

# AWS Data Redundancy Removal System Project Document

## Project Overview

The AWS Data Redundancy Removal System was developed to optimize data storage in AWS by identifying and removing redundant data from S3 buckets. This project was designed to reduce storage costs, improve data management, and enhance system efficiency.

## Technologies Used

- **Amazon S3:** For data storage and lifecycle management.
- **S3 Lifecycle Policies:** For automated management of data retention.
- **AWS Lambda:** For automated execution of data analysis and redundancy removal.
- **Amazon CloudWatch:** For monitoring, logging, and triggering alarms.
- **AWS IAM:** For secure access management and permissions control.

## Roles and Responsibilities

- **Cloud Engineer (Your Role):**
  - Configured S3 buckets and implemented lifecycle policies.
  - Developed a Lambda function for automatic data redundancy detection and removal.
  - Set up CloudWatch for monitoring and logging.
  - Defined IAM roles and policies for secure access management.
  - Documented the solution and provided training to team members.

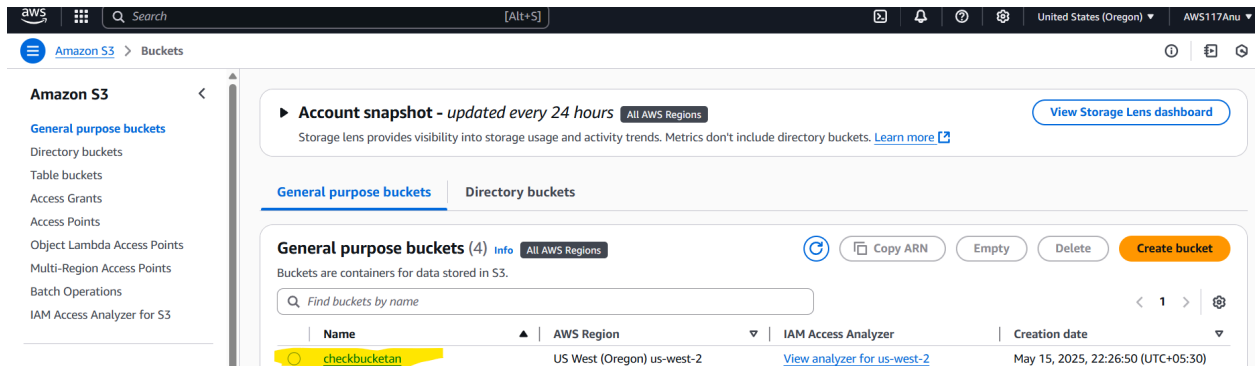
## Project Scope

- Implement a data redundancy detection system for S3 bucket data.
- Automate data lifecycle management using S3 Lifecycle Policies.
- Set up real-time monitoring with CloudWatch.
- Ensure secure access management with IAM policies.
- Optimize the cost of data storage in AWS.

## Implementation Steps

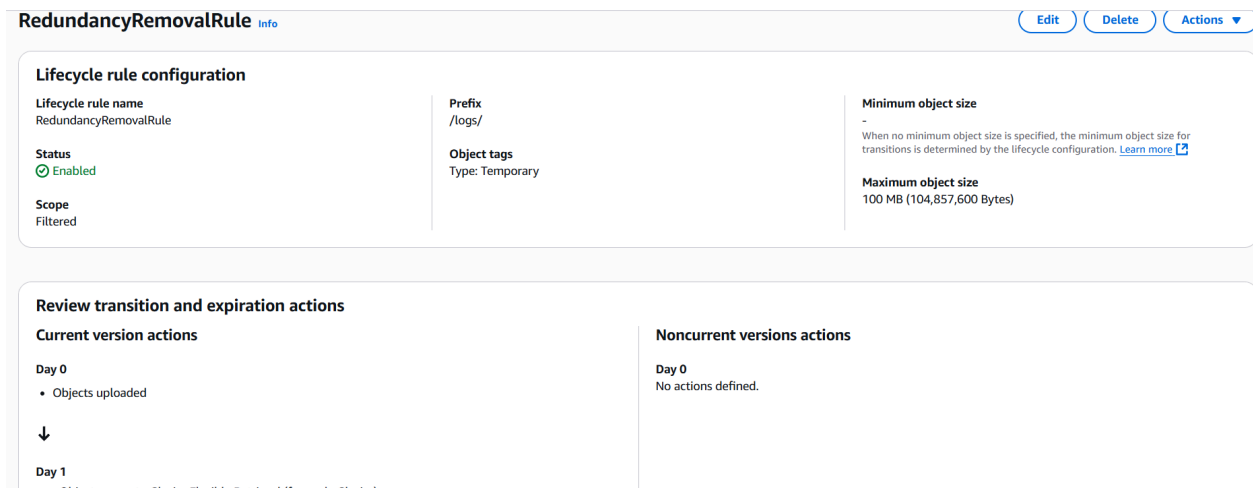
## Step 1: Setting Up S3 Buckets

- Created an S3 bucket with appropriate naming conventions.
- Configured versioning to maintain multiple versions of data.



## Step 2: Defining Lifecycle Policies


- Set up S3 Lifecycle Policies to automatically transition or delete redundant data.



## Step 3: Creating the Lambda Function

- Developed a Lambda function using Python to detect redundant data.
- Integrated Lambda with S3 for automatic triggering.
- Configured IAM roles to grant Lambda the necessary permissions.

## Functions (1)

Last fetched 25 seconds ago 

Actions ▾

Create function

🔍 Filter by attributes or search by keyword

< 1 > 

<input type="checkbox"/>	Function name ▾	Description ▾	Package type ▾	Runtime ▾	Architecture ▾	Code size ▾	Memory (MB) ▾	Timeout (s) ▾	Last modified ▾
<input type="checkbox"/>	<a href="#">lambdaRedundRemFunc2</a>	-	Zip	Python 3.9	x86_64	736 byte	128	3	8 hours ago

## Lambda Function

```
import boto3
```

```
import hashlib
```

```
from datetime import datetime
```

```
s3_client = boto3.client('s3')
```

```
def lambda_handler(event, context):
```

```
    bucket_name = 'checkbucketan' # Replace with your bucket name
```

```
    # List all objects in the bucket
```

```
    response = s3_client.list_objects_v2(Bucket=bucket_name)
```

```
    if 'Contents' not in response:
```

```
        return {'status': 'No files found'}
```

```
    # Dictionary to store hash and file metadata
```

```
    file_hashes = {}
```

```
    for obj in response['Contents']:
```

```
        key = obj['Key']
```

```
        # Get object content to compute MD5 hash
```

```

obj_content = s3_client.get_object(Bucket=bucket_name,
Key=key) ['Body'].read()

md5_hash = hashlib.md5(obj_content).hexdigest()

if md5_hash in file_hashes:

    # Compare last modified time to keep the oldest file

    existing_file = file_hashes[md5_hash]

    new_file_time = obj['LastModified']

    if new_file_time > existing_file['LastModified']:

        # Delete newer duplicate

        s3_client.delete_object(Bucket=bucket_name, Key=key)

        print(f"Deleted duplicate: {key}")

    else:

        # Delete older file and update hash entry

        s3_client.delete_object(Bucket=bucket_name,
Key=existing_file['Key'])

        file_hashes[md5_hash] = {'Key': key, 'LastModified':
new_file_time}

        print(f"Deleted duplicate: {existing_file['Key']}")

    else:

        file_hashes[md5_hash] = {'Key': key, 'LastModified':
obj['LastModified']}

return {'status': 'Deduplication complete'}

```

## Step 4: Configuring CloudWatch

- Set up CloudWatch for monitoring Lambda function executions.
- Enabled CloudWatch Logs for debugging and tracking Lambda performance.

/aws/lambda/lambdaRedundRemFunc2

ActionsView in Logs InsightsStart tailingSearch log group

▼ Log group details

Log class  
Standard

ARN  
arn:aws:logs:us-west-2:207567803021:log-group:/aws/lambda/lambdaRedundRemFunc2:\*

Creation time  
15 hours ago

Retention  
Never expire

Stored bytes  
-

Metric filters  
0

Subscription filters  
0

Contributor Insights rules  
-

KMS key ID  
-

Anomaly detection  
Configure

Data protection  
-

Sensitive data count  
-

Field indexes  
Configure

Transformer  
Configure

Log streams (8)

Filter log streams or try prefix search

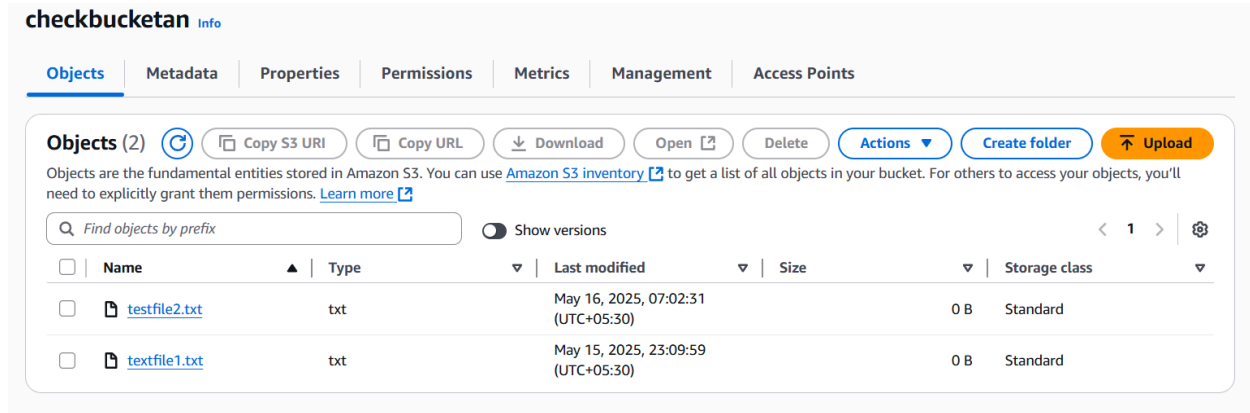
Exact matchShow expiredInfo

Log stream	Last event time
2025/05/15/[\$LATEST]0391cd9487ba4994ae32733da05075bc	2025-05-15 23:10:16 (UTC+05:30)
2025/05/15/[\$LATEST]eec8f5b121204cf7a53b40ea95bfdd8d	2025-05-15 22:50:41 (UTC+05:30)
2025/05/15/[\$LATEST]e7e7289083e64f2882a5e1e0bc918bbe	2025-05-15 22:32:09 (UTC+05:30)
2025/05/15/[\$LATEST]5b21854506b347b78364e872b81a156c	2025-05-15 22:27:54 (UTC+05:30)
2025/05/15/[\$LATEST]bb880a58e0a2494ba57f7a4032109baa	2025-05-15 22:02:33 (UTC+05:30)
2025/05/15/[\$LATEST]ad26ab2234204d4682aa0f3f26a90e00	2025-05-15 20:34:14 (UTC+05:30)
2025/05/15/[\$LATEST]a4b50a1ec8e44d42a924dc6be8d02992	2025-05-15 20:23:00 (UTC+05:30)
2025/05/15/[\$LATEST]afb8e26e8f0e4657a9f7abc5d7811ea8	2025-05-15 16:31:33 (UTC+05:30)

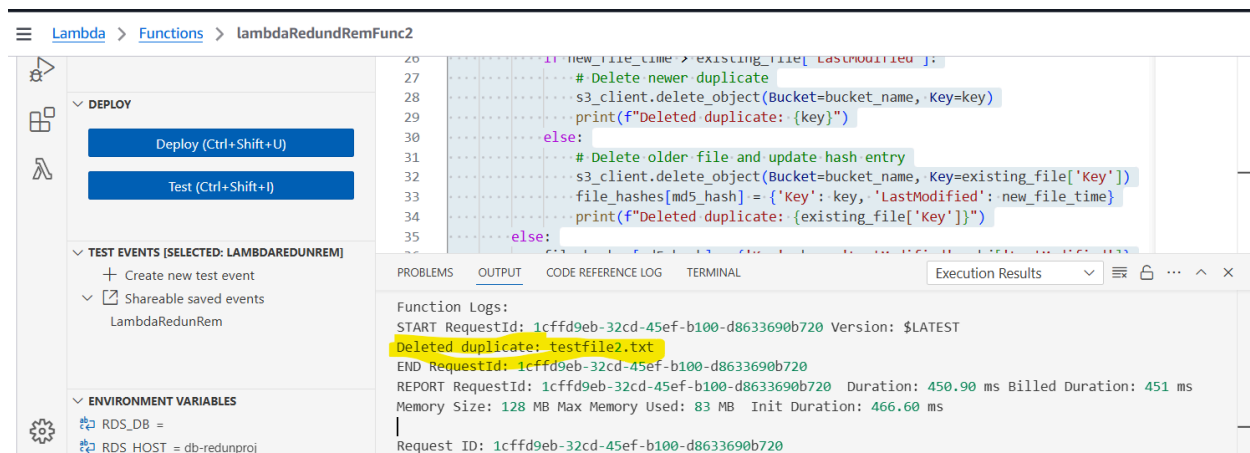
## Step 5: Testing and Validation

- Uploaded test data to S3 to validate redundancy detection.
- Verified that redundant data was automatically identified and removed.

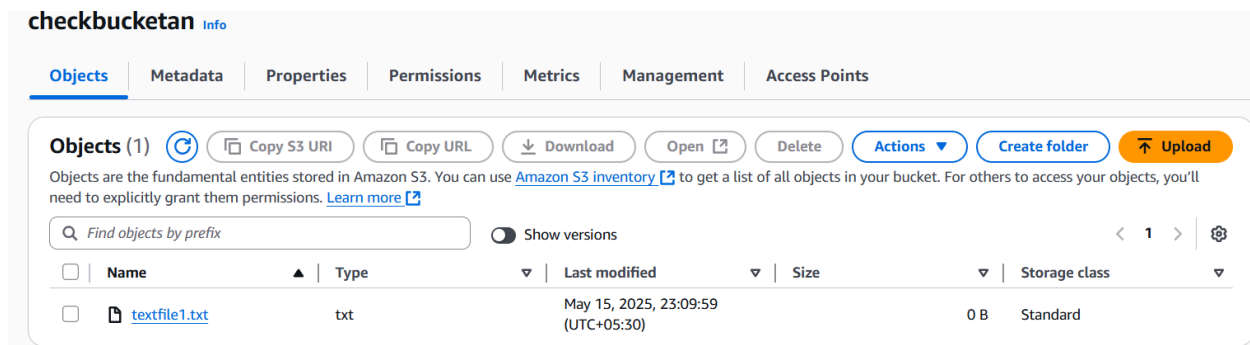
I uploaded two differently named files, but the content is the same.



I have tested the lambda function as shown in the picture.



Here we can see that testfile2.txt is removed.



## Step 6: Security Management with IAM

- Created IAM roles and policies with the least privilege principle.
- Ensured secure access for Lambda and S3.

Identity and Access Management (IAM)

Search IAM

Dashboard

Access management

Users

Roles

Policies

Identity providers

Account settings

Root access management

Access reports

Permissions

Trust relationships

Tags

Last Accessed

Revoke sessions

Permissions policies (4)

Info

Simulate

Remove

Add permissions

You can attach up to 10 managed policies.

Search

Filter by Type

All types

Policy name

Type

Attached entities

AmazonRDSElasticAccess

AWS managed

2

AmazonS3FullAccess

AWS managed

1

AmazonS3ReadOnlyAccess

AWS managed

2

CloudWatchLogsFullAccess

AWS managed

3

## Step 7: Monitoring and Optimization

- Set up CloudWatch Alarms for error notifications.
- Monitored storage costs to verify cost savings.

Log streams

Tags

Anomaly detection

Metric filters

Subscription filters

Contributor Insights

Data protection

Field

Log streams (9)

Delete

Create log stream

Search all log streams

Filter log streams or try prefix search

Exact match

Show expired

Info

Log stream

Last event time

2025/05/16/[LATEST]2dea824fa1244fad8bbe1946ae8fb663

2025-05-16 07:05:23 (UTC+05:30)

2025/05/15/[LATEST]0391cd9487ba4994ae32733da05075bc

2025-05-15 23:10:16 (UTC+05:30)

2025/05/15/[LATEST]eec8f5b121204cf7a53b40ea95bfdd8d

2025-05-15 22:50:41 (UTC+05:30)

2025/05/15/[LATEST]e7e7289083e64f2882a5e1e0bc918bbe

2025-05-15 22:32:09 (UTC+05:30)

2025/05/15/[LATEST]5b21854506b347b78364e872b81a156c

2025-05-15 22:27:54 (UTC+05:30)

2025/05/15/[LATEST]bb880a58e0a2494ba57f7a4032109baa

2025-05-15 22:02:33 (UTC+05:30)

2025/05/15/[LATEST]ad26ab2234204d4682aa0f3f26a90e00

2025-05-15 20:34:14 (UTC+05:30)

2025/05/15/[LATEST]a4b50a1ec8e44d42a924dc6be8d02992

2025-05-15 20:23:00 (UTC+05:30)

2025/05/15/[LATEST]afb8e26e8f0e4657a9f7abc5d7811ea8

2025-05-15 16:31:33 (UTC+05:30)

## Problem Solved

The project addressed the issue of excessive storage costs due to redundant data in S3. By implementing an automated redundancy detection and removal process, the project reduced unnecessary data storage and improved cost efficiency.

## Benefits to the Organization

- **Reduction in Storage Costs:** Minimized expenses by eliminating redundant data.
- **Increased Profitability:** Optimized storage usage without affecting data integrity.

- **Improved Efficiency:** Automated the process of identifying and deleting redundant data.

## Team Size

- The project was completed by Anulata Priyadarshini.

## Conclusion

The AWS Data Redundancy Removal System effectively optimized data storage in AWS S3, leading to reduced costs and enhanced data management capabilities for the organization.