Cloud-Based Document Management System with AWS Elasticsearch

A Course Project Report Submitted in partial fulfillment of the course requirements for the award of grades in the subject of

CLOUD BASED AIML SPECIALITY
(22SDCS07A)

by

Deekshitha Bethireddy

(2210030070)

Under the esteemed guidance of

Ms. P. Sree Lakshmi
Assistant Professor,
Department of Computer Science and Engineering

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

K L Deemed to be UNIVERSITY

Aziznagar, Moinabad, Hyderabad,
Telangana, Pincode: 500075

April 2025

K L Deemed to be UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Certificate

This is Certified that the project entitled **“Cloud-Based Document Management System with AWS Elasticsearch”** which is an experimental & Simulation work carried out by Deekshitha Bethireddy (2210030070), in partial fulfillment of the course requirements for the award of grades in the subject of **CLOUD BASED AIML SPECIALITY**, during the year **2024-2025**. The project has been approved as it satisfies the academic requirements.

Ms.P.Sree Lakshmi	**Dr. Arpita Gupta**
Course Coordinator	**Head of the Department**

Ms. P. Sree Lakshmi

Course Instructor

CONTENTS

1. INTRODUCTION

In the digital era, organizations generate vast amounts of documents, necessitating efficient storage, retrieval, and search capabilities. Traditional document management systems often face challenges with scalability, search performance, and infrastructure management. A serverless cloud-based solution addresses these issues by leveraging managed services for scalability, cost-efficiency, and performance. The Cloud-Based Document Management System with AWS Elasticsearch utilizes Amazon OpenSearch Service (formerly AWS Elasticsearch) to enable fast, full-text search and analytics on documents stored in Amazon S3, integrated with serverless AWS services for automation and accessibility [1][2].

This project builds a scalable, serverless document management system that allows users to upload documents to an S3 bucket, automatically index their content in OpenSearch, and perform keyword-based searches via an API Gateway. The system supports various file types, such as text (e.g., “design_principles.txt”), and uses AWS Lambda for automating text extraction and indexing. It is particularly valuable for applications like digital libraries, enterprise content management, and archival systems, where quick document retrieval enhances productivity and decision-making [3][4].

By integrating services like Amazon S3, AWS Lambda, Amazon OpenSearch Service, and API Gateway, this project delivers a fault-tolerant, scalable, and secure document management solution. The use of serverless technologies minimizes operational overhead, while OpenSearch ensures robust search capabilities. This project demonstrates the power of AWS-native services in building modern, cloud-based document management systems that meet the demands of dynamic business environments [5].

This architecture is particularly valuable for industries such as education, healthcare, and corporate environments, where quick access to documents improves decision-making and operational efficiency.

2. AWS Services Used as part of the project

1. **Amazon S3 – Scalable Document Storage**

Amazon S3 serves as the primary storage layer for documents, offering:

- Highly durable and scalable storage for files like PDFs and text documents.
- Event notifications to trigger AWS Lambda functions upon file uploads.
- Integration with other AWS services for processing and archival [5].
- Supports lifecycle policies for cost-effective archival of older documents.

2. **Amazon OpenSearch Service – Full-Text Search and Analytics**

Amazon OpenSearch Service (formerly AWS Elasticsearch) powers the search functionality, enabling:

- Full-text indexing and searching of document content using a documents index.
- Real-time search queries for keywords (e.g., “aboutMe” or “design”).
- Scalable and managed search infrastructure for high-performance retrieval [1].
- Scales automatically to handle large document collections.

3. **AWS Lambda – Serverless Compute for Automation**

AWS Lambda automates document processing and indexing, providing:

- Text extraction from uploaded documents using libraries like pymupdf.
- Indexing extracted content into OpenSearch for searchability.

- Event-driven processing triggered by S3 uploads or API Gateway requests [5].

4. AWS API Gateway – Secure API for Search Queries

AWS API Gateway enables secure and scalable access to the document search functionality:

- Exposes RESTful endpoints for querying the OpenSearch index.
- Integrates with Lambda to process search requests and return results.
- Implements authentication and throttling to ensure secure access.
- Supports testing via tools like Postman for debugging and validation [6].

5. AWS IAM (Identity & Access Management) – Security & Access Control

AWS IAM ensures secure access to AWS resources:

- Role-based access control (RBAC) for users like dms-admin and roles like dms-ExtractText-role-gvkjhfb.
- Policies granting permissions for S3, OpenSearch, Lambda, and API Gateway.
- Encryption and authentication to protect document data [5].
- Tracks access logs via CloudTrail for auditing purposes.
  This enables monitoring and review of all system activities.

6. Amazon CloudWatch – Monitoring & Logging

Amazon CloudWatch monitors system performance and logs:

- CloudWatch tracks AWS Lambda executions, S3 events, and API Gateway performance metrics in real time.
- CloudWatch triggers alerts for operational anomalies such as failed uploads, high error rates, or service downtime.
- Tracks Lambda function executions, API Gateway requests, and OpenSearch performance.
- Logs errors like NoSuchKey or 403 Forbidden for debugging.
- Provides metrics for optimizing resource utilization [5].

3. STEPS INVOLVED IN SOLVING PROJECT PROBLEM STATEMENT

Understanding the Problem Statement

- Identify the need for a scalable document management system to store, index, and search documents like “about_design_v2.txt”.
- Define requirements for full-text search, automation of indexing, and secure access via APIs.
- Ensure scalability, low latency, and cost-efficiency using serverless AWS services.

Designing the AWS Architecture

- Use Amazon S3 as the storage layer for documents, with event triggers for new uploads [5].
- Implement Amazon OpenSearch Service to index document content in a documents index for fast searches [1].
- Deploy AWS Lambda to extract text from documents (e.g., using pymupdf for PDFs) and index it in OpenSearch [7].
- Configure AWS API Gateway to provide a RESTful interface for searching documents by keywords [6].
- Use Amazon CloudWatch to monitor system performance and log errors [5].

Data Ingestion & Processing

- Upload documents (e.g., “AI-ML_internship.pdf”) to an S3 bucket (documentt-storage-bucket).
- Trigger a Lambda function on S3 upload to extract text using pymupdf or similar libraries.
- Index extracted text in the OpenSearch documents index for searchability.
- Handle errors like NoSuchKey by verifying S3 object keys and paths.

Security & Access Control

- Set up IAM roles (e.g., dms-ExtractText-role-gvkjhfb) with policies like AmazonS3FullAccess and AmazonOpenSearchServiceFullAccess [5].
- Resolve 403 Forbidden errors for user dms-admin by updating OpenSearch domain policies and role mappings [8].
- Implement encryption for data at rest and in transit.

Monitoring & Optimization

- Use CloudWatch to monitor Lambda logs, API Gateway metrics, and OpenSearch performance [5].
- Optimize Lambda functions by using Lambda Layers for dependencies like pymupdf to avoid compatibility issues.
- Adjust OpenSearch shard configurations for efficient indexing and search.

Deployment & Testing

- Deploy the system, including S3 bucket, OpenSearch domain, Lambda functions, and API Gateway.
- Test search functionality using Postman with queries like “design” or “internship”.
- Debug issues like DNS errors (getaddrinfo ENOTFOUND) or empty search results by verifying configurations.

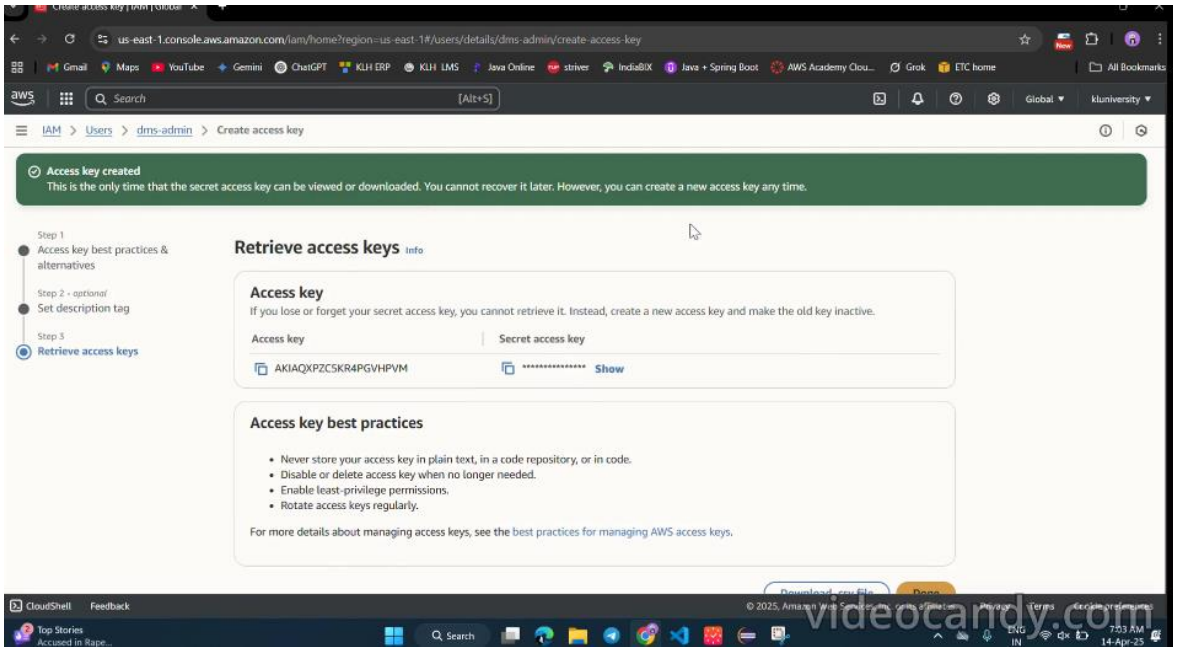
Search & Retrieval

- Query the OpenSearch index via API Gateway to retrieve documents matching keywords.
- Validate search results for accuracy and performance using tools like Postman.

4. STEPWISE SCREENSHOTS WITH BRIEF DESCRIPTION

Step 1: Create an IAM User for Secure Access

- Navigate to IAM, create a user (dms-admin), and attach policies:
  - AmazonS3FullAccess
  - AmazonOpenSearchServiceFullAccess
  - AWSLambdaFullAccess
- Generate an access key for programmatic access.



us-east-1.console.aws.amazon.com/iam/home?region=us-east-1#/users/details/dms-admin/create-access-key

IAM > Users > dms-admin > Create access key

Access key created
This is the only time that the secret access key can be viewed or downloaded. You cannot recover it later. However, you can create a new access key any time.

Step 1
Access key best practices & alternatives

Step 2 - optional
Set description tag

Step 3
Retrieve access keys

Retrieve access keys Info

Access key
If you lose or forget your secret access key, you cannot retrieve it. Instead, create a new access key and make the old key inactive.

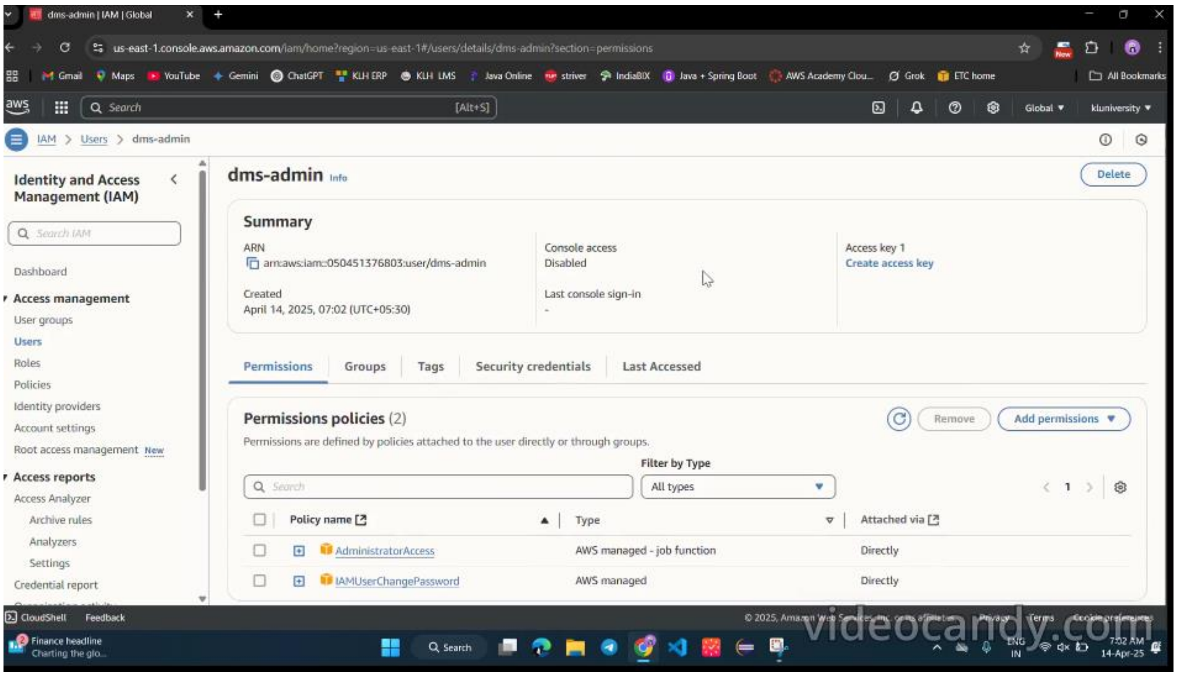
Access key	Secret access key
AKIAQXPZCSKR4PGVHPVM	*************** Show

Access key best practices

- Never store your access key in plain text, in a code repository, or in code.
- Disable or delete access key when no longer needed.
- Enable least-privilege permissions.
- Rotate access keys regularly.

For more details about managing access keys, see the best practices for managing AWS access keys.

Fig 4.1.1 Creating the IAM user dms-admin and access key.


dms-admin

Summary

ARN: arn:aws:iam::050451376803:user/dms-admin

Created: April 14, 2025, 07:02 (UTC+05:30)

Console access: Disabled

Last console sign-in: -

Access key 1: Create access key

Permissions policies (2)

Permissions are defined by policies attached to the user directly or through groups.

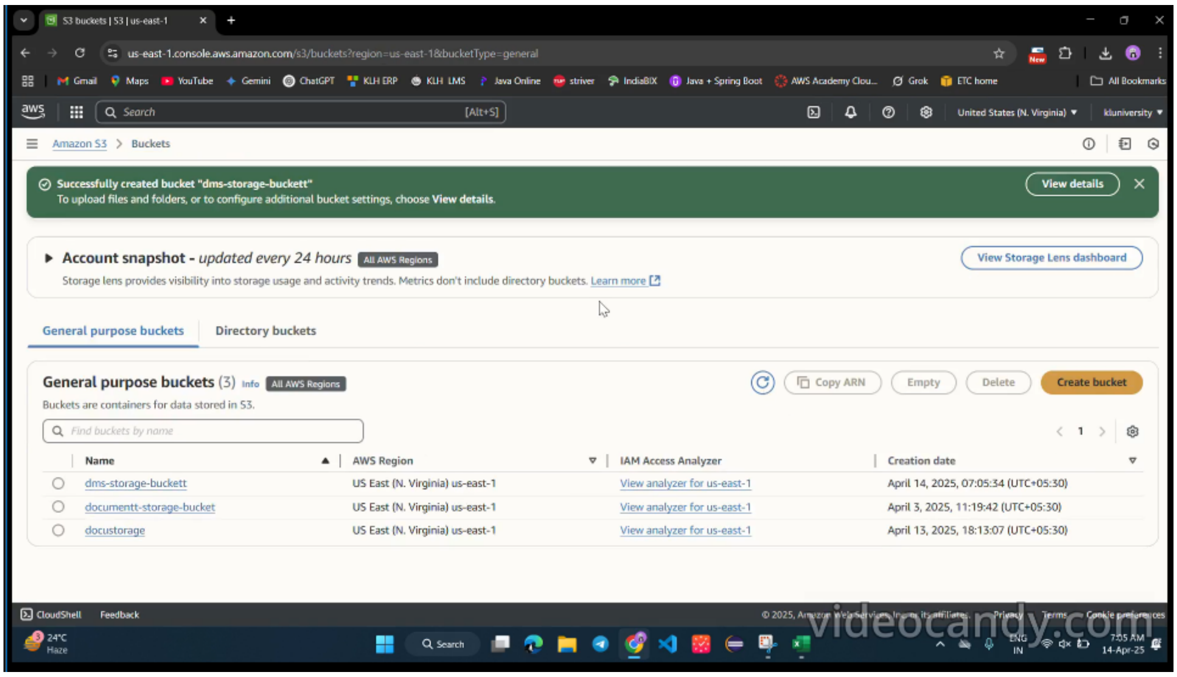
Filter by Type: All types

Policy name	Type	Attached via
AdministratorAccess	AWS managed - job function	Directly
IAMUserChangePassword	AWS managed	Directly

Fig 4.1.2 Attaching required policies for S3, OpenSearch, and Lambda.

Step 2 : Create an S3 Bucket for Document Storage

- Create a general-purpose S3 bucket (documentt-storage-bucket) for storing documents.
- Configure event notifications to trigger Lambda on file uploads.


Successfully created bucket "dms-storage-buckett"

To upload files and folders, or to configure additional bucket settings, choose **View details**.

Account snapshot - *updated every 24 hours*

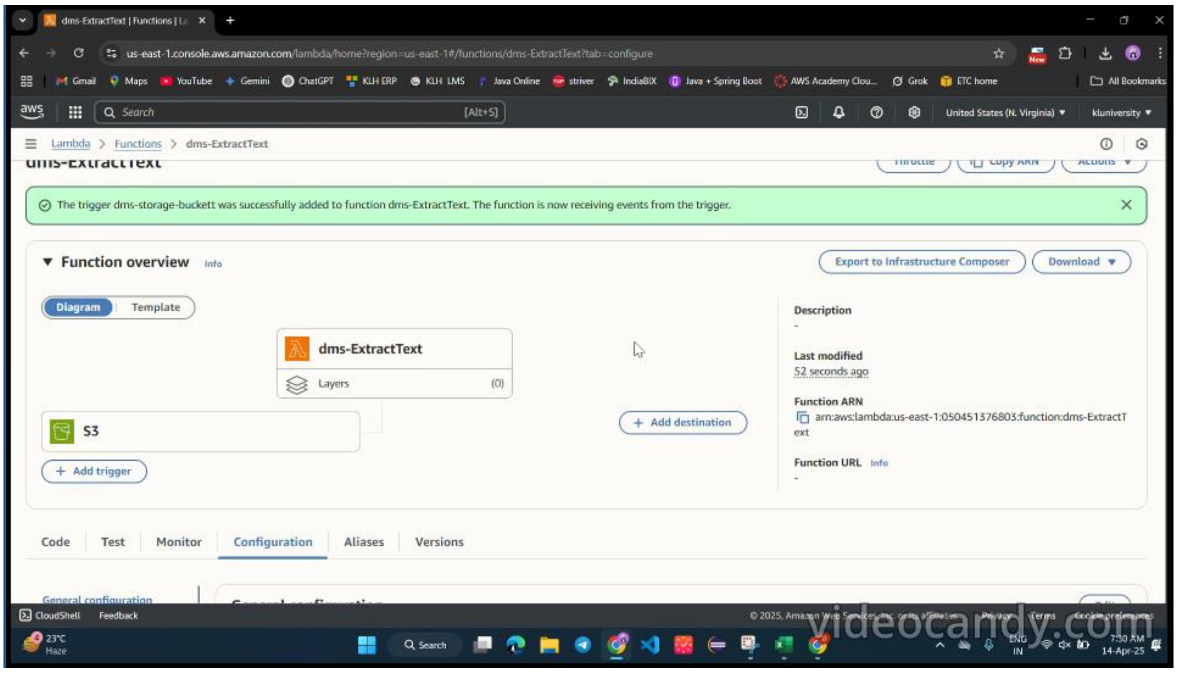
Storage lens provides visibility into storage usage and activity trends. Metrics don't include directory buckets. Learn more

General purpose buckets (3)

Buckets are containers for data stored in S3.

Name	AWS Region	IAM Access Analyzer	Creation date
dms-storage-buckett	US East (N. Virginia) us-east-1	View analyzer for us-east-1	April 14, 2025, 07:05:34 (UTC+05:30)
documentt-storage-bucket	US East (N. Virginia) us-east-1	View analyzer for us-east-1	April 3, 2025, 11:19:42 (UTC+05:30)
docustorage	US East (N. Virginia) us-east-1	View analyzer for us-east-1	April 13, 2025, 18:13:07 (UTC+05:30)

Fig 4.2.1 Successfully created S3 bucket.

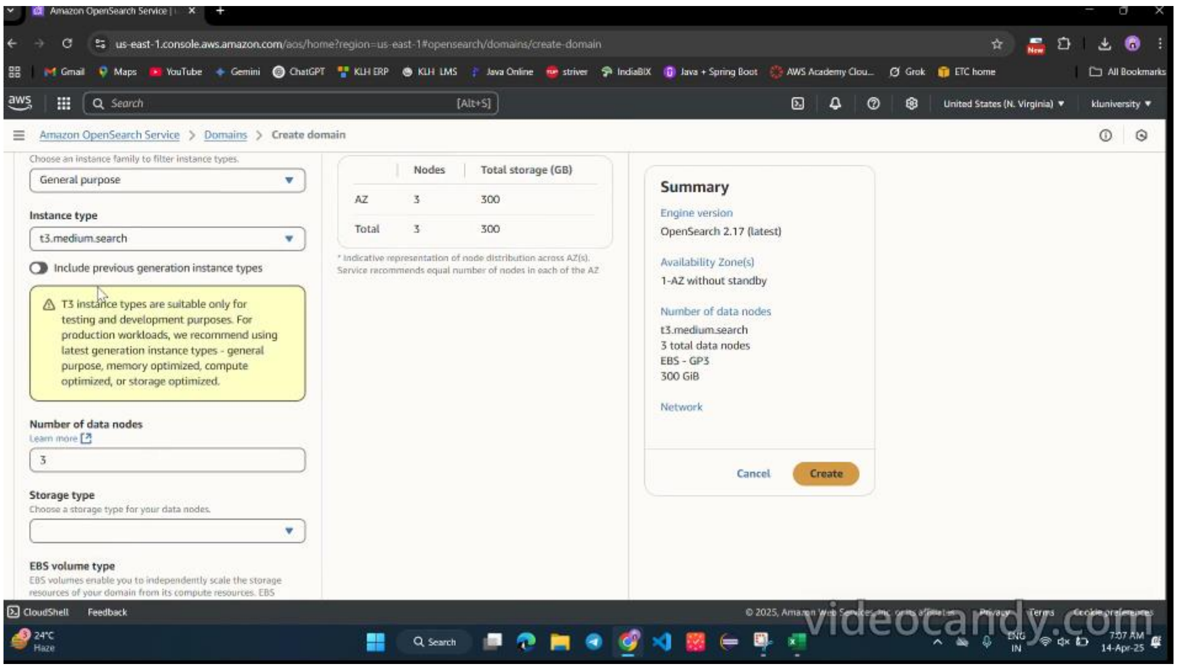

The trigger dms-storage-buckett was successfully added to function dms-ExtractText. The function is now receiving events from the trigger.

Function ARN: arn:aws:lambda:us-east-1:050451376803:function:dms-ExtractText

Fig 4.2.2   Configuring S3 event notification for Lambda.

Step 3 : Set Up Amazon OpenSearch Service

- Create an OpenSearch domain and configure the documents index for document content.
- Click “Create domain” and name the domain (e.g., dms-search).
- Select the “Development and testing” deployment type for cost-efficiency during prototyping.
- Choose the latest OpenSearch version available (e.g., OpenSearch 2.11) for optimal features.
- Configure instance types (e.g., t3.small.search) and set 1 data node for simplicity.
- Verify domain status as active.



Amazon OpenSearch Service > Domains > Create domain

Choose an instance family to filter instance types.

General purpose

Instance type

t3.medium.search

Include previous generation instance types

T3 instance types are suitable only for testing and development purposes. For production workloads, we recommend using latest generation instance types - general purpose, memory optimized, compute optimized, or storage optimized.

Number of data nodes

Learn more

3

Storage type

Choose a storage type for your data nodes.

EBS volume type

EBS volumes enable you to independently scale the storage resources of your domain from its compute resources. EBS

	Nodes	Total storage (GB)
AZ	3	300
Total	3	300

* Indicative representation of node distribution across AZ(s). Service recommends equal number of nodes in each of the AZ

Summary

Engine version

OpenSearch 2.17 (latest)

Availability Zone(s)

1-AZ without standby

Number of data nodes

t3.medium.search

3 total data nodes

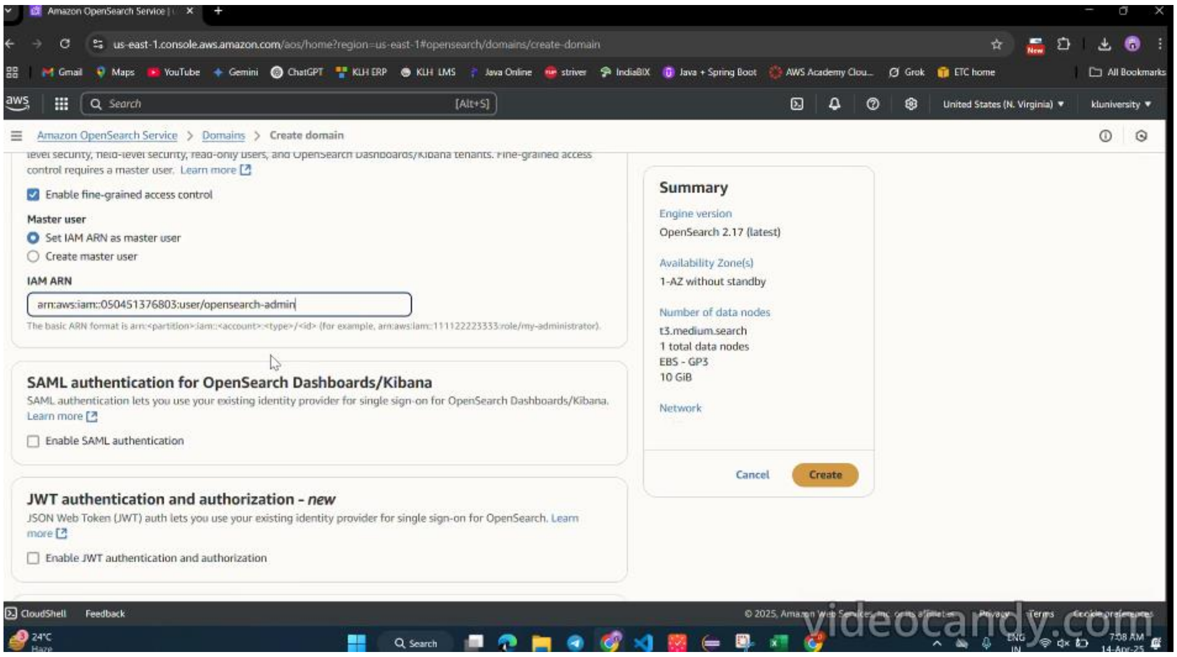
EBS - GP3

300 GiB

Network

Cancel Create

Fig 4.3.1 Accessing the OpenSearch Service console.



Amazon OpenSearch Service > Domains > Create domain

level security, field-level security, read-only users, and OpenSearch Dashboards/Kibana tenants. Fine-grained access control requires a master user. Learn more

☑ Enable fine-grained access control

Master user

● Set IAM ARN as master user

○ Create master user

IAM ARN

arn:aws:iam::050451376803:user/opensearch-admin

The basic ARN format is arn:<partition>:iam::<account>:<type>/<id> (for example, arn:aws:iam::111122223333:role/my-administrator).

SAML authentication for OpenSearch Dashboards/Kibana

SAML authentication lets you use your existing identity provider for single sign-on for OpenSearch Dashboards/Kibana. Learn more

☐ Enable SAML authentication

JWT authentication and authorization - *new*

JSON Web Token (JWT) auth lets you use your existing identity provider for single sign-on for OpenSearch. Learn more

☐ Enable JWT authentication and authorization

Summary

Engine version

OpenSearch 2.17 (latest)

Availability Zone(s)

1-AZ without standby

Number of data nodes

t3.medium.search

1 total data nodes

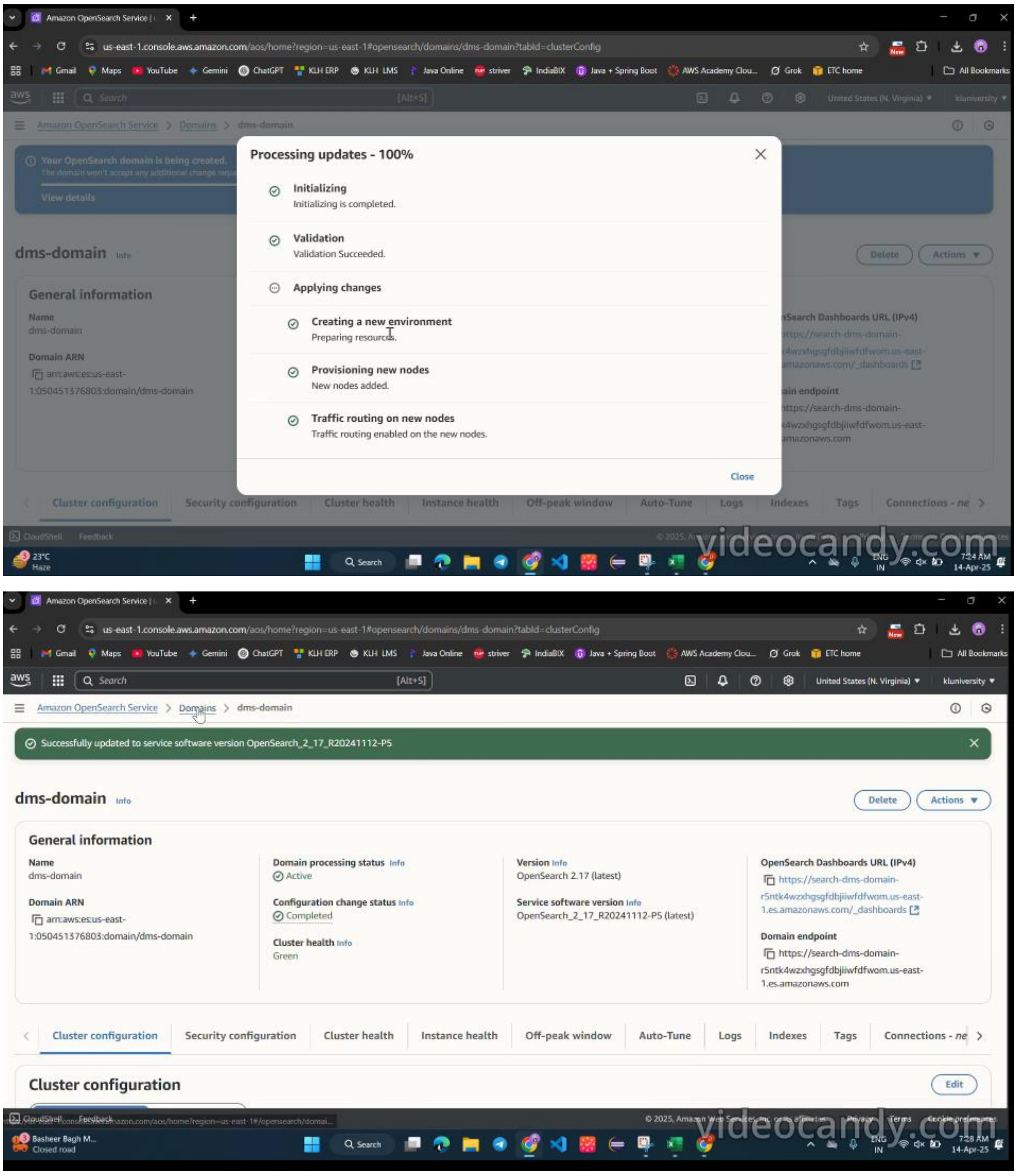
EBS - GP3

10 GiB

Network

Cancel Create

Fig 4.3.2   Configuring the OpenSearch domain settings.


Processing updates - 100%

Initializing
Initializing is completed.

Validation
Validation Succeeded.

Applying changes

Creating a new environment
Preparing resources.

Provisioning new nodes
New nodes added.

Traffic routing on new nodes
Traffic routing enabled on the new nodes.

Successfully updated to service software version OpenSearch_2_17_R20241112-P5

dms-domain

General information

Name
dms-domain

Domain ARN
arn:aws:es:us-east-1:050451376803:domain/dms-domain

Domain processing status
Active

Configuration change status
Completed

Cluster health
Green

Version
OpenSearch 2.17 (latest)

Service software version
OpenSearch_2_17_R20241112-P5 (latest)

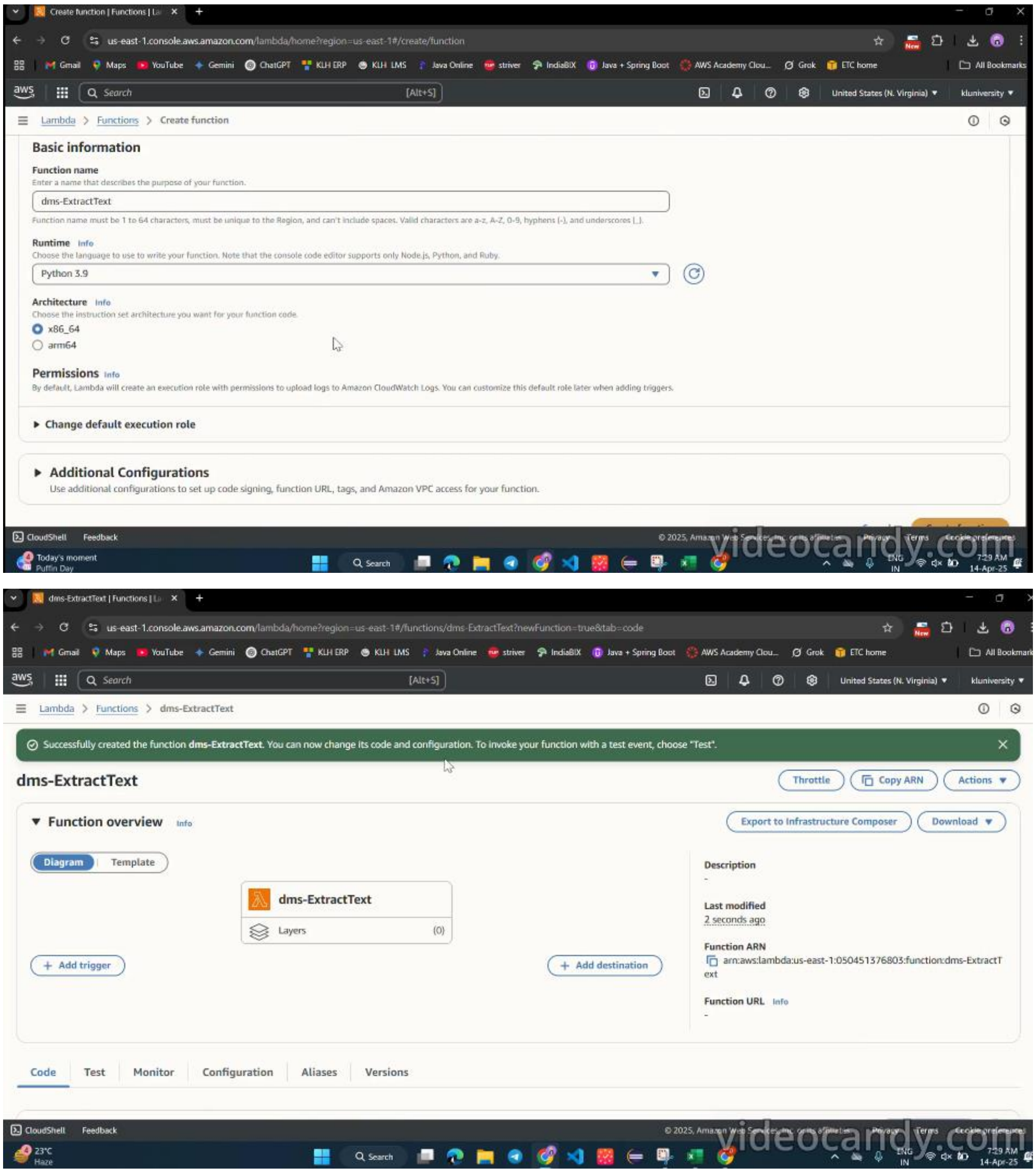
OpenSearch Dashboards URL (IPv4)
https://search-dms-domain-r5ntk4wzxhgsgfdbjiiwfdfwom.us-east-1.es.amazonaws.com/_dashboards

Domain endpoint
https://search-dms-domain-r5ntk4wzxhgsgfdbjiiwfdfwom.us-east-1.es.amazonaws.com

Fig 4.3.3 OpenSearch domain status successful

Step 4 : Create a Lambda Function for Indexing

- Create a Lambda function with an IAM role (dms-ExtractText-role-gvkjhfb).
- Add a Lambda Layer for pymupdf to handle PDF text extraction.
- Write code to extract text from S3 uploads and index it in OpenSearch.



Lambda > Functions > Create function

Basic information

Function name
Enter a name that describes the purpose of your function.
dms-ExtractText
Function name must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (_).

Runtime Info
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.
Python 3.9

Architecture Info
Choose the instruction set architecture you want for your function code.
[✓ x86_64]
arm64

Permissions Info
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

▸ Change default execution role

▸ Additional Configurations
Use additional configurations to set up code signing, function URL, tags, and Amazon VPC access for your function.

Lambda > Functions > dms-ExtractText

Successfully created the function **dms-ExtractText**. You can now change its code and configuration. To invoke your function with a test event, choose "Test".

dms-ExtractText

▾ Function overview Info

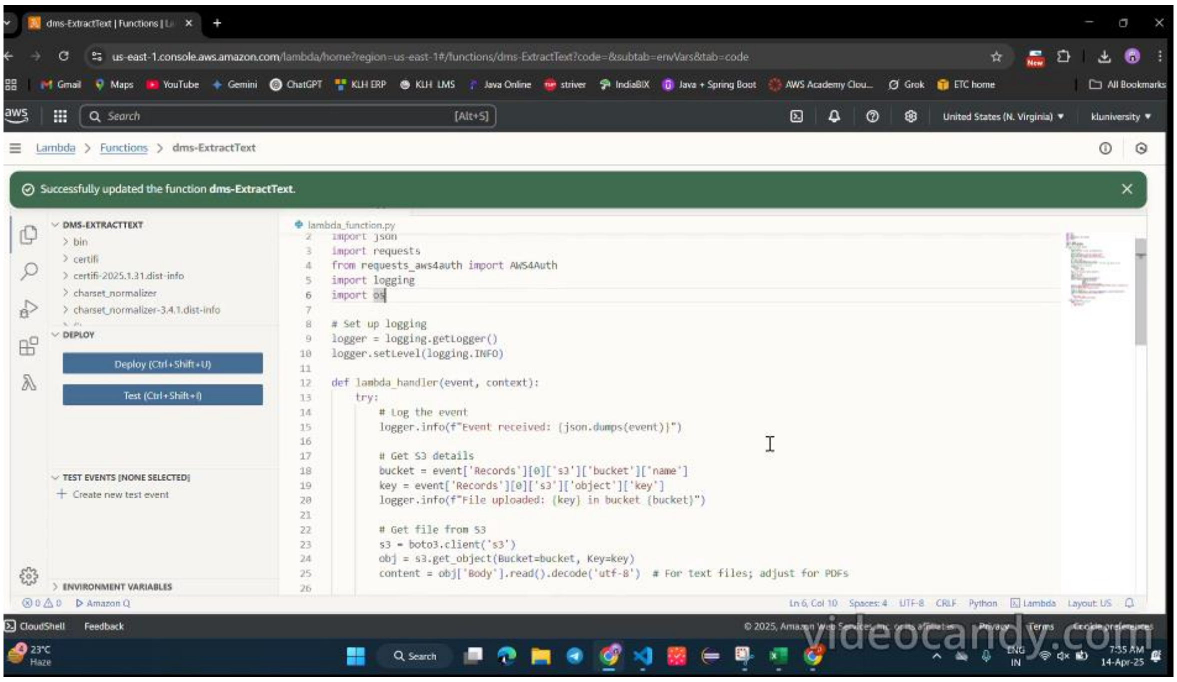
Description
-

Last modified
2 seconds ago

Function ARN
arn:aws:lambda:us-east-1:050451376803:function:dms-ExtractText

Function URL Info
-

Fig 4.4.1 Creating the Lambda function.



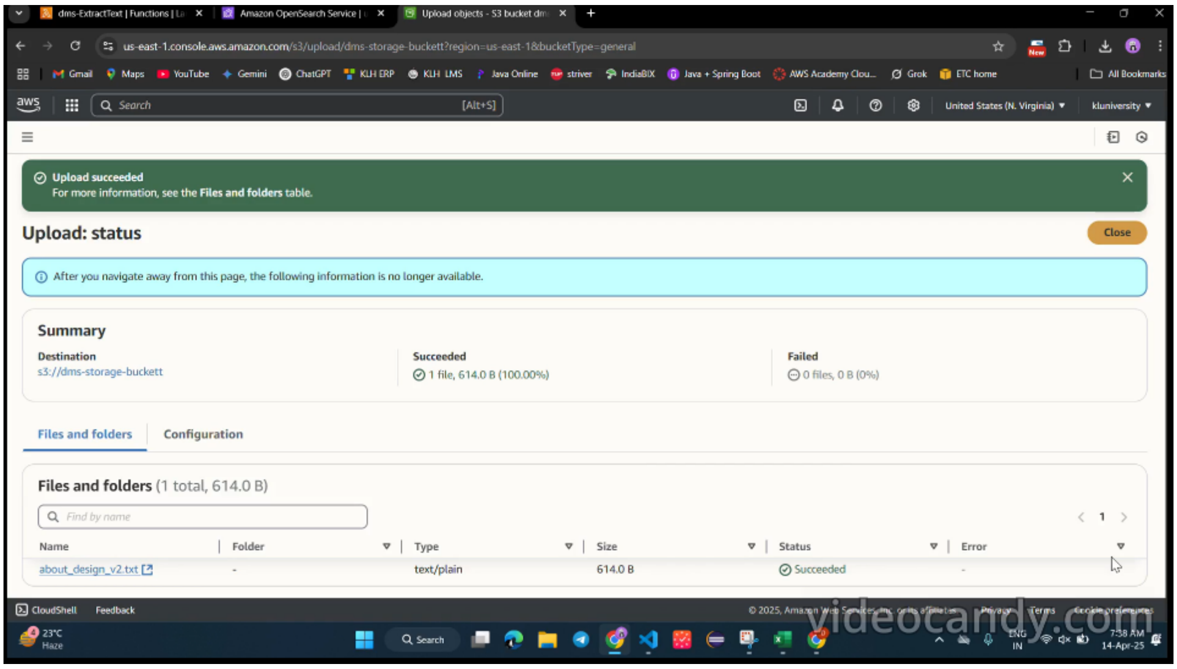
Successfully updated the function dms-ExtractText.

```python
import json
import requests
from requests_aws4auth import AWS4Auth
import logging
import os

# Set up logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):
    try:
        # Log the event
        logger.info(f"Event received: {json.dumps(event)}")

        # Get S3 details
        bucket = event['Records'][0]['s3']['bucket']['name']
        key = event['Records'][0]['s3']['object']['key']
        logger.info(f"File uploaded: {key} in bucket {bucket}")

        # Get file from S3
        s3 = boto3.client('s3')
        obj = s3.get_object(Bucket=bucket, Key=key)
        content = obj['Body'].read().decode('utf-8')  # For text files; adjust for PDFs

```

Fig 4.4.2   Lambda function code for indexing.

Step 5 : Upload and Test Documents

- Upload a test file (about_design_v2.txt) to the S3 bucket.
- Verify that the Lambda function indexes the file content in OpenSearch.



Upload succeeded
For more information, see the Files and folders table.

Upload: status

After you navigate away from this page, the following information is no longer available.

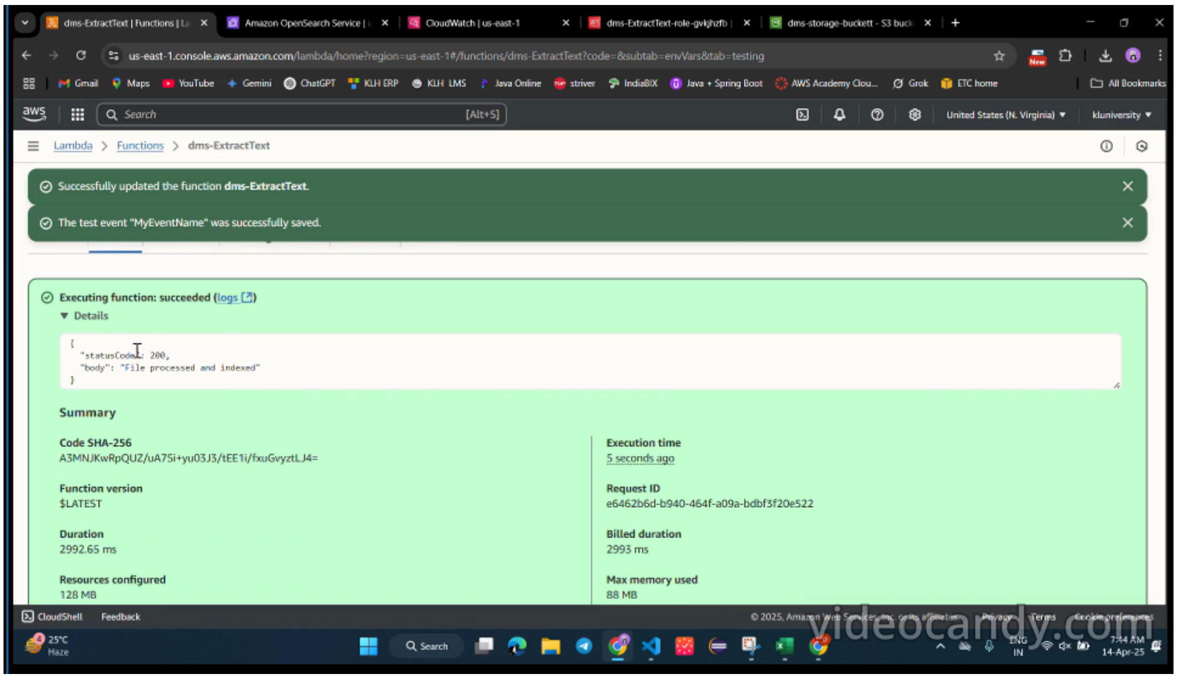
Summary

Destination	Succeeded	Failed
s3://dms-storage-buckett	1 file, 614.0 B (100.00%)	0 files, 0 B (0%)

Files and folders (1 total, 614.0 B)

Name	Folder	Type	Size	Status	Error
about_design_v2.txt	-	text/plain	614.0 B	Succeeded	-

Fig 4.5.1   Uploading about_design_v2.txt to S3.



Successfully updated the function **dms-ExtractText**.

The test event "MyEventName" was successfully saved.

Executing function: succeeded (logs)

```
{
  "statusCode": 200,
  "body": "File processed and indexed"
}
```

Summary

Code SHA-256
A3MNJKwRpQUZ/uA75i+yu03J3/tEE1i/fxuGvyztLJ4=

Function version
$LATEST

Duration
2992.65 ms

Resources configured
128 MB

Execution time
5 seconds ago

Request ID
e6462b6d-b940-464f-a09a-bdbf3f20e522

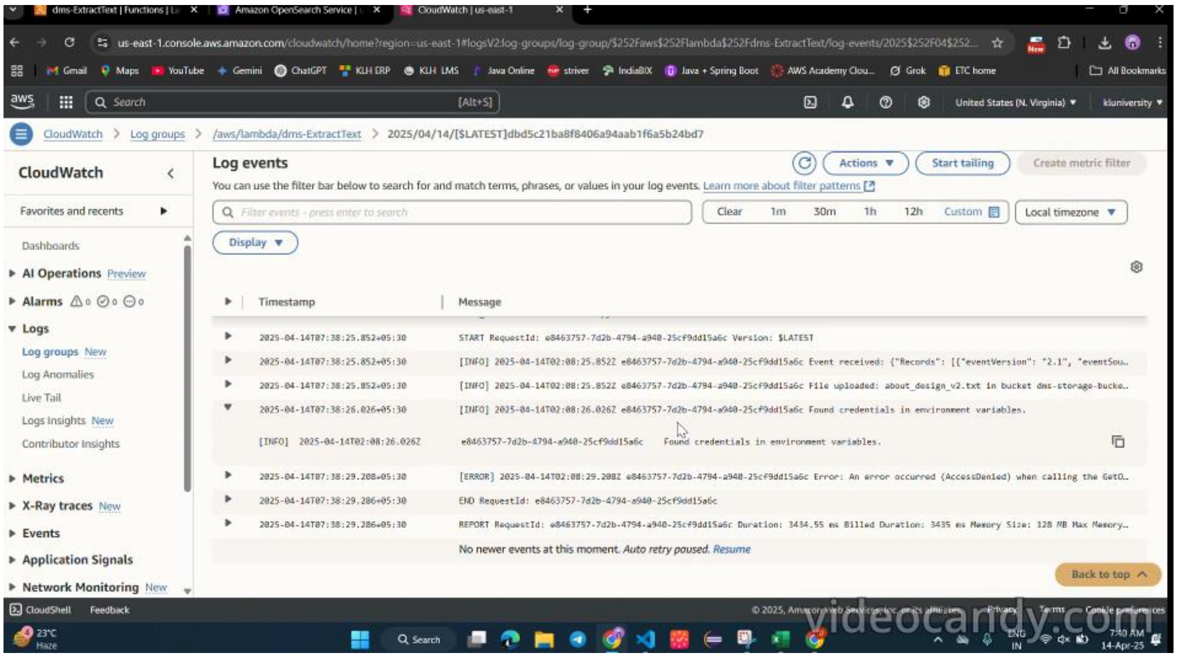
Billed duration
2993 ms

Max memory used
88 MB

Fig 4.5.2 Checking OpenSearch index for indexed content.

Step 6 : Monitor System Performance

- Use CloudWatch to monitor Lambda executions, API Gateway requests, and OpenSearch metrics.
- Check logs for errors like NoSuchKey or NameError.



CloudWatch > Log groups > /aws/lambda/dms-ExtractText > 2025/04/14/[$LATEST]dbd5c21ba8f8406a94aab1f6a5b24bd7

Log events

You can use the filter bar below to search for and match terms, phrases, or values in your log events.

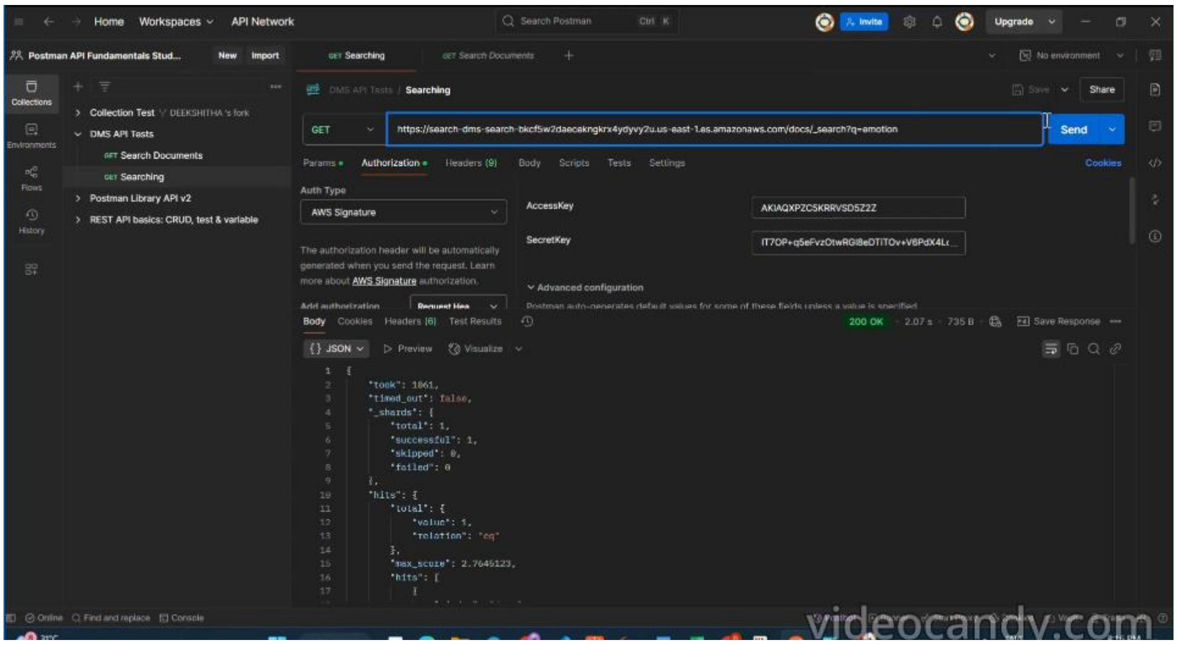
```
2025-04-14T07:38:25.852+05:30   START RequestId: e8463757-7d2b-4794-a940-25cf9dd15a6c Version: $LATEST
2025-04-14T07:38:25.852+05:30   [INFO] 2025-04-14T02:08:25.852Z e8463757-7d2b-4794-a940-25cf9dd15a6c Event received: {"Records": [{"eventVersion": "2.1", "eventSou…
2025-04-14T07:38:25.852+05:30   [INFO] 2025-04-14T02:08:25.852Z e8463757-7d2b-4794-a940-25cf9dd15a6c File uploaded: about_design_v2.txt in bucket dms-storage-bucke…
2025-04-14T07:38:26.026+05:30   [INFO] 2025-04-14T02:08:26.026Z e8463757-7d2b-4794-a940-25cf9dd15a6c Found credentials in environment variables.
    [INFO]  2025-04-14T02:08:26.026Z    e8463757-7d2b-4794-a940-25cf9dd15a6c    Found credentials in environment variables.
2025-04-14T07:38:29.208+05:30   [ERROR] 2025-04-14T02:08:29.208Z e8463757-7d2b-4794-a940-25cf9dd15a6c Error: An error occurred (AccessDenied) when calling the GetO…
2025-04-14T07:38:29.286+05:30   END RequestId: e8463757-7d2b-4794-a940-25cf9dd15a6c
2025-04-14T07:38:29.286+05:30   REPORT RequestId: e8463757-7d2b-4794-a940-25cf9dd15a6c Duration: 3434.55 ms Billed Duration: 3435 ms Memory Size: 128 MB Max Memory…
```

No newer events at this moment. *Auto retry paused.* Resume

Fig 4.6 CloudWatch logs for Lambda execution.

Step 7 : Test Search Functionality

- Query the API with keywords like “design” or “emotion ” using Postman.
- Verify search results and optimize query performance.



GET https://search-dms-search-bkcf5w2daecakngkrx4ydyvy2u.us-east-1.es.amazonaws.com/docs/_search?q=emotion

Auth Type: AWS Signature

AccessKey: AKIAQXPZC5KRRVSD5Z2Z

SecretKey: IT7OP+q5eFvzOtwRGIBeDTITOv+V6PdX4Lc...

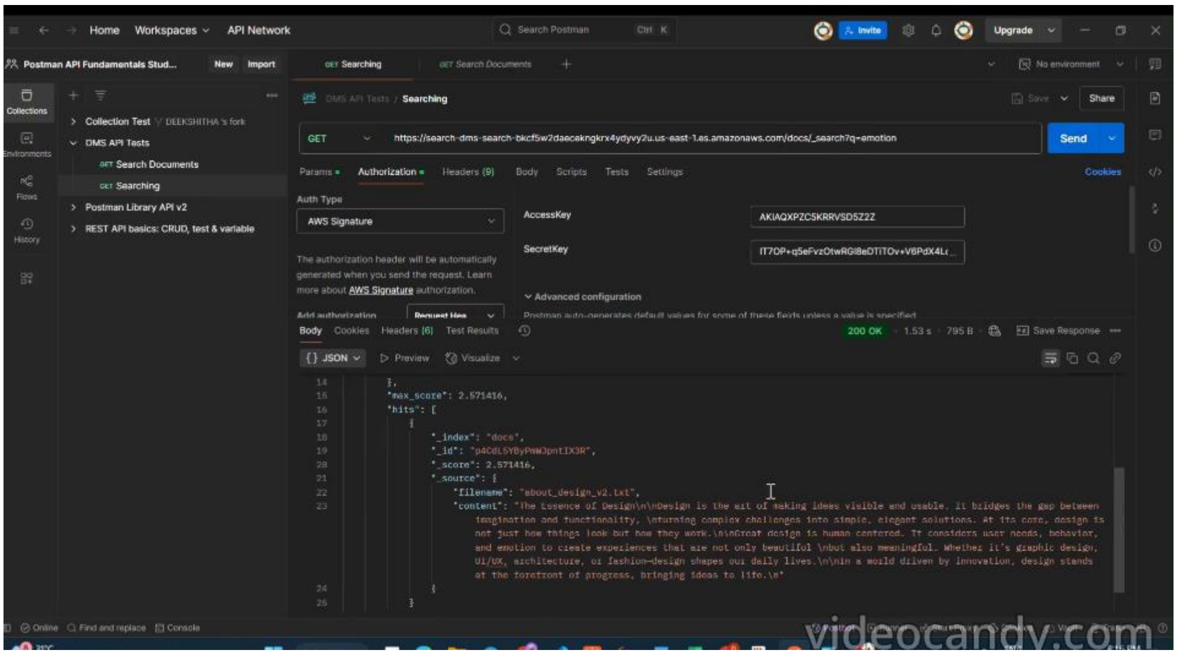
The authorization header will be automatically generated when you send the request. Learn more about AWS Signature authorization.

Advanced configuration

Postman auto-generates default values for some of these fields unless a value is specified

200 OK · 2.07 s · 735 B

```
{
    "took": 1861,
    "timed_out": false,
    "_shards": {
        "total": 1,
        "successful": 1,
        "skipped": 0,
        "failed": 0
    },
    "hits": {
        "total": {
            "value": 1,
            "relation": "eq"
        },
        "max_score": 2.7645123,
        "hits": [
            {
```



GET https://search-dms-search-bkcf5w2daecakngkrx4ydyvy2u.us-east-1.es.amazonaws.com/docs/_search?q=emotion

Auth Type: AWS Signature

AccessKey: AKIAQXPZC5KRRVSD5Z2Z

SecretKey: IT7OP+q5eFvzOtwRGIBeDTITOv+V6PdX4Lc...

The authorization header will be automatically generated when you send the request. Learn more about AWS Signature authorization.

Advanced configuration

Postman auto-generates default values for some of these fields unless a value is specified

200 OK · 1.53 s · 795 B

```
        },
        "max_score": 2.571416,
        "hits": [
            {
                "_index": "docs",
                "_id": "p4CdL5YByPmWJpntIX3R",
                "_score": 2.571416,
                "_source": {
                    "filename": "about_design_v2.txt",
                    "content": "The Essence of Design\n\nDesign is the art of making ideas visible and usable. It bridges the gap between
                        imagination and functionality, \nturning complex challenges into simple, elegant solutions. At its core, design is
                        not just how things look but how they work.\n\nGreat design is human centered. It considers user needs, behavior,
                        and emotion to create experiences that are not only beautiful \nbut also meaningful. Whether it's graphic design,
                        UI/UX, architecture, or fashion—design shapes our daily lives.\n\nIn a world driven by innovation, design stands
                        at the forefront of progress, bringing ideas to life.\n"
                }
            }
```

Fig 4.7.1 Postman response with search results.

5. LEARNING OUTCOMES

Through the development of the Cloud-Based Document Management System with AWS Elasticsearch, several key skills were acquired:

- Understanding AWS OpenSearch for Full-Text Search

  Gained hands-on experience in indexing and searching document content using Amazon OpenSearch Service, enabling efficient retrieval of documents by keywords [1].
- Serverless Automation with AWS Lambda

  Learned to automate document processing and indexing using Lambda, including text extraction with pymupdf and handling S3 events [7].
- Serverless Automation with AWS Lambda

  Mastered the use of S3 for scalable document storage and IAM for secure access control, resolving issues like 403 Forbidden errors [5][8].
- API Development with AWS API Gateway

  Developed skills in building RESTful APIs for search functionality, testing with Postman, and debugging issues like DNS errors [6].
- System Monitoring with Amazon CloudWatch

  Enhanced proficiency in monitoring system performance and debugging errors using CloudWatch logs and metrics [5].
- Scalability and Fault Tolerance in Serverless Architectures

  Designed a fault-tolerant, scalable system using serverless AWS services, optimizing for performance and cost-efficiency [5].

This project deepened the understanding of cloud-native architectures and practical applications of AWS services in document management.

6. CONCLUSION

The Cloud-Based Document Management System with AWS Elasticsearch demonstrated the power of AWS services like S3, OpenSearch, Lambda, and API Gateway in building a scalable, serverless solution for document storage, indexing, and search. The project successfully enabled the upload, indexing, and keyword-based retrieval of documents like “about_design_v2”, with a secure and automated workflow. It addressed challenges like permissions errors, dependency management, and search accuracy, showcasing the design of fault-tolerant and scalable cloud architectures. This hands-on experience strengthened skills in cloud computing, serverless automation, and search technologies, providing a solid foundation for future advancements in cloud-based document management and enterprise applications.

7. REFERENCES

[1] Amazon OpenSearch Service Documentation: https://docs.aws.amazon.com/opensearch-service/latest/developerguide/what-is.html

[2] AWS S3 Documentation: https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html

[3] AWS Lambda Documentation: https://docs.aws.amazon.com/lambda/latest/dg/welcome.html

[4] AWS API Gateway Documentation: https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html

[5] AWS CloudWatch Documentation: https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/WhatIsCloudWatch.html

[6] AWS IAM Documentation: https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html

[7] PyMuPDF Documentation: https://pymupdf.readthedocs.io/en/latest/

[8] Troubleshooting Amazon OpenSearch Service: https://docs.aws.amazon.com/opensearch-service/latest/developerguide/troubleshooting.html

[9] Building a Serverless Document Management System with AWS: https://aws.amazon.com/blogs/compute/building-a-serverless-document-management-system/