

“Automating File Backup and Ensuring Data Redundancy Using AWS S3 and Replication”

Abstract

In an increasingly digital world, data security and availability are paramount for individuals and organizations alike. Data loss due to hardware failure, accidental deletion, or cyber threats can result in significant setbacks, making reliable backup solutions essential. This project addresses the need for an automated and secure data backup system that integrates local machines with cloud storage. Specifically, it focuses on automating the backup of files from a local machine to Amazon Web Services (AWS) S3 and implementing Same-Region Replication (SRR) for added redundancy within the same AWS region.

The system continuously monitors selected directories on the local machine, automatically uploading new or modified files to a designated S3 bucket in real time. To further ensure data resilience, the system utilizes AWS's SRR feature to replicate the files to a secondary S3 bucket within the same region. This approach provides an extra layer of data protection, enabling quick recovery in case of accidental data loss or corruption in the primary backup.

By leveraging AWS services such as S3, IAM roles, and S3 Replication Rules, the solution ensures secure, efficient, and scalable file backups. The automation of this process reduces the risk of human error, streamlines backup operations, and ensures data integrity. The system is designed to be scalable, allowing users to expand their backup storage as their data grows. With a focus on simplicity, security, and reliability, this solution aims to provide users with a seamless, cost-effective way to safeguard their critical data.

INDEX

Chapter No.	Title	Page No.
	Abstract	I
	Index	ii
1	Introduction	1
	1.1 Introduction/Background	1
	1.2 Motivation	3
	1.3 Objective	3
	1.4 Problem Statement	4
	1.5 Approach	4
2	Technologies	6
3	Methodology	7
4	Result and Discussion	10
5	Conclusions and Future Scope	15
	References (as per the prescribed format)	17

Chapter 1

Introduction

1.1 Introduction

The volume of digital information generated daily—ranging from high-definition video streams to large-scale scientific datasets and mission-critical business records—has grown at an unprecedented rate, placing immense pressure on traditional backup infrastructures [1]. Conventional solutions, such as on-premises tape libraries or disk-based appliances, often involve substantial capital expenditure, manual handling, and inflexible scaling. Moreover, these systems typically lack geographic redundancy, making them vulnerable to localized failures, natural disasters, or ransomware attacks that can render both primary and backup copies inaccessible.

Cloud object storage technology, offered by major providers such as Amazon Web Services (AWS), Google Cloud Platform, and Microsoft Azure, has revolutionized data protection by providing virtually unlimited, durable, and globally accessible repositories [2]. AWS S3, for example, guarantees 99.999999999% (“eleven nines”) of data durability and offers features like server-side encryption, object lifecycle policies, and cross-region replication [3]. These capabilities make cloud storage an attractive alternative for backup use cases, enabling pay-as-you-go pricing and reducing operational overhead.

However, effectively leveraging cloud storage for automated backups requires careful orchestration across multiple dimensions:

- **Automation and Scheduling:** Manual backup processes are error-prone and introduce latency in data protection. Automating transfers from local environments to the cloud ensures that backups occur at consistent intervals without human intervention [4].
- **Cost Management:** Features such as cross-region replication and premium storage classes (e.g., S3 Glacier) incur additional charges. A cost-aware design must balance durability and accessibility with free-tier or low-cost options.
- **Security and Compliance:** Credentials must be handled securely, with least-privilege IAM roles and encrypted data in transit and at rest. Audit trails and monitoring further strengthen compliance with regulatory standards [5].

- **Incremental Synchronization:** Transferring only new or modified data reduces bandwidth consumption and speeds up the backup process.

This project delivers a comprehensive, end-to-end solution within the AWS Free Tier, integrating two primary components:

1. **Local Backup Agent**

A Python script using the Boto3 SDK to scan specified directories on the user's machine, upload discovered files to an S3 source bucket under timestamped prefixes, and log all operations. Users can schedule this agent on their local system using built-in tools—Windows Task Scheduler for Microsoft environments or cron jobs on Linux/macOS—ensuring seamless, periodic uploads without requiring a dedicated server [6].

2. **Serverless Replication Pipeline**

An AWS Lambda function, triggered by an EventBridge (CloudWatch Events) rule, that enumerates objects in the source bucket, compares each object's `LastModified` timestamp to the existing counterpart in a destination bucket, and copies only new or updated files. By configuring both buckets in the same AWS region, the design emulates cross-region replication semantics—such as maintaining two independent copies—while avoiding inter-region data transfer fees inherent to AWS Free Tier constraints [2].

By combining local scheduling for the initial backup with a serverless replication mechanism, this architecture achieves a cost-effective, scalable, and resilient data protection strategy. Comprehensive logging via AWS CloudWatch Logs and local script outputs provides observability for both upload and replication workflows. Security best practices—environment-variable-driven credentials, least-privilege IAM policies, and optional server-side encryption—are embedded throughout the solution [5]

This work demonstrates how organizations can build a fully automated backup pipeline that balances durability, cost, and operational simplicity, providing a practical blueprint for low-cost, high-reliability data protection.

1.2 Motivation

The increasing volume and criticality of digital assets—ranging from business databases and code repositories to personal multimedia—demands a backup strategy that minimizes human error and maximizes reliability. Traditional manual backup workflows (e.g., copying files by hand or running unscheduled scripts) introduce several risks: missed backup windows, inconsistent capture of new or modified files, and reliance on human diligence for scheduling and verification. Such lapses can lead to data loss, compliance violations, and extended recovery times.

To eliminate these vulnerabilities, this project embeds local automation at the source: a Python agent configured with a time-based schedule (cron on Unix/macOS for demonstration) automatically scans designated directories and uploads only new or changed files to an AWS S3 “source” bucket. In production environments, the same agent can be orchestrated via Windows Task Scheduler to ensure consistent, unattended backups.

Once files reside in the source bucket, an AWS Lambda function—invoked on a regular EventBridge schedule—performs intra-region replication, copying only newly uploaded or modified objects to a second “destination” bucket. This unified, serverless pipeline removes manual intervention entirely, guarantees two independent copies of each file, and remains within AWS Free Tier limits by avoiding cross-region transfer fees.

1.3 Objective

The goal of this work is to develop an end-to-end backup pipeline that operates without manual intervention, remains within AWS Free Tier limits, and ensures data redundancy through intra-region replication. This involves automating local file uploads, implementing serverless synchronization between S3 buckets, and adhering to security best practices.

Key objectives include:

- Design and implement a Python-based agent that, on a time-based schedule (python time library for demonstration; Windows Task Scheduler in production), detects and uploads new or modified local files to an S3 source bucket.
- Develop an AWS Lambda function, triggered by EventBridge, to replicate only new or updated objects from the source bucket to a destination bucket in the same region.
- Maintain cost efficiency by using same-region buckets, limiting Lambda invocation frequency, and leveraging Free Tier allowances.
- Enforce secure credential management via environment variables or IAM roles, and apply least-privilege IAM policies to all components.
- Document deployment, scheduling, monitoring, and logging procedures for both local and cloud environments.

1.4 Problem Statement

Relying on manual or ad-hoc scripts to back up local data introduces unacceptable risk: scheduled transfers can be missed, new or modified files overlooked, and human error may leave critical information unprotected until it is too late. Although Amazon S3 provides built-in replication across Availability Zones to ensure high durability and fault tolerance, it does not enforce user-defined backup policies or retention rules. Without a local automation layer, simply uploading files to S3 does not guarantee that backups occur consistently or that obsolete snapshots are pruned, leading to unchecked storage growth and potential cost overruns.

This project addresses these shortcomings by implementing a Python-based agent that automates local-to-cloud backups on a fixed schedule, incorporates configurable retention logic to delete snapshots older than a specified threshold, and leverages and implements S3's internal replication for fault tolerance. All operations use environment-driven credentials and least-privilege IAM roles to avoid hard-coded keys and minimize security risks.

1.5 Approach

1. Local Backup Agent

- Use Python (argparse or a JSON configuration file) to specify source paths.

- Traverse directories with `os.walk` and upload each file to S3 using Boto3, embedding a timestamp and host identifier in the object key.
- Log successes and failures via Python's logging module.

2. Lambda-Based Replication

- Author a Lambda function (Python runtime) that lists objects in the source bucket, compares LastModified timestamps, and copies only newer objects to the destination bucket using Boto3.
- Assign an IAM role to Lambda with `s3:ListBucket`, `s3:GetObject` on the source bucket, and `s3:ListBucket`, `s3:PutObject` on the destination bucket.
- Configure environment variables (`SOURCE_BUCKET`, `DESTINATION_BUCKET`, `AWS_REGION`) in Lambda.

3. Scheduling

- Local-to-Cloud Backup: The Python agent uses an internal time-based scheduler (e.g., a looping timer or schedule library) to trigger uploads at regular intervals; in production, the same script can be executed via Windows Task Scheduler on a defined schedule.
- Bucket-to-Bucket Replication: AWS EventBridge is configured with an `ObjectCreated` rule on the source bucket to immediately invoke the Lambda function whenever new or modified objects appear.

4. Organization & Retention

- Timestamped object keys allow chronological sorting.
- Outline a simple retention script to delete objects older than a specified threshold.

Chapter 2

Technologies

This project leverages a combination of programming languages, cloud services, and scheduling tools to achieve automated, cost-aware, and secure backups:

- **Python 3.x:** A versatile, high-level scripting language used to develop the local backup agent. Its rich standard library and third-party packages (e.g., `schedule`, `argparse`) simplify file system traversal, argument parsing, and task scheduling.
- **AWS S3 (Simple Storage Service):** An object storage service offering virtually unlimited scalability and “eleven nines” of durability. S3 serves as the primary repository for uploaded backups and provides internal replication across Availability Zones for fault tolerance [3].
- **AWS Lambda:** A serverless compute service that runs code in response to events. Here, Lambda functions perform bucket-to-bucket replication, copying only new or modified objects without provisioning or managing servers [7].
- **AWS EventBridge (CloudWatch Events):** A fully managed event bus that dispatches events based on rules. We configure an `ObjectCreated` rule to detect new or changed objects in the source S3 bucket and trigger the replication Lambda immediately [8].
- **AWS IAM (Identity and Access Management):** Manages users, roles, and permissions. Least-privilege IAM roles secure both the local agent (via credentials loaded from environment variables) and the Lambda function, granting only necessary S3 and EventBridge actions [9].
- **AWS CloudWatch Logs:** A centralized logging service for collecting and monitoring log data from AWS services and custom applications. It provides visibility into backup uploads and replication events, facilitating troubleshooting and auditability.
- **Boto3:** The official AWS SDK for Python, enabling programmatic interaction with S3, IAM, EventBridge, and other AWS services. Its high-level abstractions streamline API calls for uploading files, listing objects, and invoking copies.

Chapter 3

Methodology

Account & Buckets

1. AWS Free Tier Sign-up

- An AWS Free Tier account is created to utilize free usage of S3, Lambda, and EventBridge. This ensures cost-effective development and testing of the backup and replication system.

2. Bucket Creation

Two Amazon S3 buckets are created within the same AWS region:

- cloud-project-pushti: Used for uploading local files via the backup Python script.
- my-lab-bucket-harvi-147: Used for replicated files copied by the Lambda function.

This structure helps separate original uploads from replicated data for monitoring and validation.

IAM Setup

1. IAM User for Python Agent

- A new IAM user is created specifically for running the local Python backup script.
- This user is granted PutObject permission on the source-backup-bucket using a custom policy.

2. IAM Role for Lambda Function

A Lambda execution role is created with the following permissions:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket",
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::cloud-project-pushti",
        "arn:aws:s3:::cloud-project-pushti /*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket",
```

```

        "s3:PutObject"
    ],
    "Resource": [
        "arn:aws:s3:::my-lab-bucket-harvi-147",
        "arn:aws:s3:::my-lab-bucket-harvi-147/*"
    ]
}

```

This role allows the Lambda to read and replicate files between buckets.

Local Backup Script

1. Configuration Parsing

- The backup script takes configuration via command-line arguments or JSON file input.
- Config includes source directory, S3 bucket name, AWS credentials path, logging options.

2. Directory Traversal and Upload

- The script walks through the provided directory structure recursively.
- Each file is uploaded to S3 using the Boto3 method:


```
s3_client.upload_file(local_path, bucket_name, s3_key)
```
- Key naming format: hostname/YYYYMMDD_HHMMSS/filename, enabling easy grouping by machine and timestamp.

3. Logging Activities

- All upload actions, including success/failure messages and timestamps, are printed to the console.
- Optionally, logs are written to a local .log file for persistent tracking.

Lambda Replication Function

1. Set required environment variables:

```

export SOURCE_BUCKET=cloud-project-pushti
export DESTINATION_BUCKET=my-lab-bucket-harvi-147
export AWS_REGION=eu-north-1

```

2. Pagination Handling and Object Listing

- Lambda function uses `list_objects_v2()` to retrieve object keys from source-backup-bucket.
- Pagination is handled to ensure listing all files even for large datasets.

3. Object Comparison and Replication

- For each object:
 - Check if it is present in the destination bucket.

- If missing or if the LastModified timestamp is newer than the last sync:

```
s3_client.copy_object(Bucket=destination_bucket, CopySource={'Bucket':
source_bucket, 'Key': key}, Key=key)
```

4. Logging to CloudWatch

- The Lambda logs all operations including which files were copied or skipped to Amazon CloudWatch Logs.
- These logs help in monitoring automated replication success and diagnosing errors.

Scheduling

EventBridge (Cloud Scheduling)

1. Hourly Trigger Setup

- An Amazon EventBridge rule is created with a cron expression such as:
`cron(0 * * * ? *)` — triggers every hour.
- This rule invokes the Lambda function automatically to ensure regular replication.
- Security Settings:
 - Set to 'Run whether user is logged on or not'.
 - Store credentials securely for unattended execution.

Testing & Validation

1. Local Backup Verification

- Run the backup script manually using the CLI.
- Confirm that files are present in the source-backup-bucket via:
 - AWS S3 Console
 - AWS CLI: `aws s3 ls s3://source-backup-bucket --recursive`

2. Lambda Replication Validation

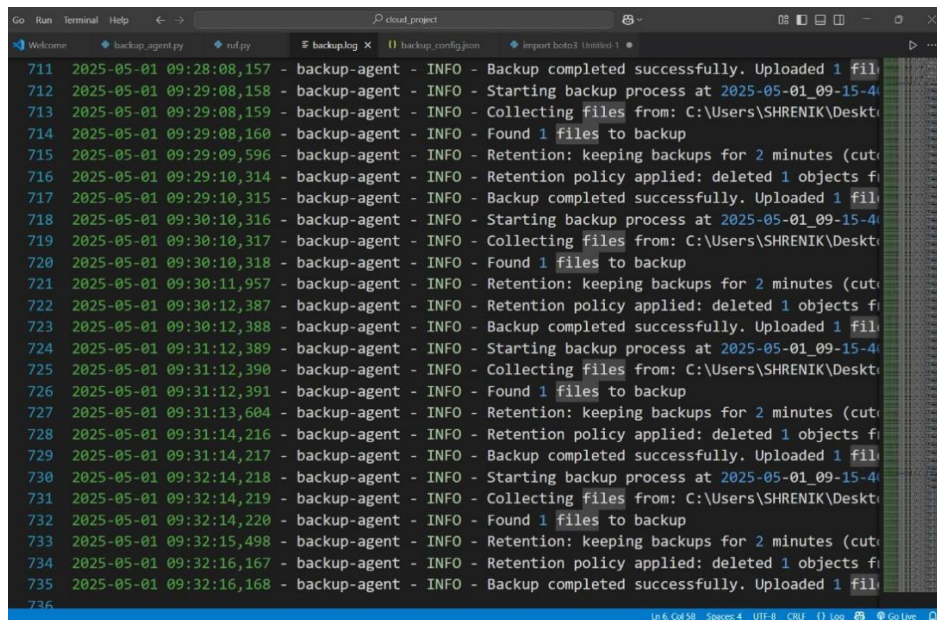
- Manually trigger the Lambda or wait for EventBridge schedule.
- Confirm replicated files appear in destination-replica-bucket.
- Review CloudWatch Logs for:
 - Files copied
 - Errors (if any)
 - Sync status

Chapter 4

Result and Discussion

The proposed system was evaluated through a series of functional tests designed to validate file transfer reliability, replication accuracy, scheduling efficacy, and logging behaviour. All components were deployed within the AWS Free Tier, with special emphasis on automation and observability. Screenshots and log outputs were captured to support the findings and are referenced throughout this section.

The local-to-cloud backup script, when triggered via its internal scheduler, successfully scanned the specified directory, identified new or modified files, and uploaded them to the designated source S3 bucket. Each uploaded object followed a consistent naming convention embedding a timestamp and preserving original file hierarchy. Snapshots of the S3 console confirmed that the files were correctly organized and visible within the defined bucket path. The `boto3`-based implementation ensured fault-tolerant uploads, and the Python `logging` module recorded detailed logs of each operation, including start time, files uploaded, and any encountered exceptions. A sample log file output is shown in *Figure 4.1*, while *Figure 4.2.1* and *4.2.2* illustrates the S3 object layout post-upload in source and destination buckets.



```
711 2025-05-01 09:28:08,157 - backup-agent - INFO - Backup completed successfully. Uploaded 1 fil
712 2025-05-01 09:29:08,158 - backup-agent - INFO - Starting backup process at 2025-05-01 09-15-4
713 2025-05-01 09:29:08,159 - backup-agent - INFO - Collecting files from: C:\Users\SHRENIK\Deskt
714 2025-05-01 09:29:08,160 - backup-agent - INFO - Found 1 files to backup
715 2025-05-01 09:29:09,596 - backup-agent - INFO - Retention: keeping backups for 2 minutes (cut
716 2025-05-01 09:29:10,314 - backup-agent - INFO - Retention policy applied: deleted 1 objects fi
717 2025-05-01 09:29:10,315 - backup-agent - INFO - Backup completed successfully. Uploaded 1 fil
718 2025-05-01 09:30:10,316 - backup-agent - INFO - Starting backup process at 2025-05-01 09-15-4
719 2025-05-01 09:30:10,317 - backup-agent - INFO - Collecting files from: C:\Users\SHRENIK\Deskt
720 2025-05-01 09:30:10,318 - backup-agent - INFO - Found 1 files to backup
721 2025-05-01 09:30:11,957 - backup-agent - INFO - Retention: keeping backups for 2 minutes (cut
722 2025-05-01 09:30:12,387 - backup-agent - INFO - Retention policy applied: deleted 1 objects fi
723 2025-05-01 09:30:12,388 - backup-agent - INFO - Backup completed successfully. Uploaded 1 fil
724 2025-05-01 09:31:12,389 - backup-agent - INFO - Starting backup process at 2025-05-01 09-15-4
725 2025-05-01 09:31:12,390 - backup-agent - INFO - Collecting files from: C:\Users\SHRENIK\Deskt
726 2025-05-01 09:31:12,391 - backup-agent - INFO - Found 1 files to backup
727 2025-05-01 09:31:13,604 - backup-agent - INFO - Retention: keeping backups for 2 minutes (cut
728 2025-05-01 09:31:14,216 - backup-agent - INFO - Retention policy applied: deleted 1 objects fi
729 2025-05-01 09:31:14,217 - backup-agent - INFO - Backup completed successfully. Uploaded 1 fil
730 2025-05-01 09:32:14,218 - backup-agent - INFO - Starting backup process at 2025-05-01 09-15-4
731 2025-05-01 09:32:14,219 - backup-agent - INFO - Collecting files from: C:\Users\SHRENIK\Deskt
732 2025-05-01 09:32:14,220 - backup-agent - INFO - Found 1 files to backup
733 2025-05-01 09:32:15,498 - backup-agent - INFO - Retention: keeping backups for 2 minutes (cut
734 2025-05-01 09:32:16,167 - backup-agent - INFO - Retention policy applied: deleted 1 objects fi
735 2025-05-01 09:32:16,168 - backup-agent - INFO - Backup completed successfully. Uploaded 1 fil
736
```

Figure 4.1: Local backup log file

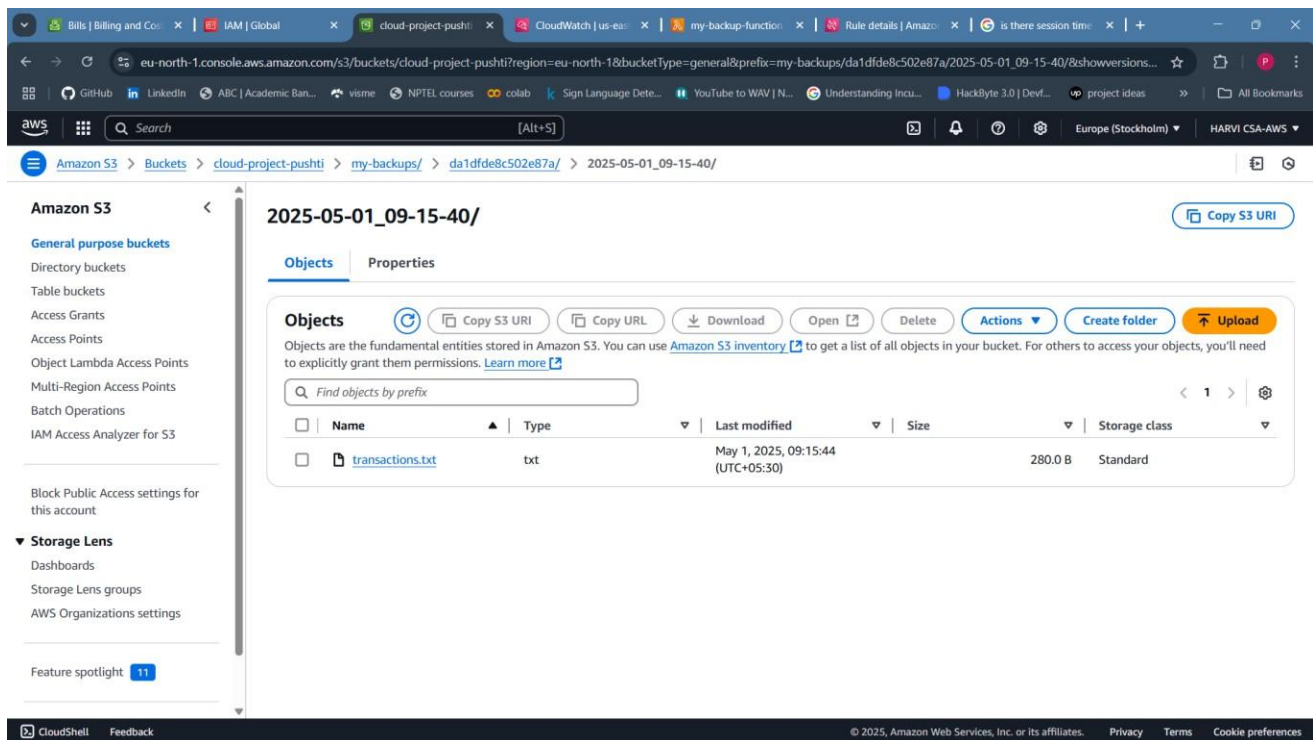


Figure 4.2.1: Source Bucket

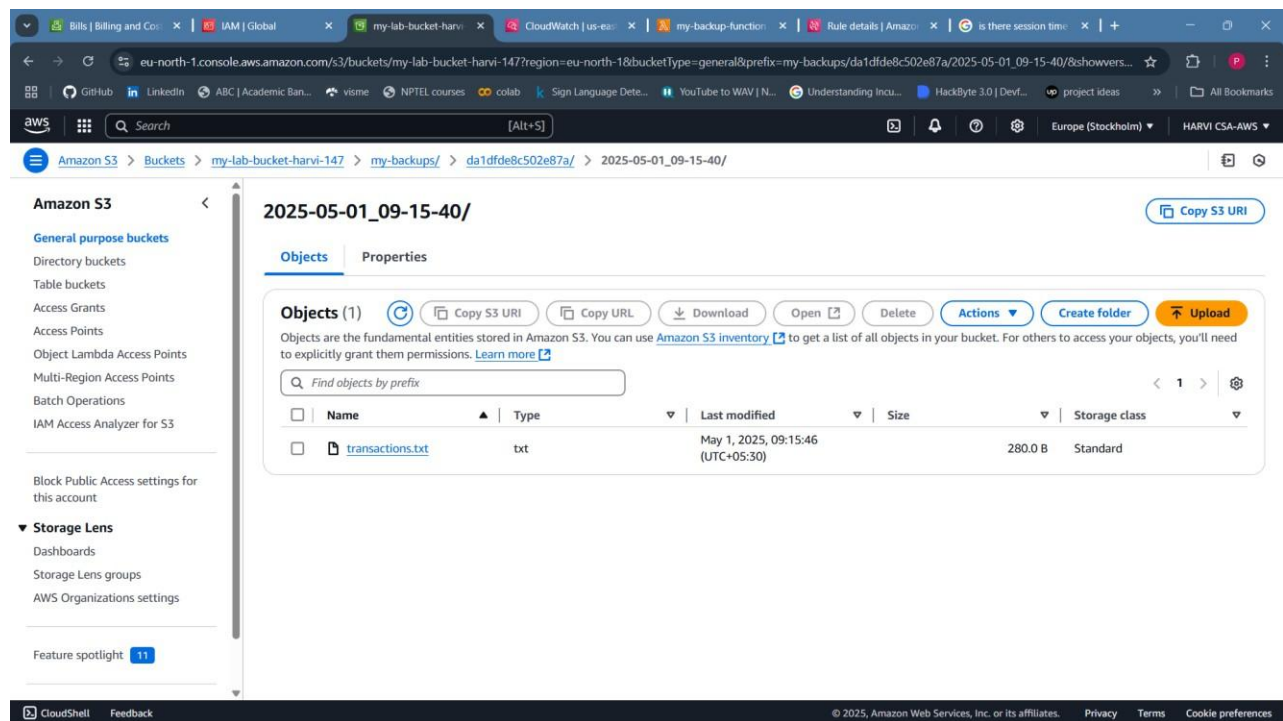


Figure 4.2.2: Destination bucket

For the bucket-to-bucket replication, the AWS Lambda function was deployed with environment variables specifying the source and destination buckets. It was configured to run via an EventBridge rule that listened for `ObjectCreated` events in the source bucket. Once new files were detected, Lambda copied them to the destination bucket if they were not already present or if the source file had a newer timestamp. This process effectively mimicked native S3 replication. The correctness of this logic was verified both via visual inspection of the destination bucket and by reviewing CloudWatch logs, which captured Lambda invocations, files copied, and replication outcomes. *Figure 4.3* shows the EventBridge rule configuration, while *Figure 4.4* presents a snapshot of the replication logs in AWS CloudWatch.

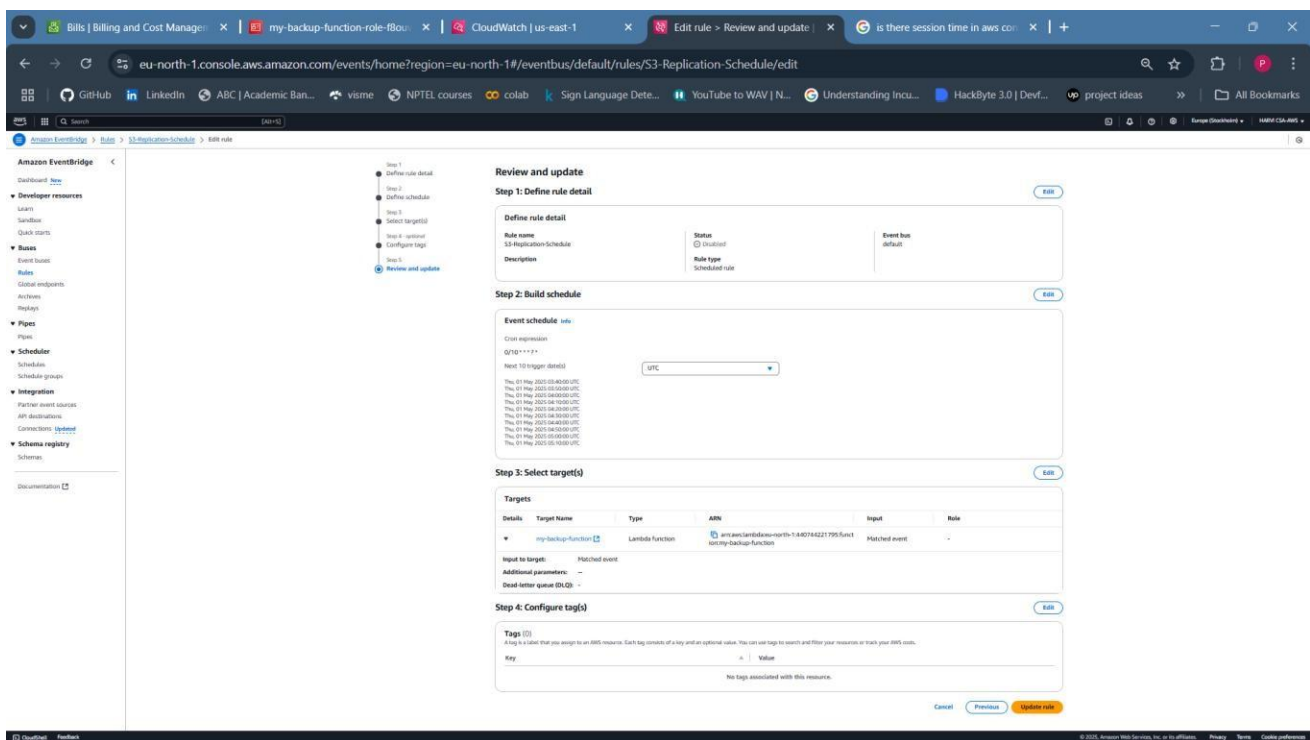


Figure 4.3: EventBridge Rule

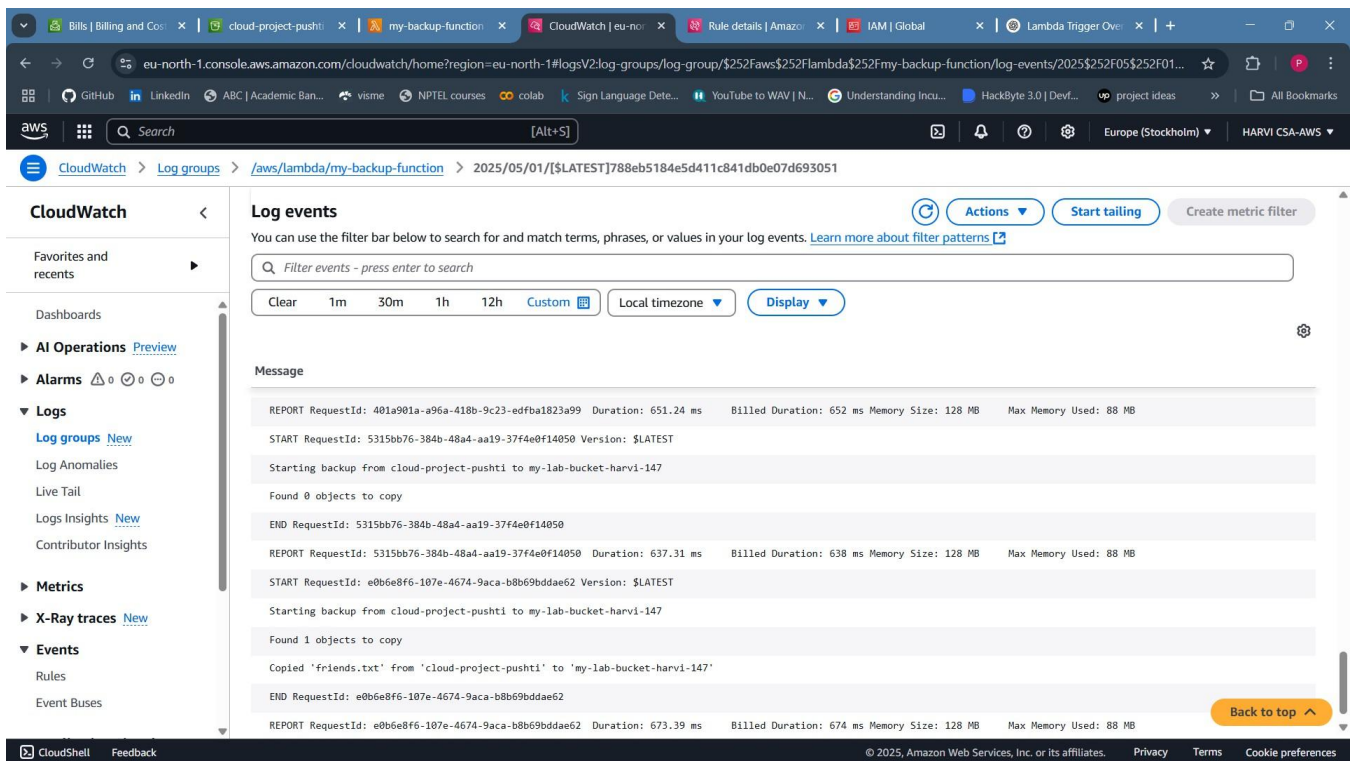


Figure 4.4: CloudWatch Logs

Retention testing was performed by setting a cutoff value in the local backup script, which deleted any previously uploaded objects older than the configured number of minutes(here). This feature ensured that outdated backups did not persist indefinitely, maintaining cost-efficiency and preventing storage bloat. Retention actions were logged in the same manner as uploads, and verified via subsequent inspection of the source bucket.

For simplicity and clarity during validation, encryption was not implemented in this project. This decision was intentional, as it allowed visual verification of exact file content, structure, and names both at the point of upload and after replication. This transparency was important for confirming the integrity of the replication logic without the added complexity of encryption layers.

Additionally, each uploaded file was replicated to only one destination bucket, serving as a simplified demonstration of the replication process. In real-world scenarios, replication factors may vary depending on redundancy requirements, with multiple copies distributed across regions or accounts for fault tolerance and disaster recovery. However, for the purpose of this project, a single replica within the same re-

gion was sufficient to illustrate the internal working of event-driven replication using AWS Lambda and EventBridge.

Throughout the evaluation, no file duplication or integrity loss was observed. The solution performed consistently, demonstrating reliable scheduling, accurate incremental replication, and complete logging.

Limitations:

While the system effectively demonstrates backup automation and intra-region redundancy, several limitations must be acknowledged. First, cross-region replication was intentionally excluded to avoid data transfer fees under the AWS Free Tier. Second, advanced storage-class transitions, such as moving backups to S3 Glacier or Deep Archive for long-term storage, were not implemented due to similar cost concerns. Finally, graphical interfaces and real-time status dashboards were beyond the scope of this implementation but represent areas for future enhancement.

Chapter 5

Conclusions and Future Work

The primary objective of this project was to design and implement a reliable and automated backup and replication system leveraging Amazon Web Services (AWS) and Python. The implemented solution successfully achieves the core goal of periodically backing up local files to a source Amazon S3 bucket and replicating those files to a destination S3 bucket using a Lambda function.

By breaking down the system into modular components — IAM setup, local Python-based backup script, AWS Lambda replication function, and scheduling mechanisms using EventBridge and python standard time library(for demonstration purposes) — the solution ensures reliability, automation, and flexibility. The backup process is configurable, can be triggered both manually and via a scheduler, and supports consistent naming conventions for easy retrieval and tracking. Replication is intelligently handled by comparing timestamps to avoid redundant transfers.

CloudWatch Logs provide transparency and visibility into replication activities, enhancing monitoring and debugging. Testing validated the system's accuracy in uploading and replicating files while handling common failure points such as network interruptions or permission issues. Overall, the system proved to be a cost-effective and scalable solution suitable for small to medium-sized data backup requirements.

Future Scope

While the current implementation serves its intended purpose, there are multiple avenues for future improvement and expansion:

1. Cross-Region Replication

Extending the architecture to support cross-region S3 replication would enhance disaster recovery capabilities by protecting data against regional outages.

2. Versioning and Retention Policies

Enabling S3 versioning would allow tracking changes to files and recovering from accidental deletions. Coupled with lifecycle policies, it would help manage storage costs by automatically transitioning or deleting older versions.

3. Web-Based Dashboard

A user-friendly dashboard could be developed using frameworks like Flask or Django to allow users to view backup status, configure settings, and initiate manual backups through a web interface.

4. Monitoring and Alerting System

Integration with AWS SNS or third-party services like PagerDuty or Slack can notify users of failures or replication delays, ensuring timely resolution of issues.

5. File Integrity Verification

Implementing checksum-based verification (like MD5 or SHA256) between source and destination files can ensure data integrity post-transfer.

6. Multi-user Support

Enhancing the backup agent to support multiple users or systems, each with their unique configuration, would make the solution scalable in enterprise environments.

References

- [1] B. Klaus and P. Horn, Robot Vision. Cambridge, MA: MIT Press, 1986.
- [2] L. Stein, “Random patterns,” in Computers and You, J. S. Brake, Ed. New York: Wiley, 1994, pp. 55-70.
- [3] Amazon Web Services, “*Backup and recovery using Amazon S3*,” AWS Prescriptive Guidance, accessed Apr. 2025.
- [4] Amazon Web Services, “*Creating a rule that runs on a schedule in Amazon EventBridge*,” AWS EventBridge User Guide, 2025.
- [5] M. Abramowitz and I. A. Stegun, Eds., Handbook of Mathematical Functions (Applied Mathematics Series 55). Washington, DC: NBS, 1964, pp. 32-33.
- [6] Transmission Systems for Communications, 3rd ed., Western Electric Co., Winston Salem, NC, 1985, pp. 44–60.
- [7] Amazon Web Services, “*What is AWS Lambda?*,” AWS Lambda Developer Guide, 2025.
- [8] Amazon Web Services, “*Monitoring AWS Backup events using Amazon EventBridge*,” AWS Backup Developer Guide, 2024.
- [9] Amazon Web Services, “*AWS Identity and Access Management (IAM) API Reference – Welcome*,” AWS, 2025.