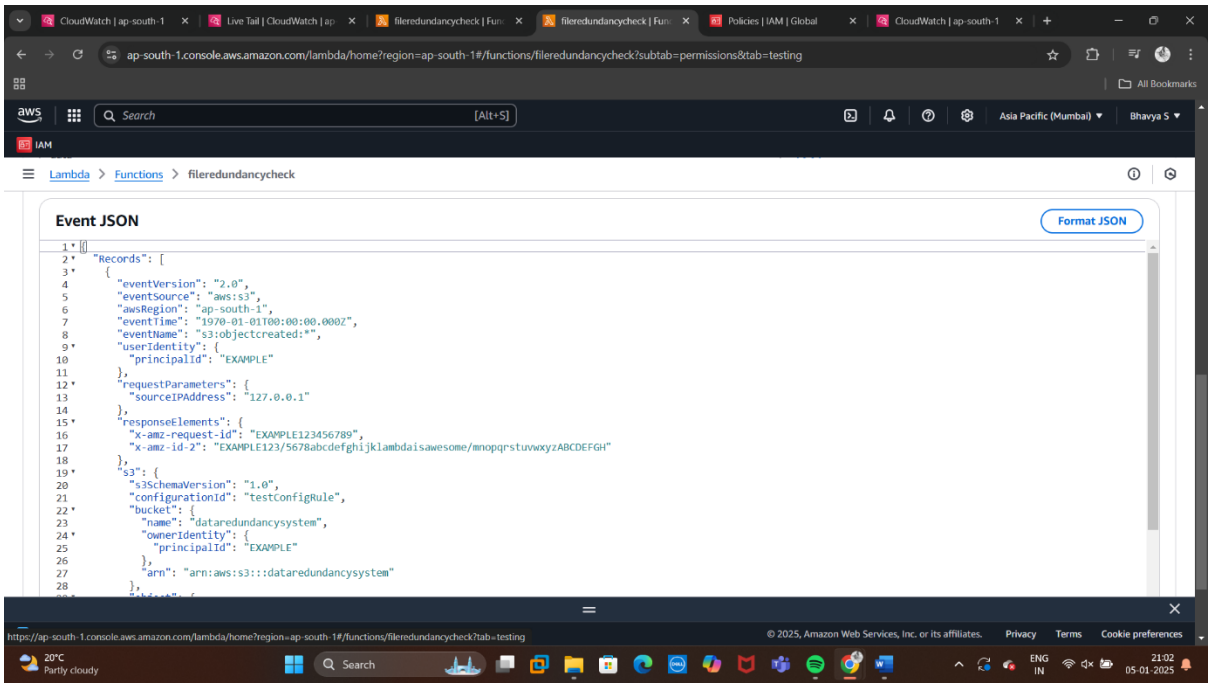
AWS DYNAMO DB :



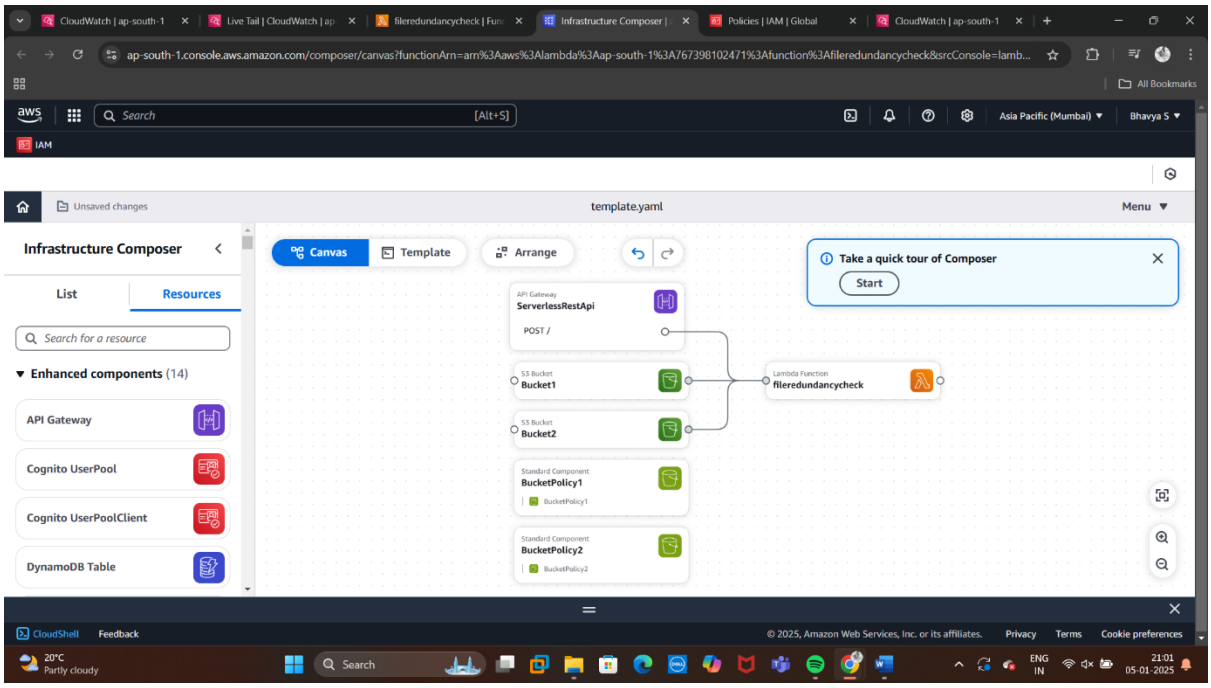
AWS LAMBDA :



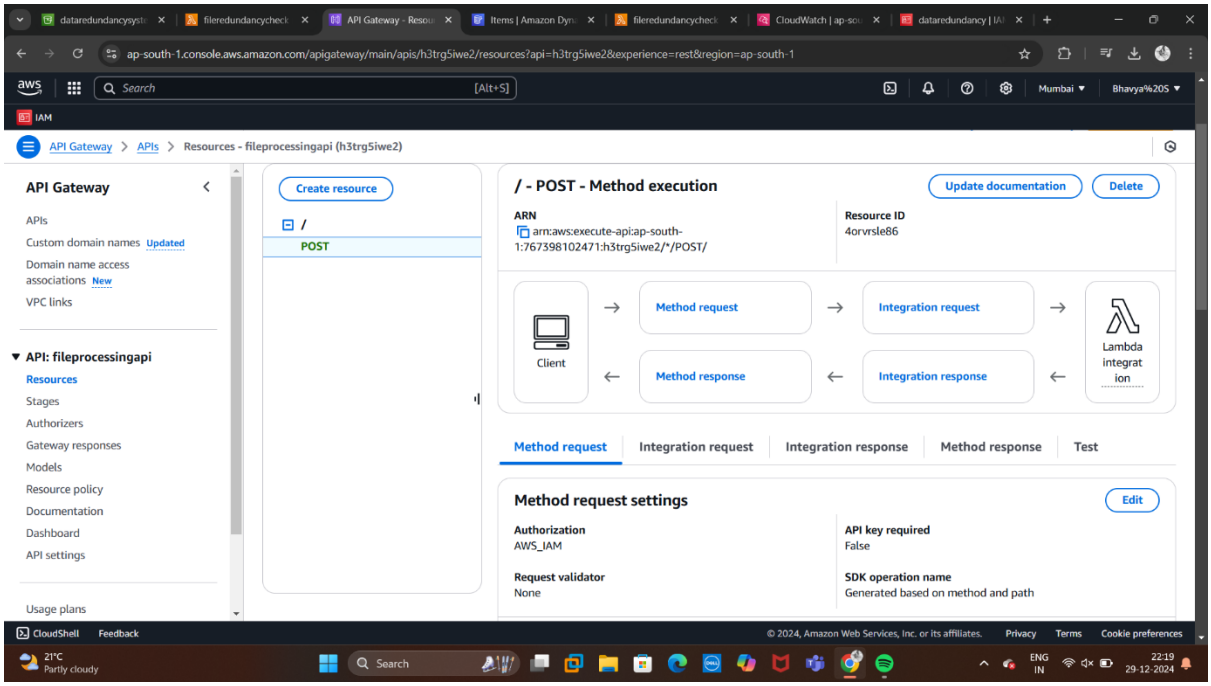
AWS LAMBDA TEST:



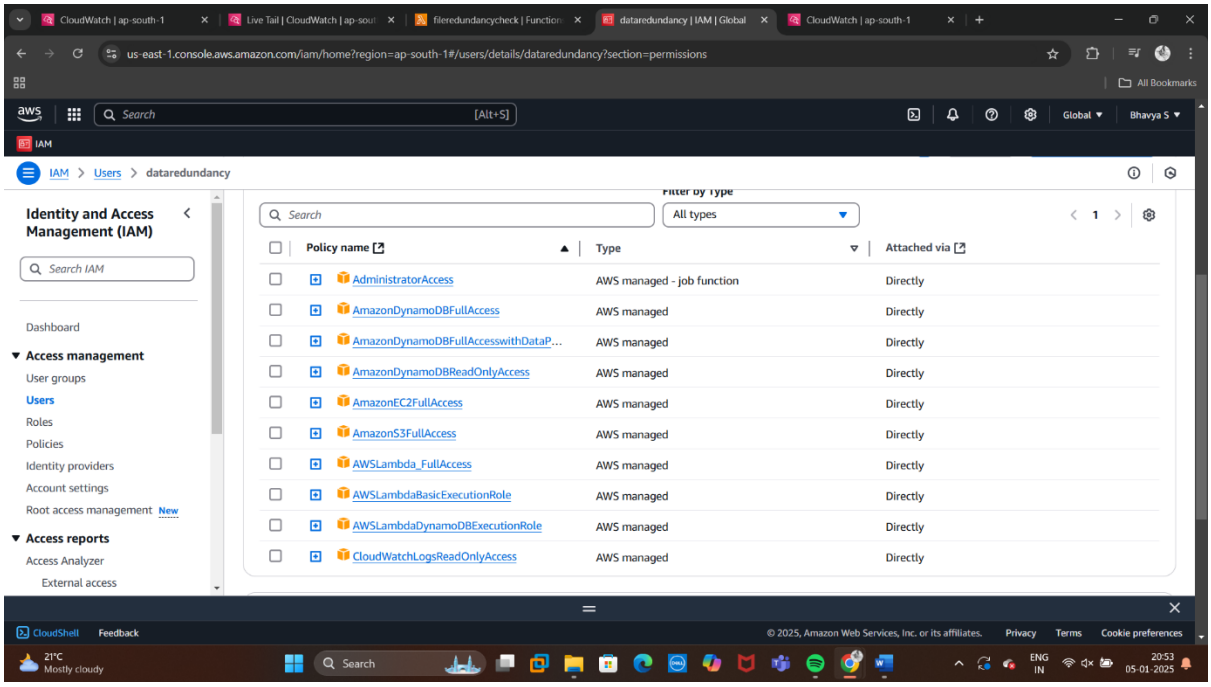
AWS LAMBDA (INFRASTRUCTURE COMPOSER) :



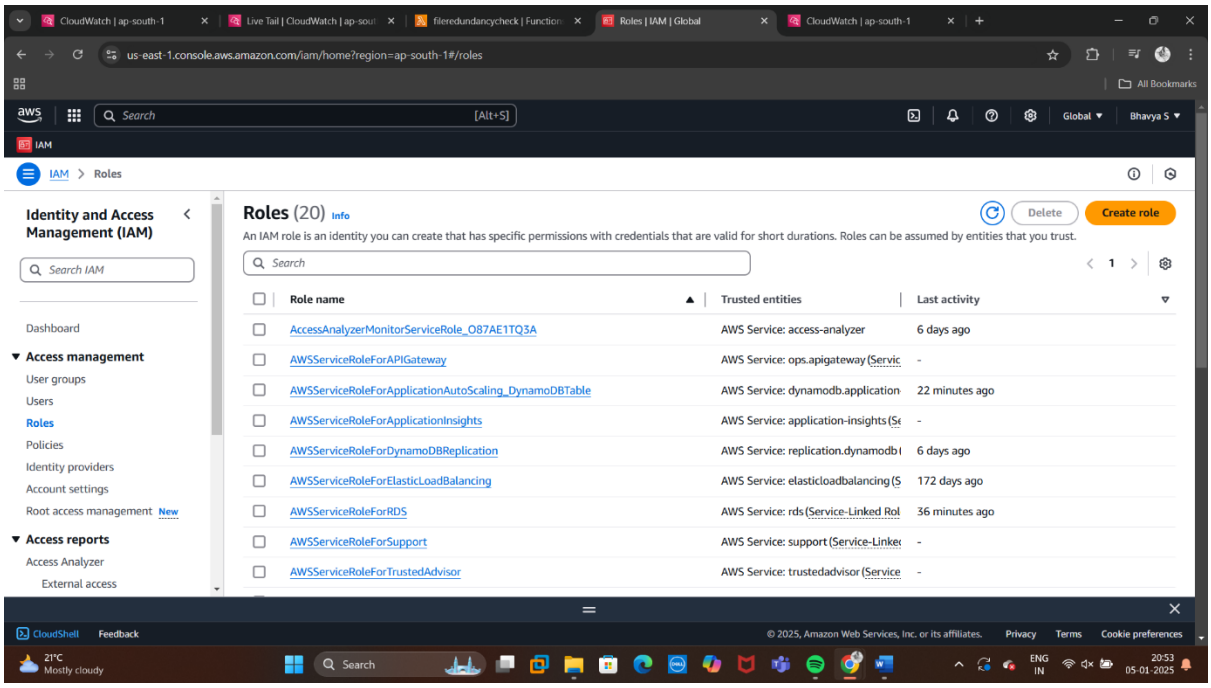
AWS API GATEWAY :



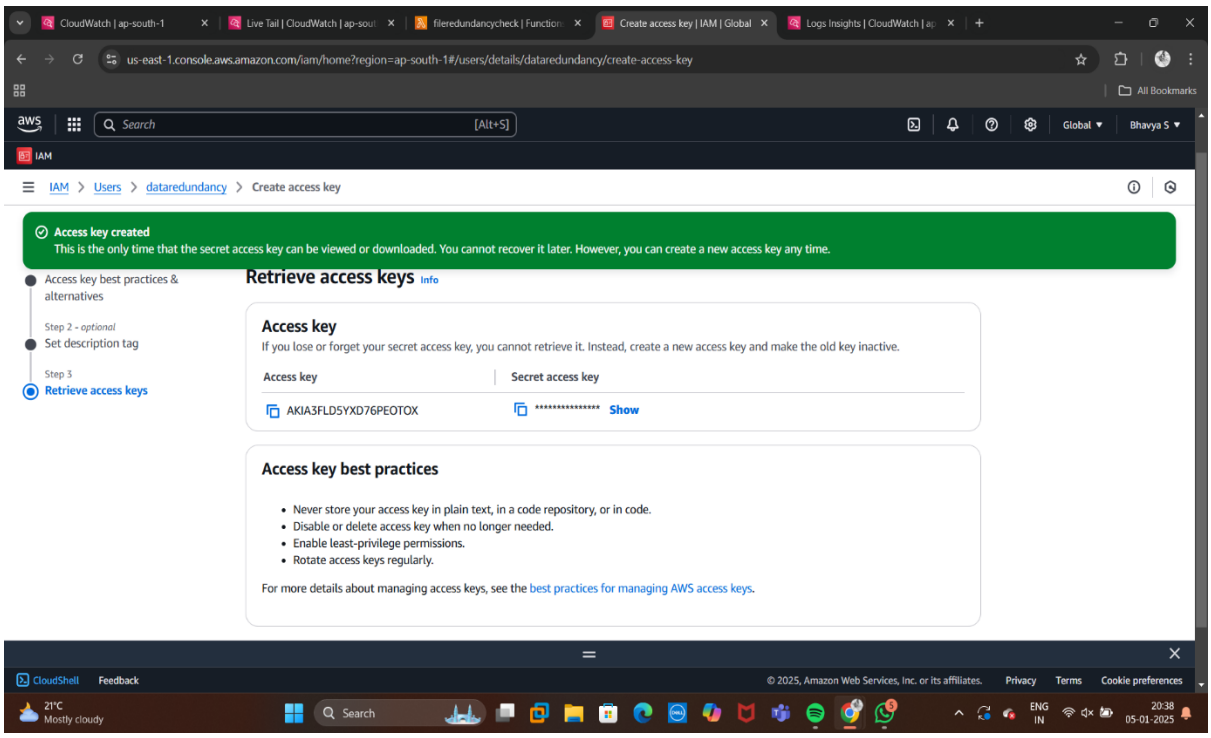
AWS IAM (USERS):



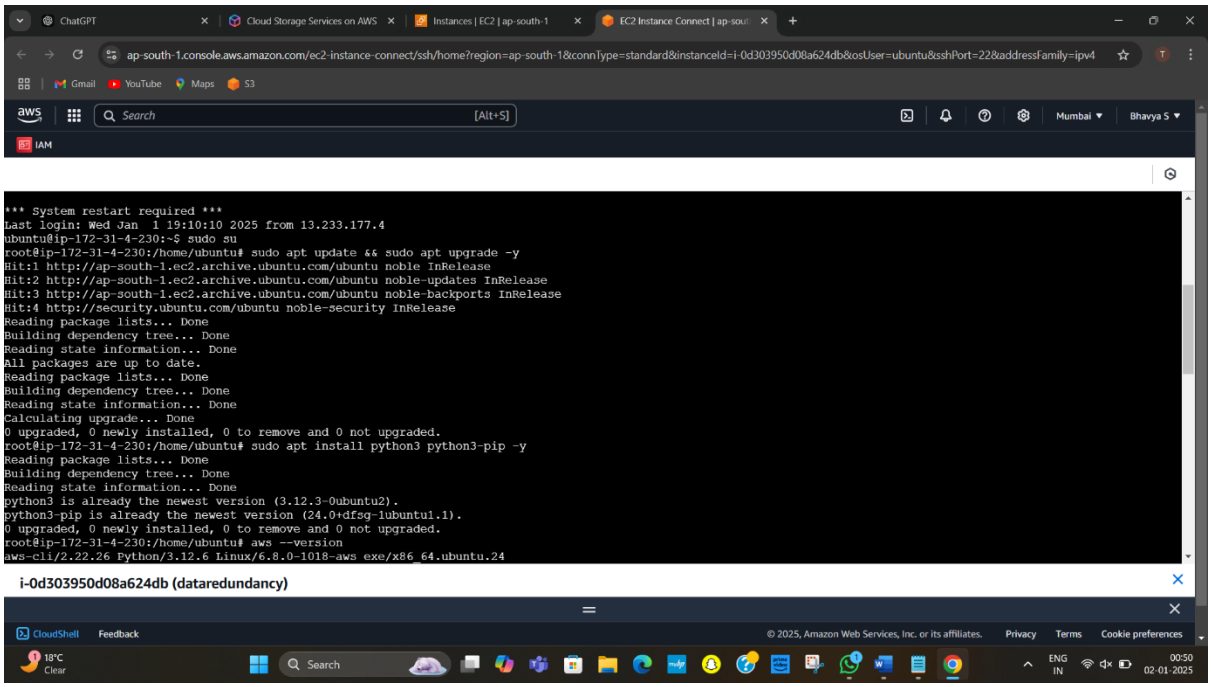
AWS IAM (ROLES) :



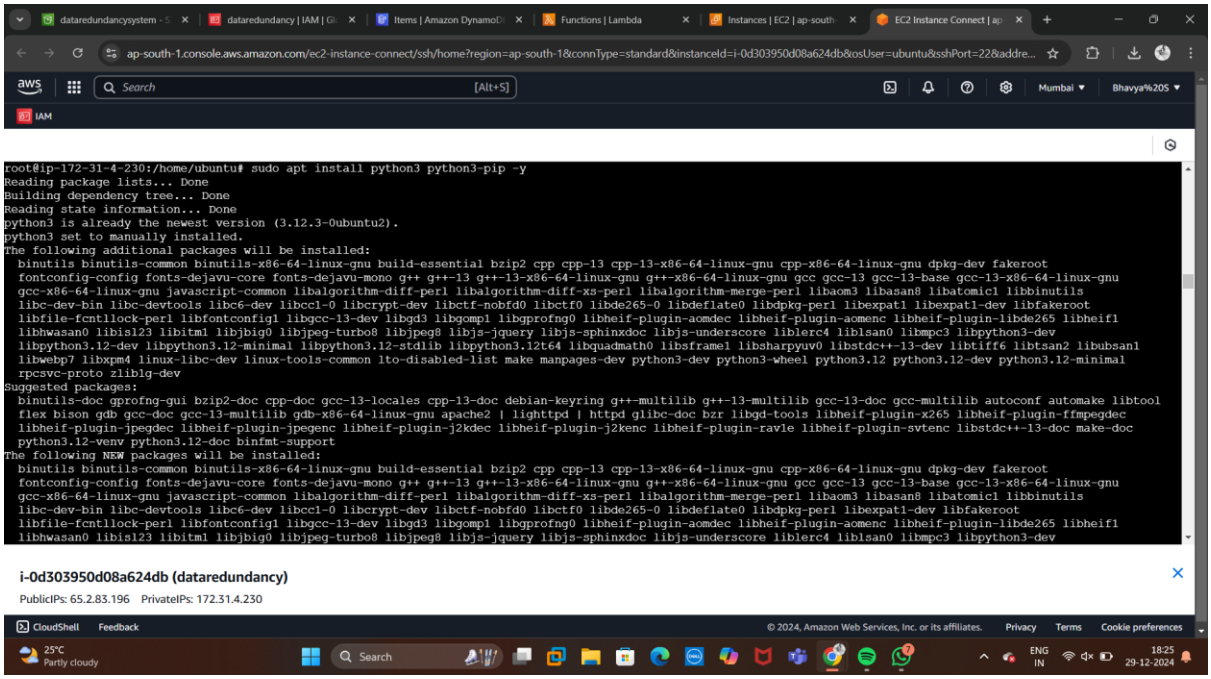
AWS IAM (SECURITY CREDENTIALS) :



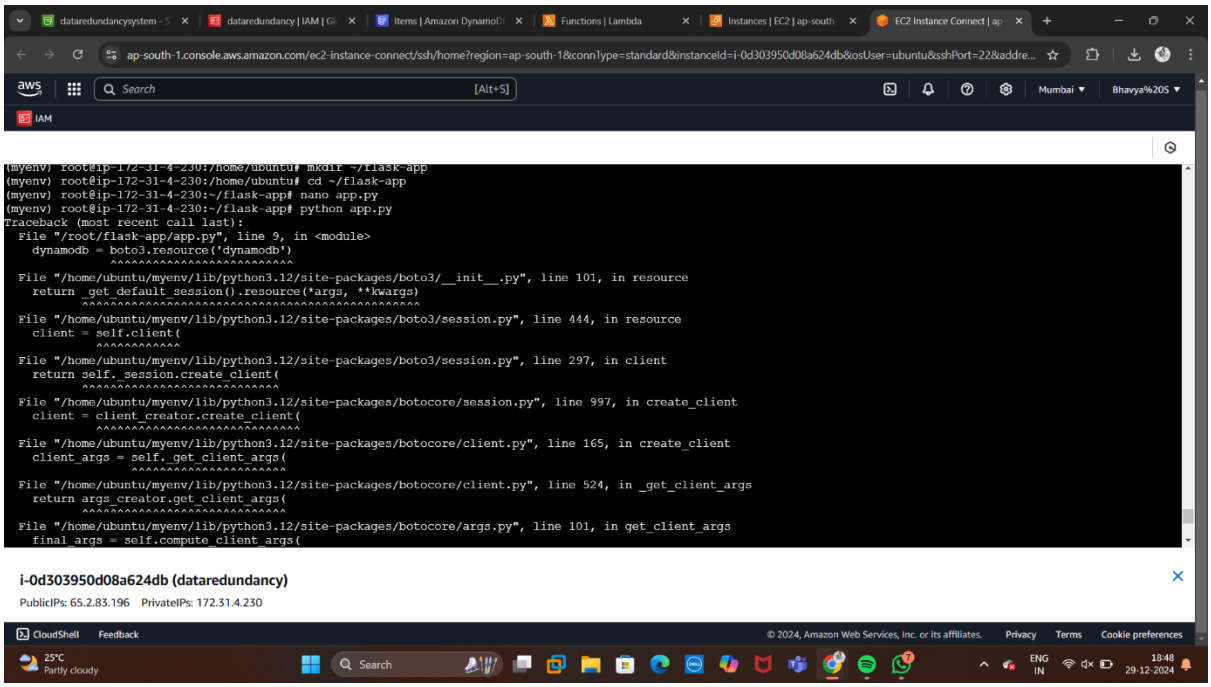
AWS EC2 :



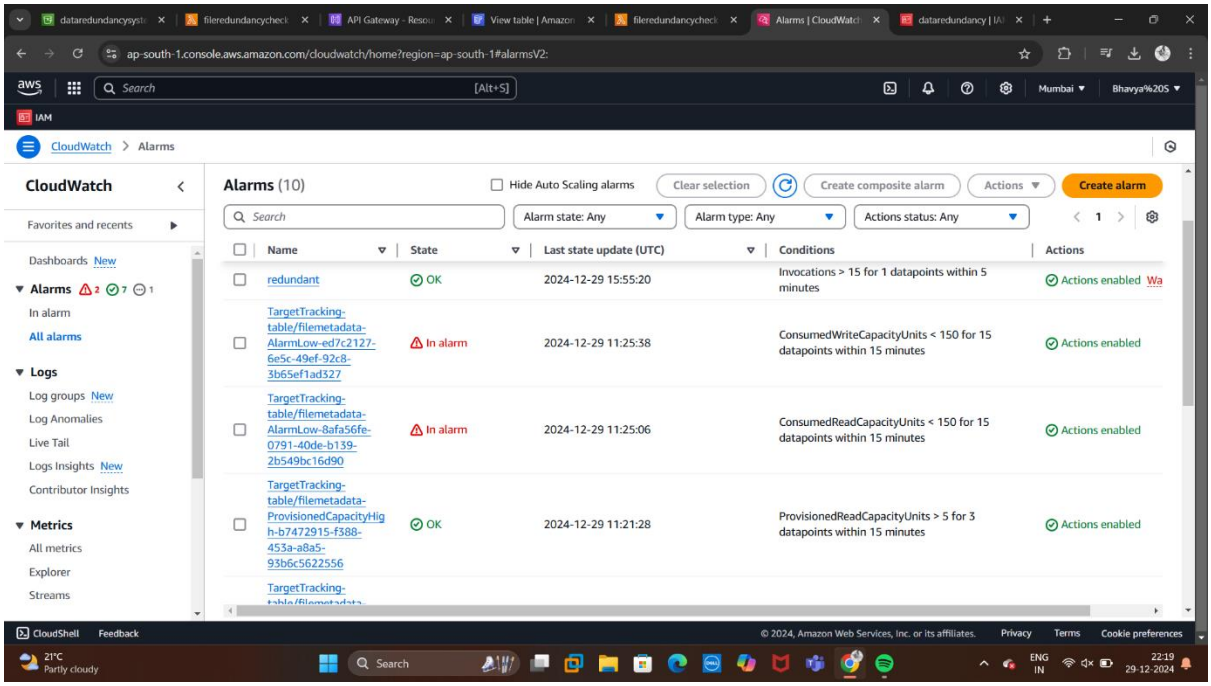
BACKEND DEVELOPMENT :



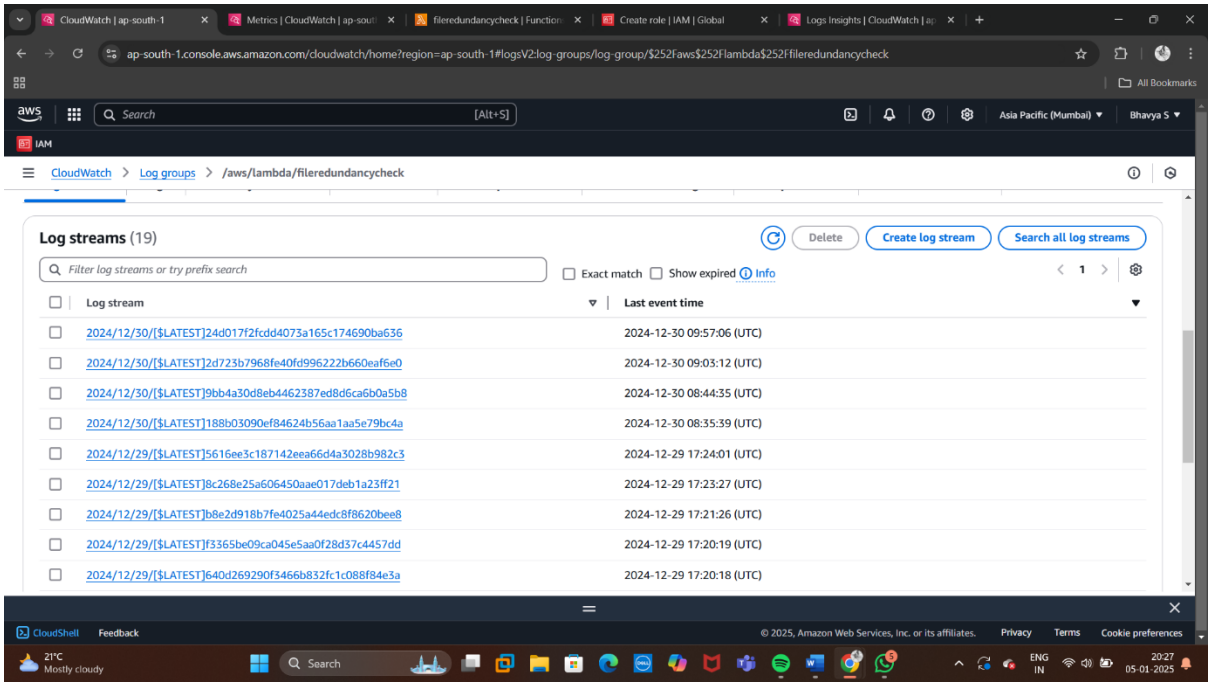
BACKEND DEVELOPMENT CONTINUATION :



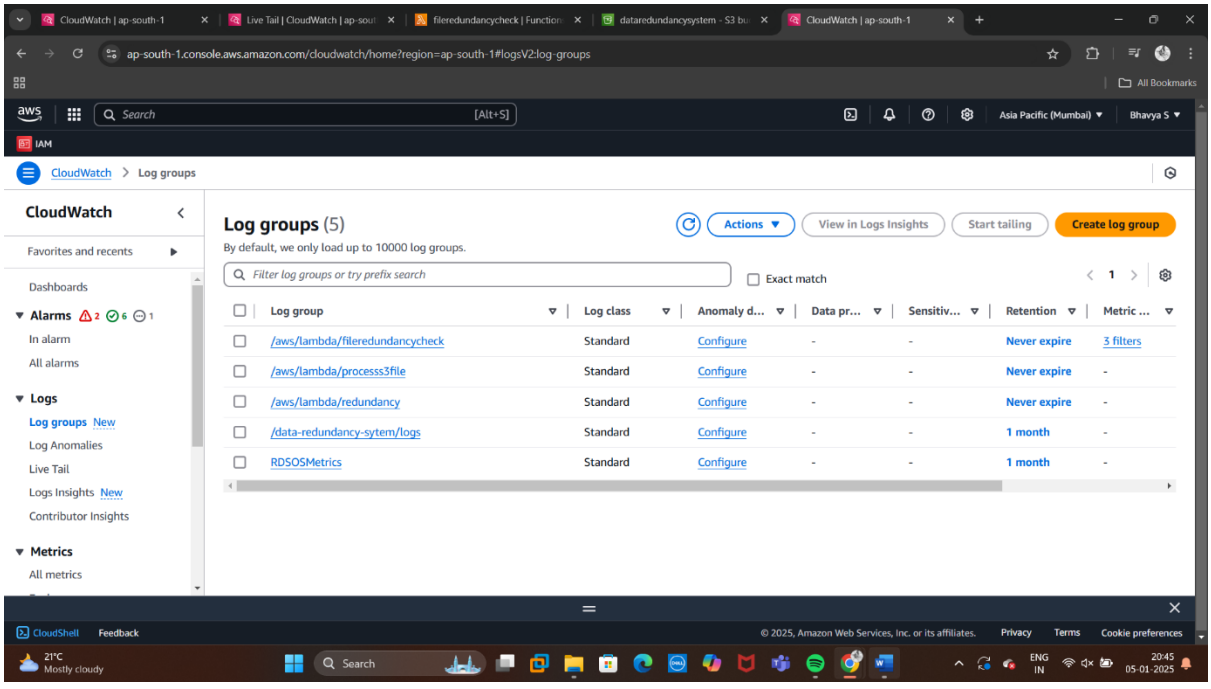
AWS CLOUDWATCH :



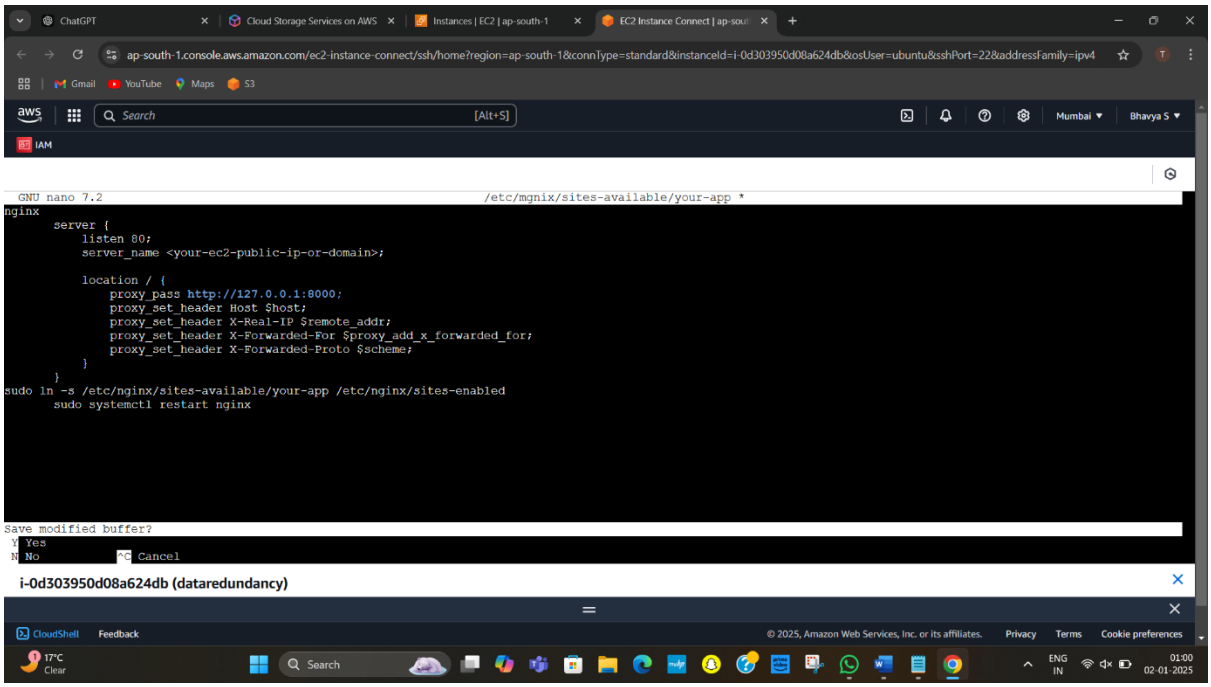
AWS CLOUDWATCH (LOG STREAMS) :



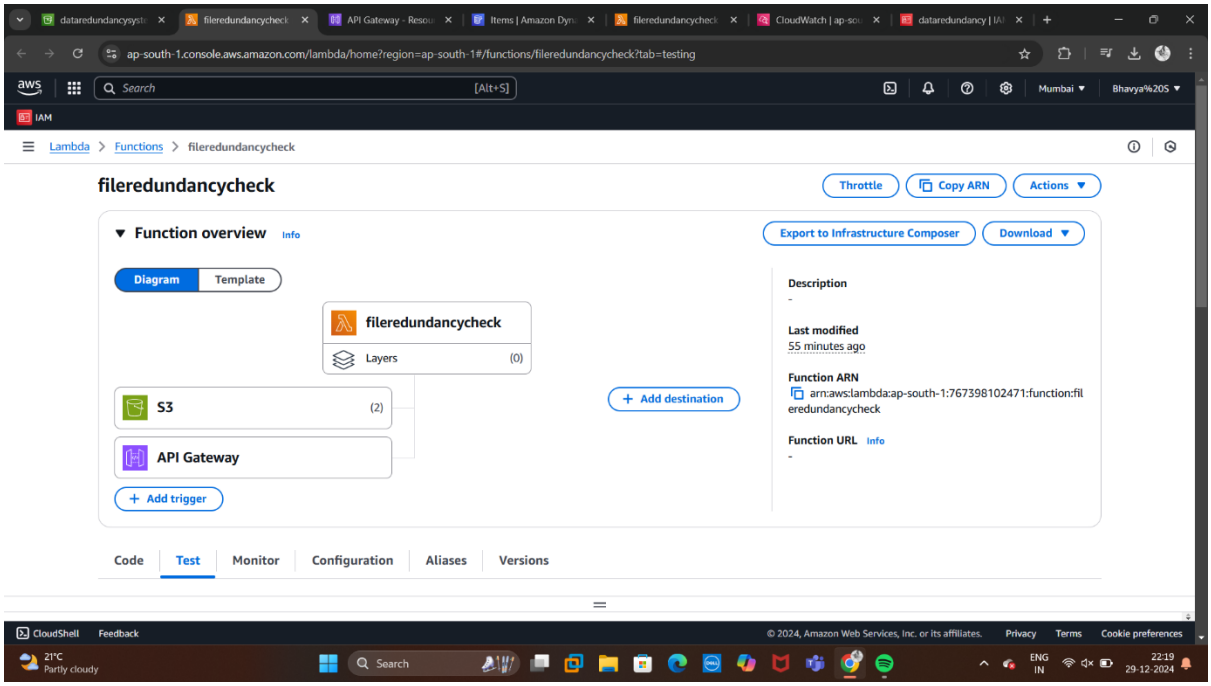
AWS CLOUDWATCH (LOG GROUPS) :



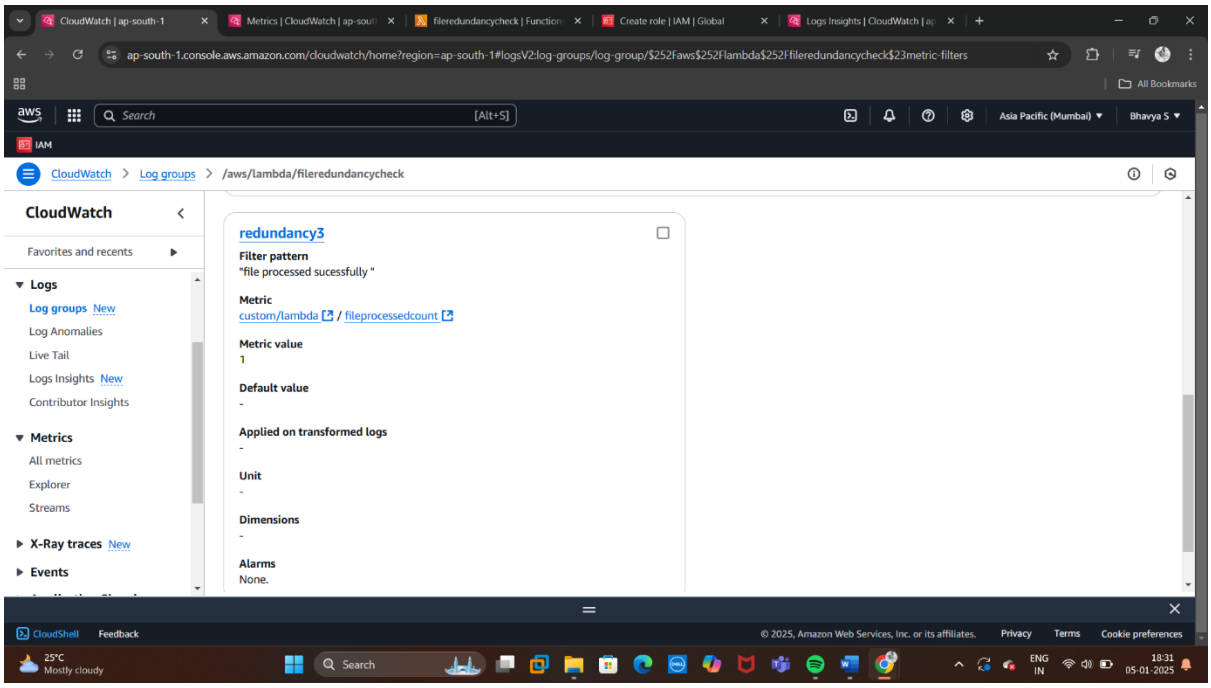
INTEGRATION AND DEPLOYMENT :

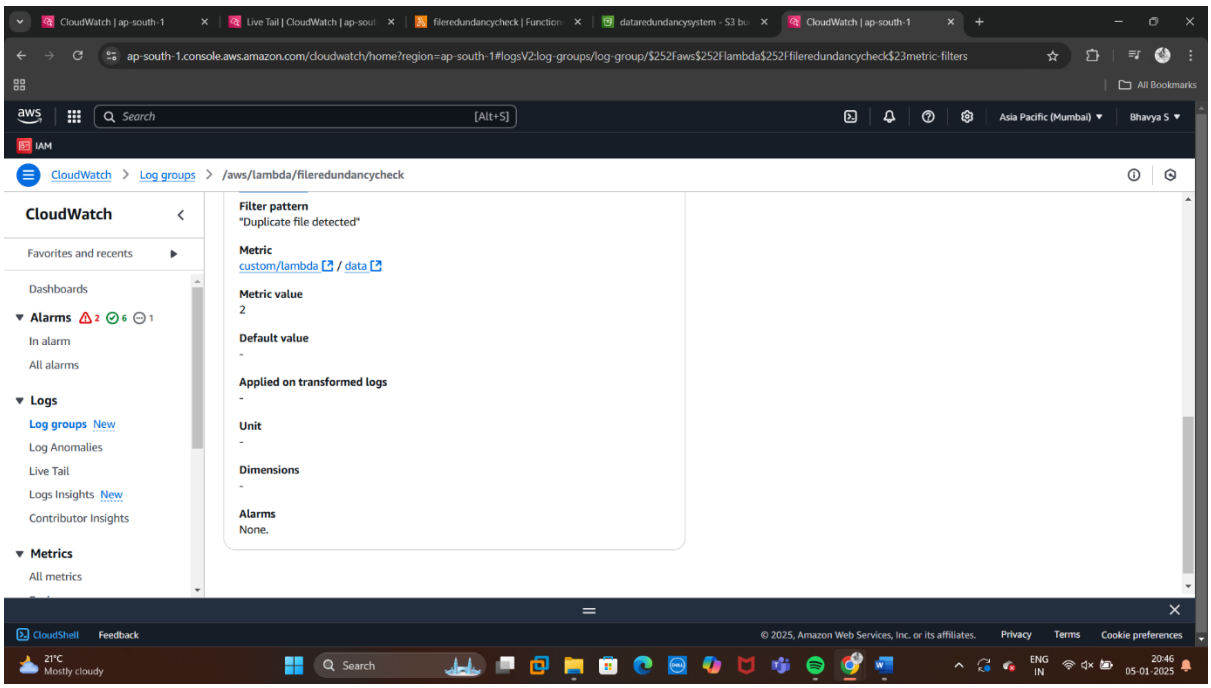


FRONTEND DEVELOPMENT :



RESULT :


CloudWatch | ap-south-1
Metrics | CloudWatch | ap-south-1
fileredundancycheck | Function
Create role | IAM | Global
Logs Insights | CloudWatch | ap-south-1
ap-south-1.console.aws.amazon.com/cloudwatch/home?region=ap-south-1#logsV2:log-groups/log-group/$252Faws$252Flambda$252Ffileredundancycheck$23metric-filters
Search [Alt+S]
Asia Pacific (Mumbai)
Bhavya S
IAM
CloudWatch > Log groups > /aws/lambda/fileredundancycheck
CloudWatch
Favorites and recents
Logs
Log groups New
Log Anomalies
Live Tail
Logs Insights New
Contributor Insights
Metrics
All metrics
Explorer
Streams
X-Ray traces New
Events
redundancy3
Filter pattern
"file processed sucessfully "
Metric
custom/lambda / fileprocessedcount
Metric value
1
Default value
-
Applied on transformed logs
-
Unit
-
Dimensions
-
Alarms
None.
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
25°C
Mostly cloudy
Search
ENG
IN
18:31
05-01-2025


CloudWatch | ap-south-1
Live Tail | CloudWatch | ap-south-1
fileredundancycheck | Function
dataredundancysystem - S3 bu
CloudWatch | ap-south-1
ap-south-1.console.aws.amazon.com/cloudwatch/home?region=ap-south-1#logsV2:log-groups/log-group/$252Faws$252Flambda$252Ffileredundancycheck$23metric-filters
Search [Alt+S]
Asia Pacific (Mumbai)
Bhavya S
IAM
CloudWatch > Log groups > /aws/lambda/fileredundancycheck
CloudWatch
Favorites and recents
Dashboards
Alarms
In alarm
All alarms
Logs
Log groups New
Log Anomalies
Live Tail
Logs Insights New
Contributor Insights
Metrics
All metrics
Filter pattern
"Duplicate file detected"
Metric
custom/lambda / data
Metric value
2
Default value
-
Applied on transformed logs
-
Unit
-
Dimensions
-
Alarms
None.
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
21°C
Mostly cloudy
Search
ENG
IN
20:46
05-01-2025