RISHIT BURMAN

Intelligent Financial Document Engine - Project Documentation

---

Overview

This project presents a robust **Intelligent Document Retrieval System** designed to handle the **extraction, semantic indexing, and question-answering over financial documents**. It leverages powerful AWS services including **Textract**, **DynamoDB**, **S3**, **SageMaker**, and **OpenSearch** for secure, scalable, and efficient processing of enterprise documents — while keeping data *private* (no use of public LLMs).

Why This Project Matters

Business Impact

- **Automates the reading and searching of thousands of financial reports**, audit documents, quarterly filings, etc.
- Eliminates manual efforts by finance teams, saving **hours of time per document**.
- Ensures **accuracy** using semantic search powered by **embeddings**.
- Keeps enterprise data **completely private** and never leaves AWS.

Potential Use-Cases:

- Financial firms (for annual/quarterly report analysis)
- Hospitals and pharma (for report lookup & compliance)

- Enterprises (for internal document retrieval)
- Law firms (for fast document clause retrieval)

Architecture Breakdown

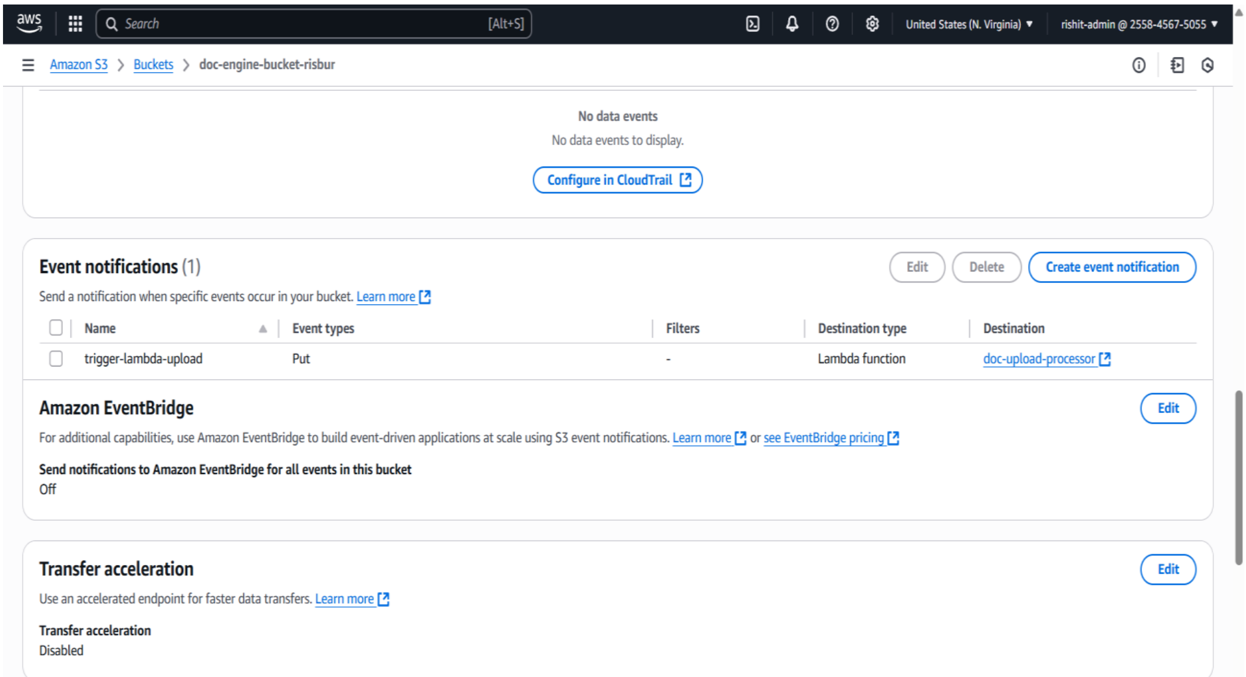
Part 1: Document Upload & Text Extraction

Flow:

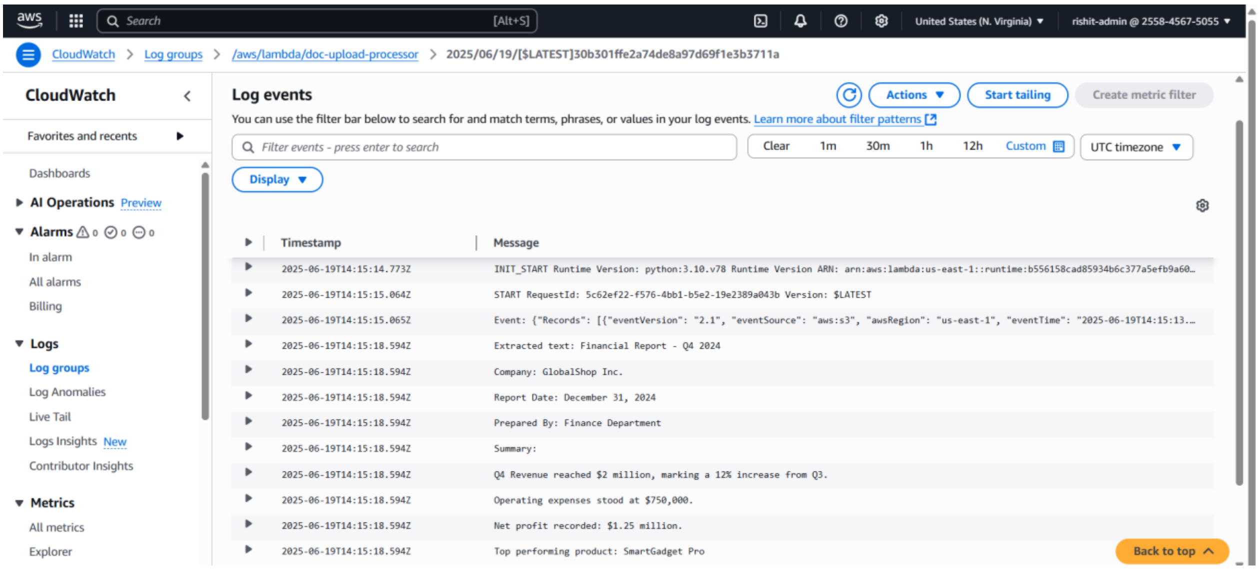
1. User uploads a PDF/Scanned image to **S3 Bucket** (doc-engine-bucket-risbur).
2. A PUT event triggers a **Lambda function** named doc-upload-processor.
3. This Lambda uses **Amazon Textract** to extract structured text from the uploaded document.
4. The text and document metadata (like file name) are stored in **DynamoDB table** DocumentTextTable.

✅ **Why Amazon Textract?**

- It supports scanned financial documents (images, tables, PDFs).
- Can extract text from complex layouts with high accuracy.
- Fully serverless and scalable.



- Upload event from S3 triggering Lambda.

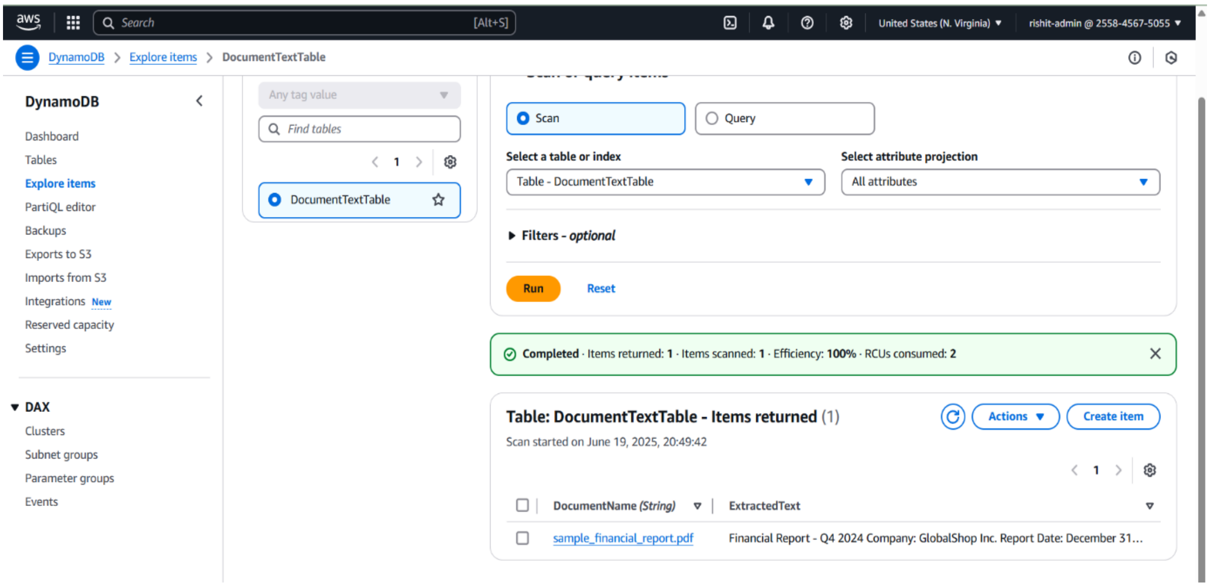


- Extracted Textract output from CloudWatch logs.

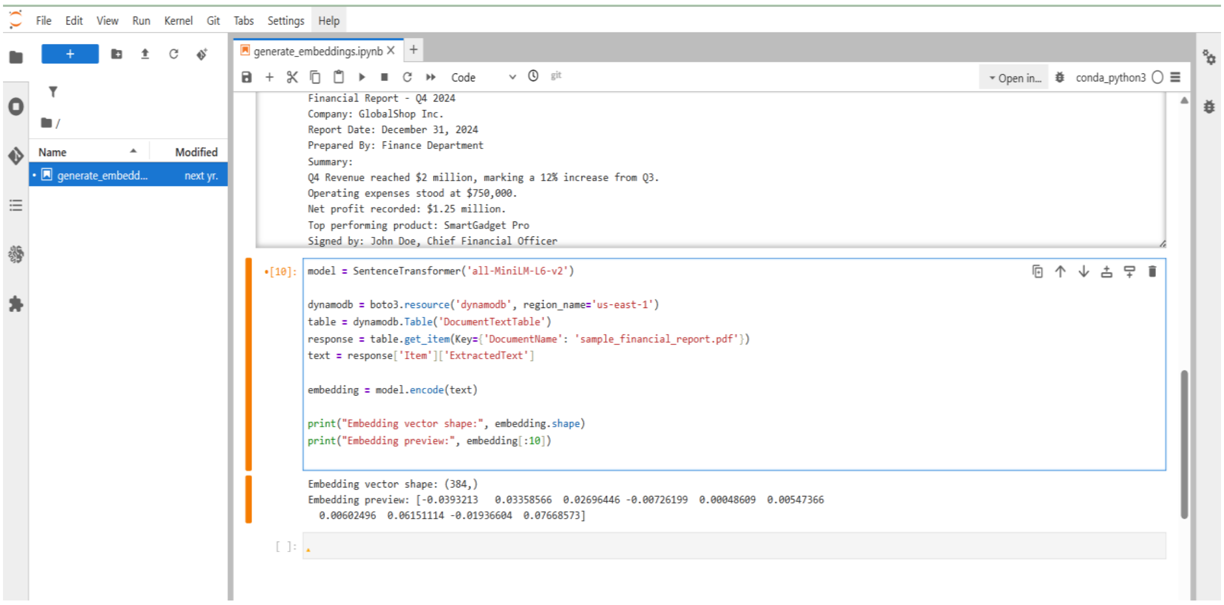
Part 2: Embedding Generation & Semantic Indexing

Tool: Jupyter Notebook (generate_embeddings.ipynb)

1. Loads each document's text from **DynamoDB**.

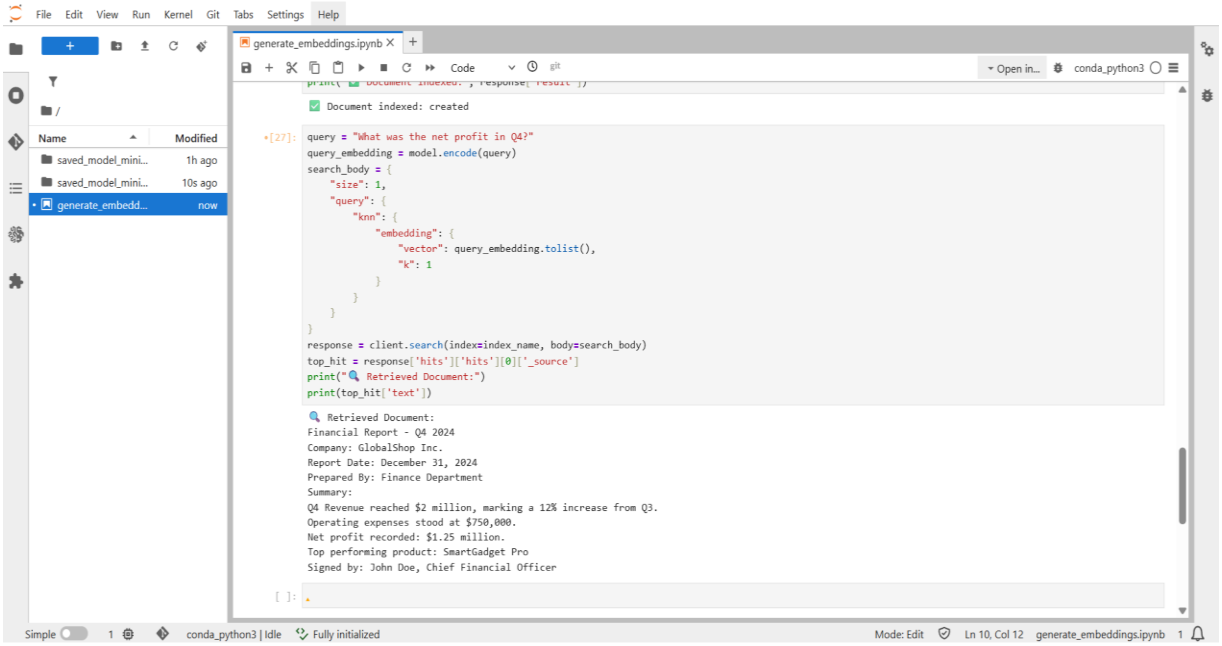


2. Uses a **SentenceTransformer MiniLM model** hosted on **SageMaker** to generate **text embeddings** (vector representations of document meaning).

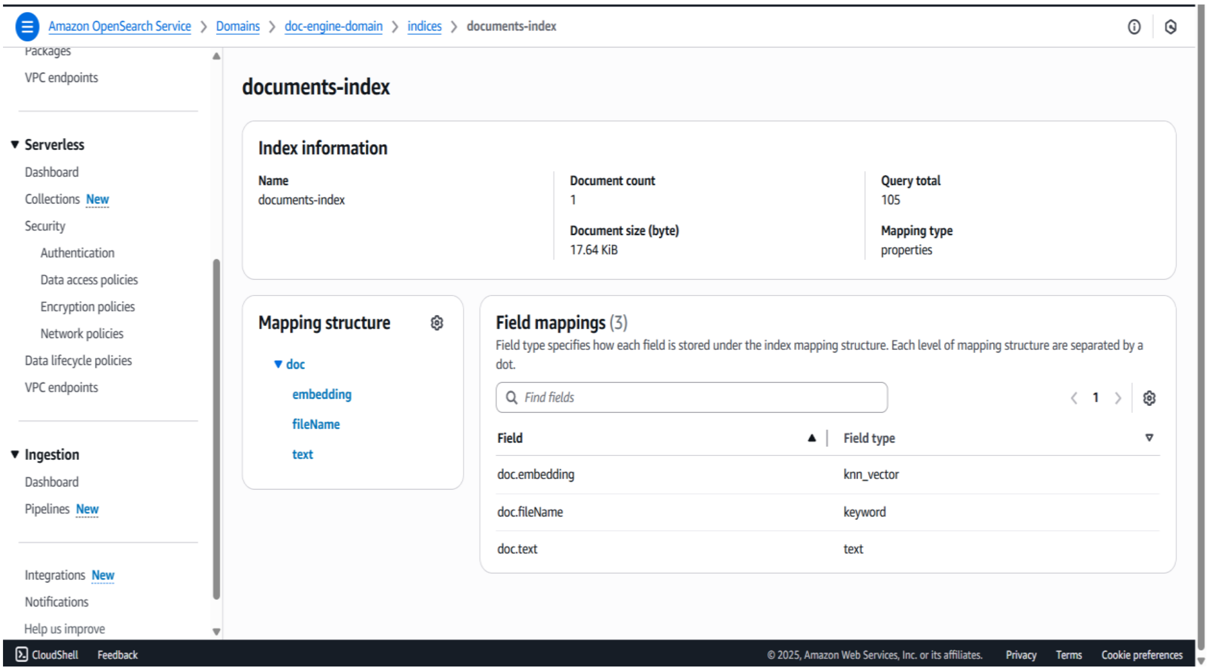


3. Embeddings + metadata are stored in **OpenSearch (KNN Index)** for fast semantic retrieval.

- Output of generated embeddings.



- Indexing confirmation on OpenSearch dashboard.



OpenSearch Vector Database — How It Works

OpenSearch's KNN indexing allows you to:

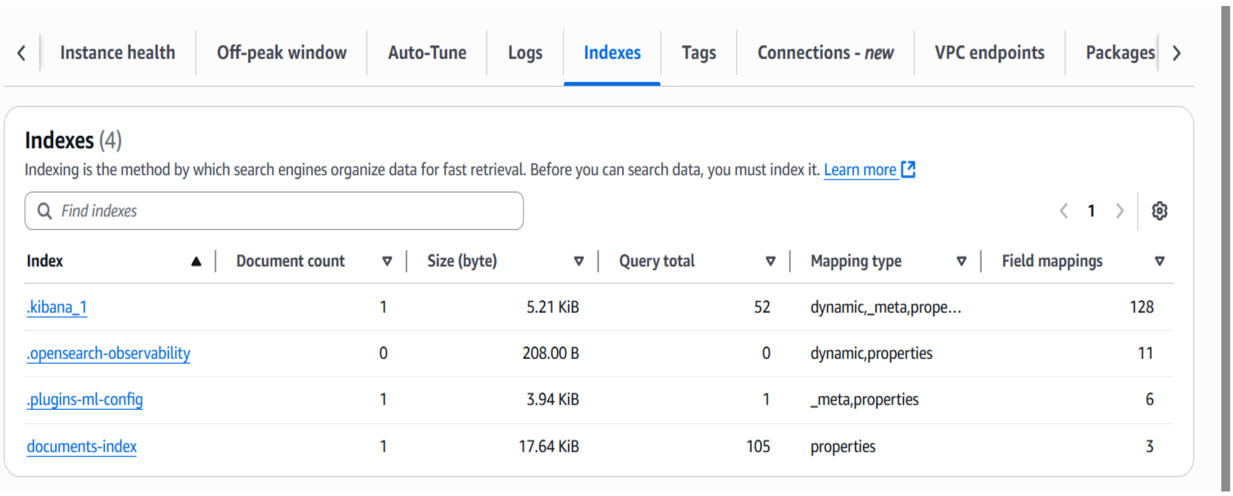
- Store **high-dimensional embedding vectors** (from SageMaker).
- Perform **Approximate Nearest Neighbour Search** on these vectors.
- Retrieve **most semantically relevant document** based on the user's question.

Example:

Query: *"What is the net profit?"*
OpenSearch returns the document that has the closest vector embedding to this question and pulls the matching line.

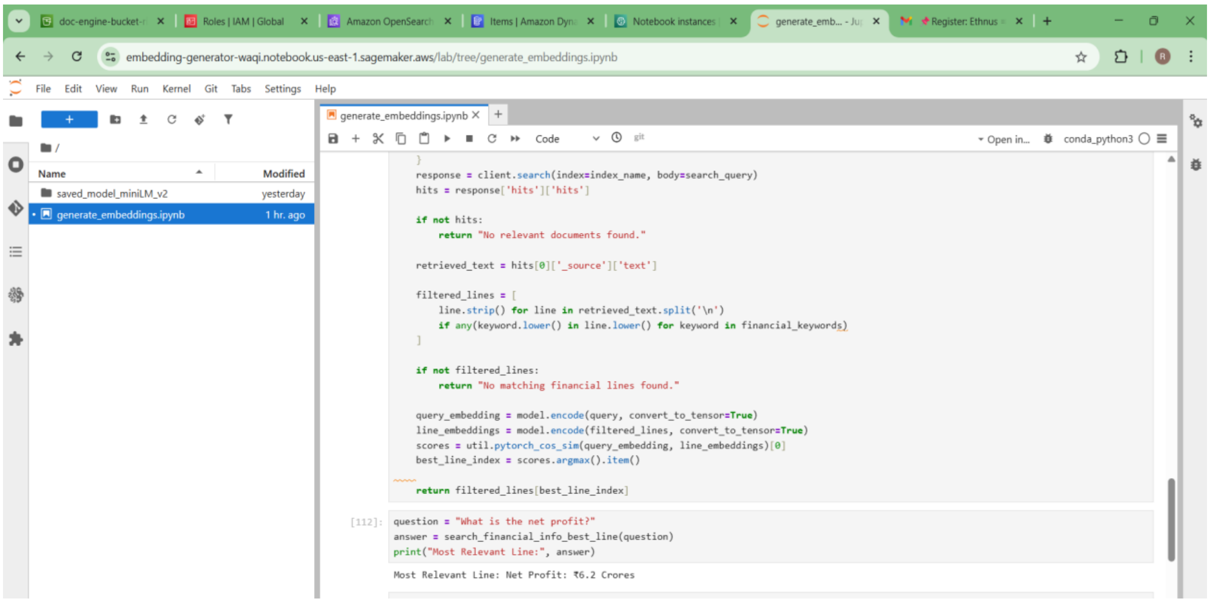
Advantage over keyword search:

- Can handle **paraphrased queries** (e.g., "profit for the quarter" → "Net profit recorded: $1.25 million").
- Robust even when documents use different wordings.
- OpenSearch index view.



Index	Document count	Size (byte)	Query total	Mapping type	Field mappings
.kibana_1	1	5.21 KiB	52	dynamic,_meta,prope...	128
.opensearch-observability	0	208.00 B	0	dynamic,properties	11
.plugins-ml-config	1	3.94 KiB	1	_meta,properties	6
documents-index	1	17.64 KiB	105	properties	3

- Search query payload and matching result.

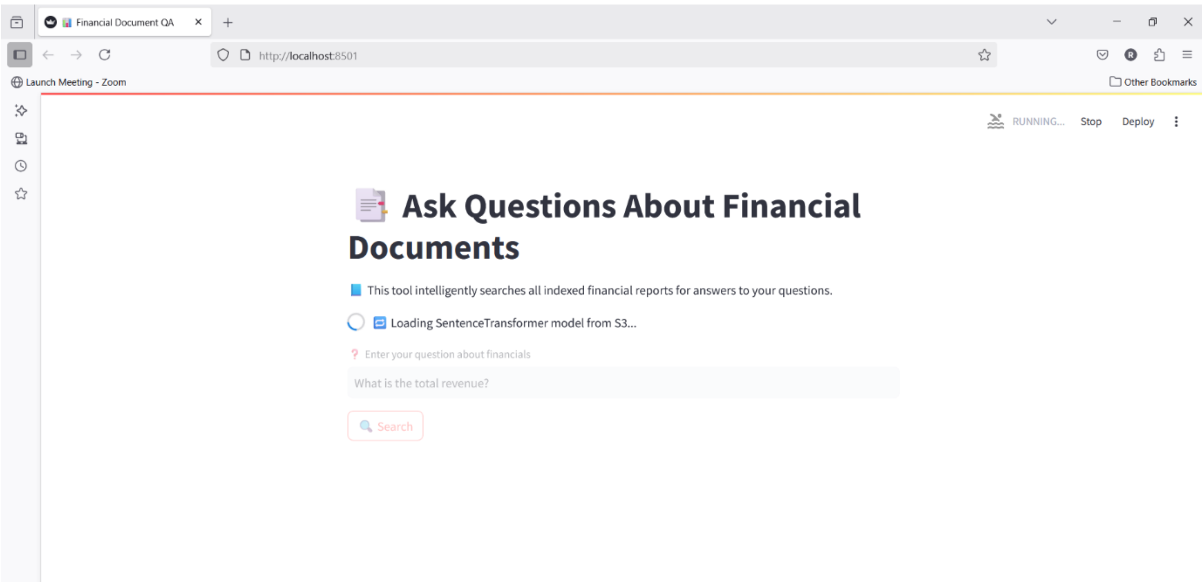


Streamlit Frontend App

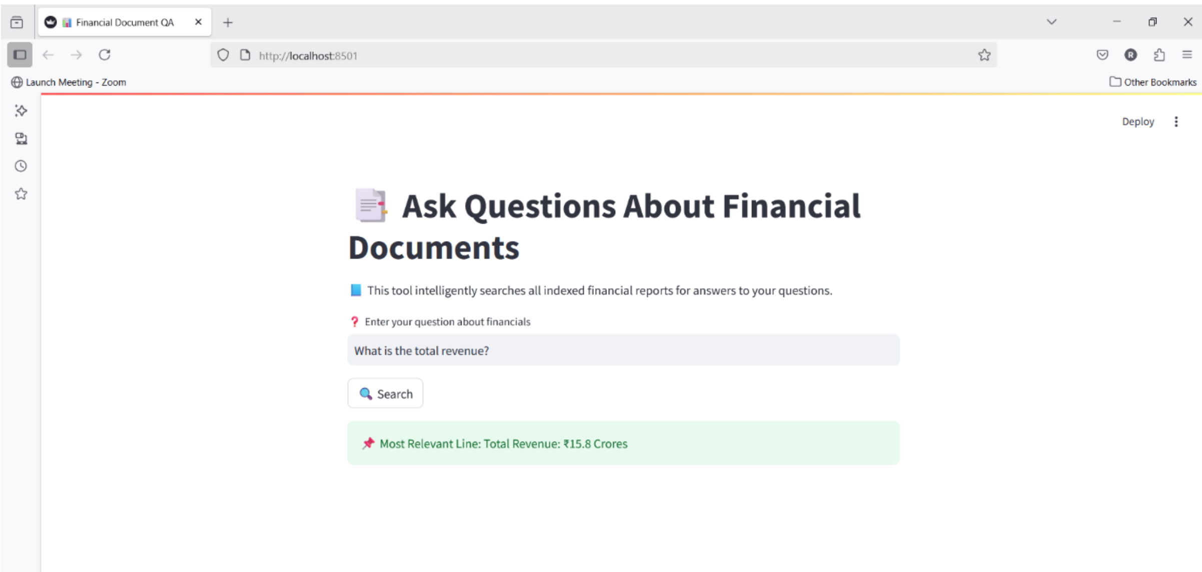
Functionality:

- Web UI for users to **ask questions** about uploaded financial documents.
- Loads the model dynamically from **S3**.
- Searches **across all indexed documents** and returns the **most relevant answer line**.
- No document selection needed — all searches are **aggregated**.

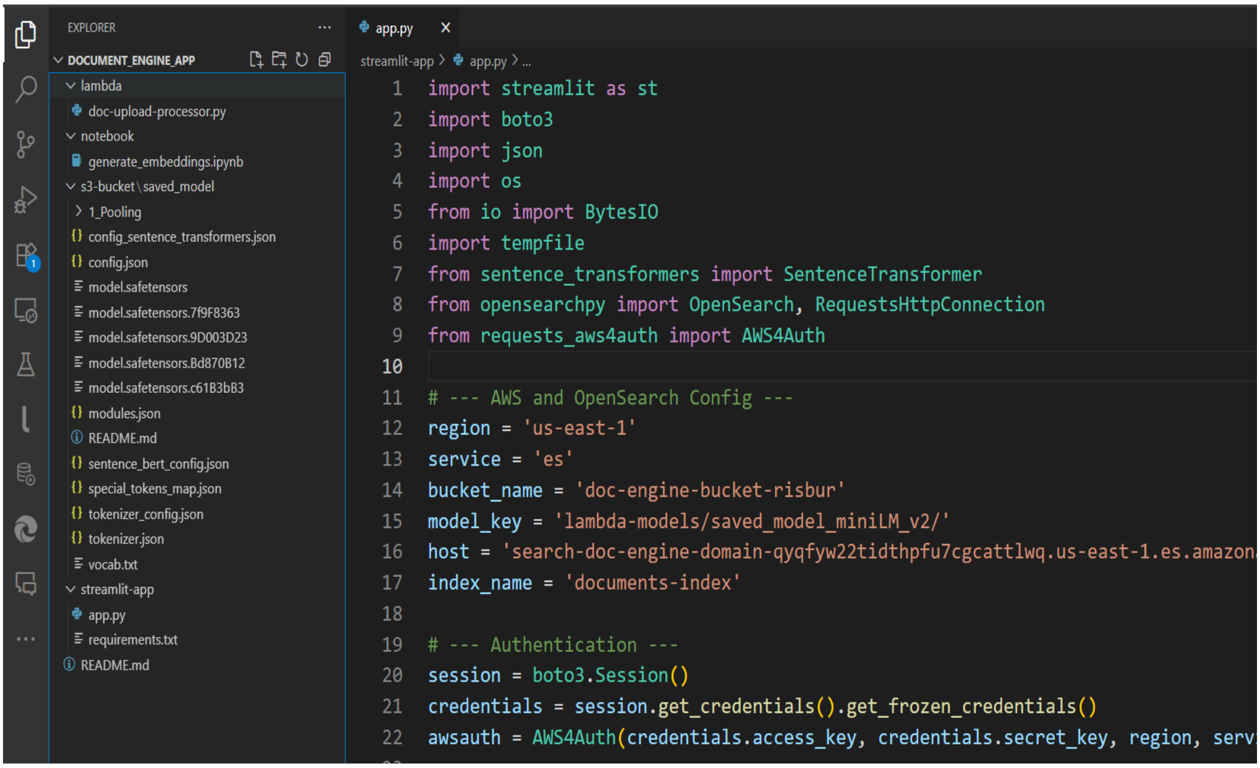
- Streamlit interface with user question.



- Output with the most relevant line highlighted.



Project Structure



Technologies Used

Component	Technology
Text Extraction	AWS Textract
Storage	S3, DynamoDB
Embeddings	SentenceTransformer (MiniLM)
Model Hosting	SageMaker
Vector Index	OpenSearch KNN
App Frontend	Streamlit
Deployment	EC2 t2.micro (Free Tier)

Component	Technology
Authentication	Boto3 + SigV4 (requests-aws4auth)

✅ Benefits Recap

- **Privacy First**: No data goes to third-party LLMs.
- **Fast Search**: Instant semantic answers from large documents.
- **Contextual Understanding**: Not just keyword matching.
- **Industry-Ready**: Applicable in finance, law, healthcare.

How to Run the Project

1. Upload Financial Docs to S3

S3 Bucket: doc-engine-bucket-risbur
Trigger Lambda: doc-upload-processor
Stores in: DocumentTextTable

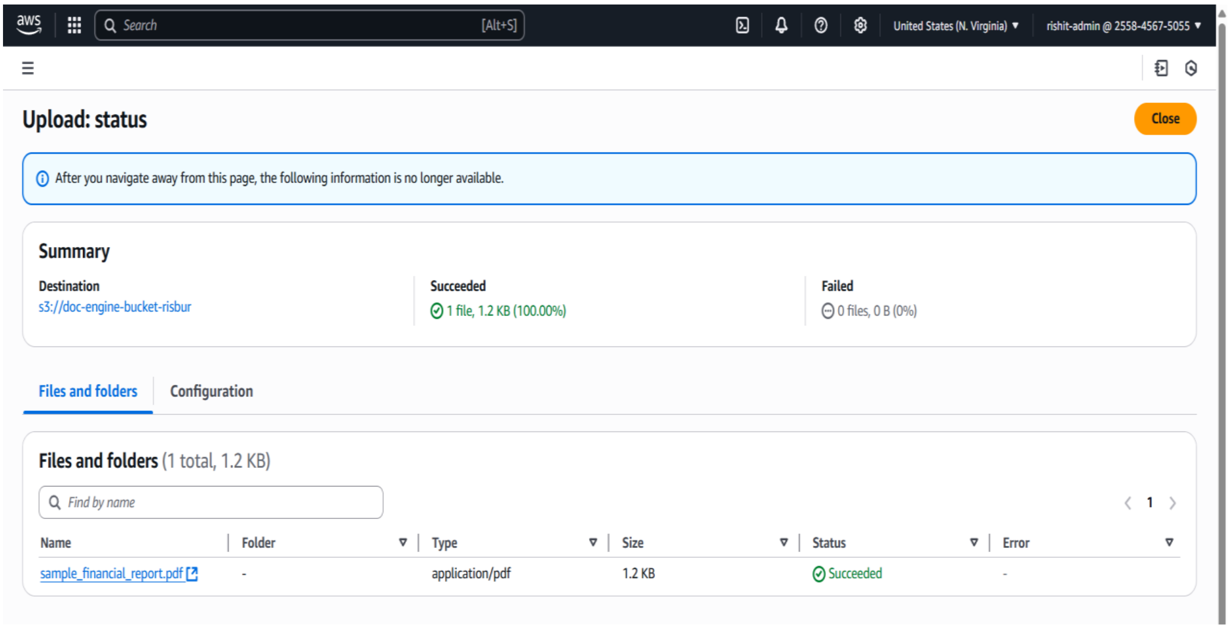
2. Run generate_embeddings.ipynb

Generates vector embeddings & stores in OpenSearch index.

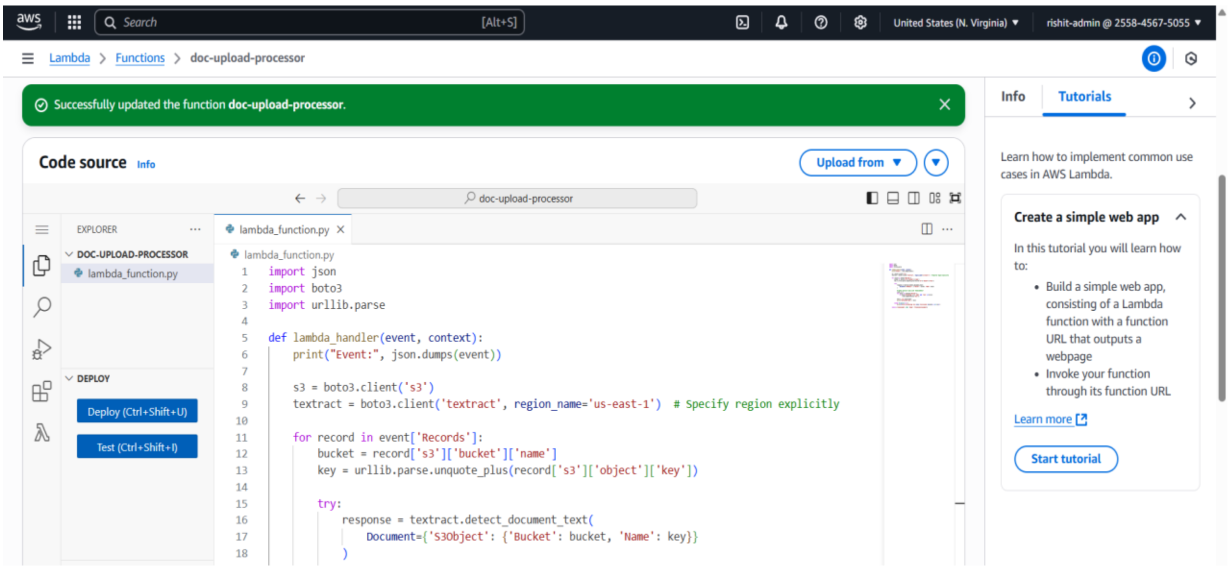
3. Launch Streamlit UI

streamlit run app.py

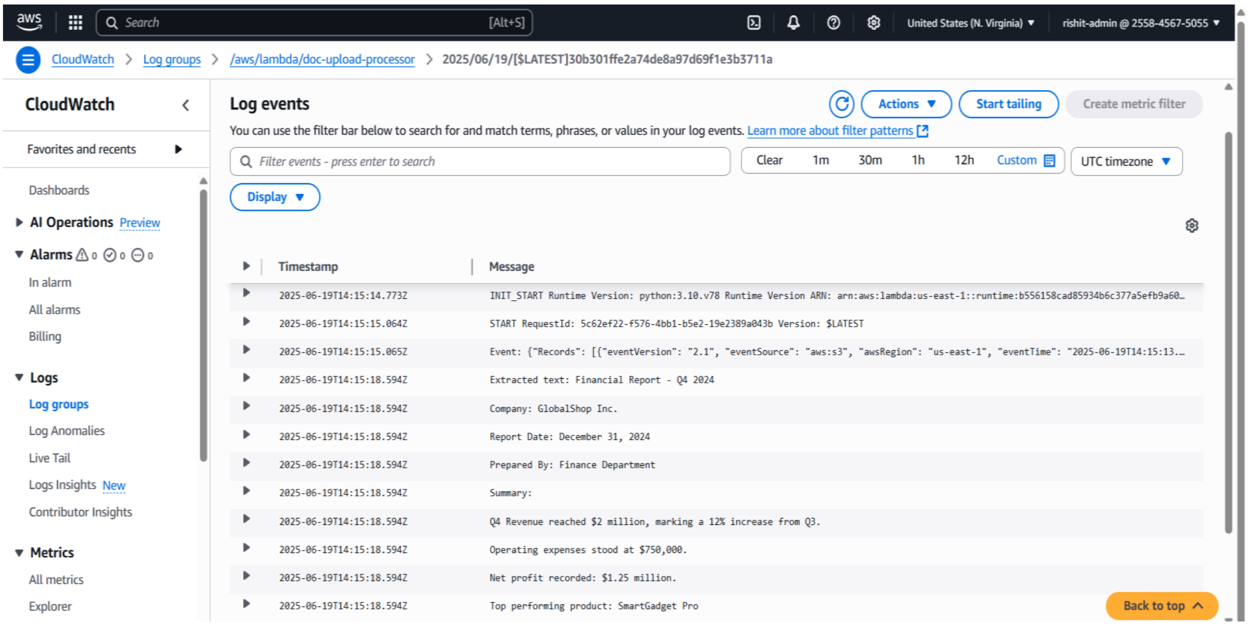
- ☑ S3 document upload interface.



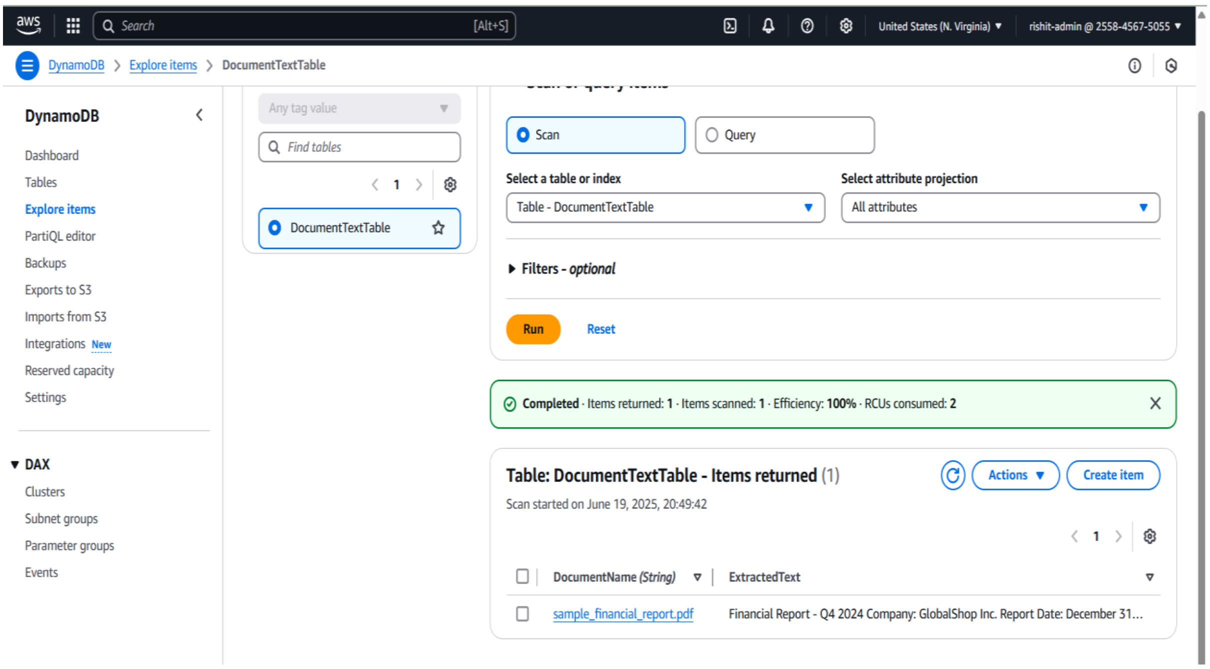
- ☑ Lambda function deployment



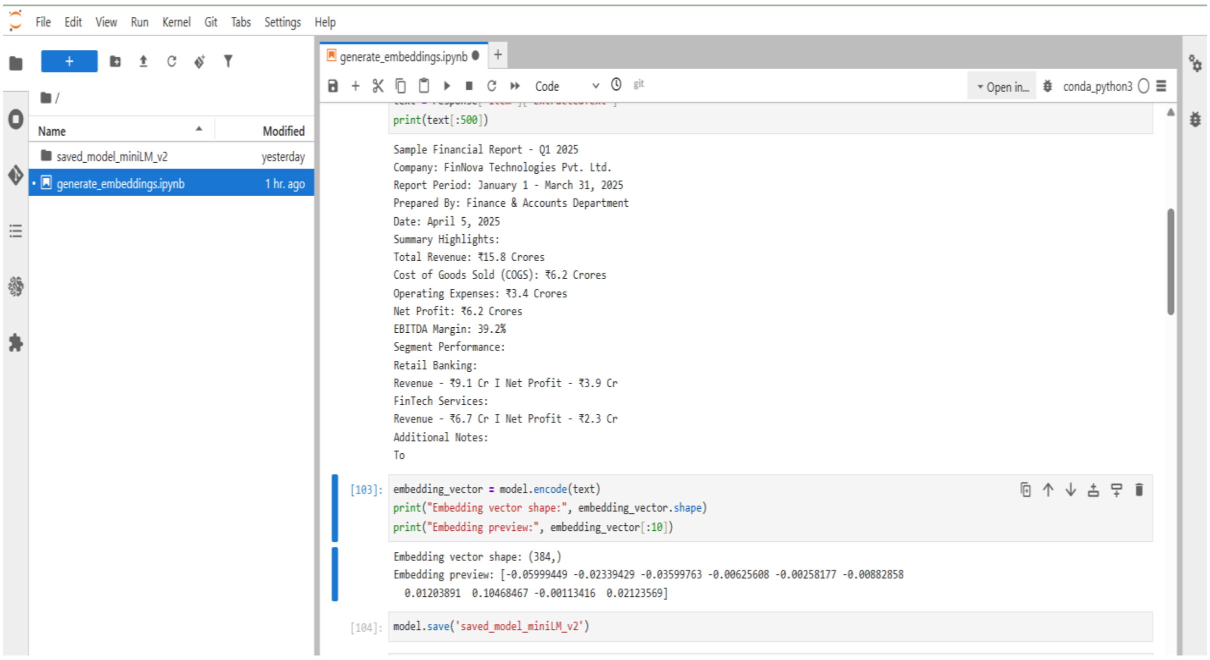
- ☑ Lambda execution log showing Textract.



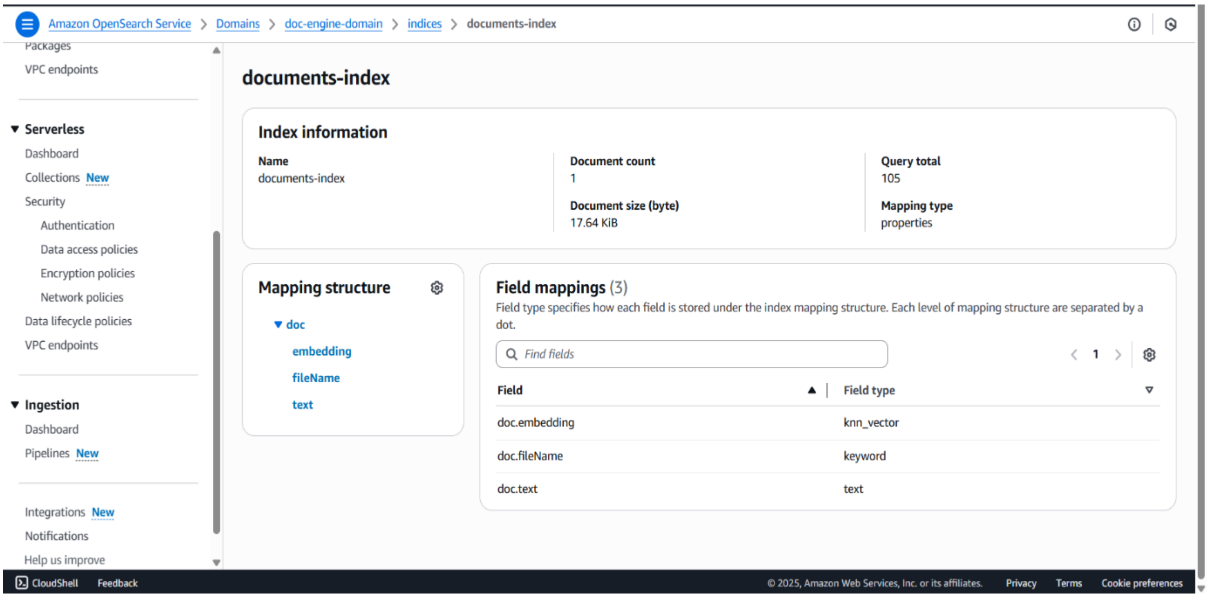
- ☑ DynamoDB console showing text records.



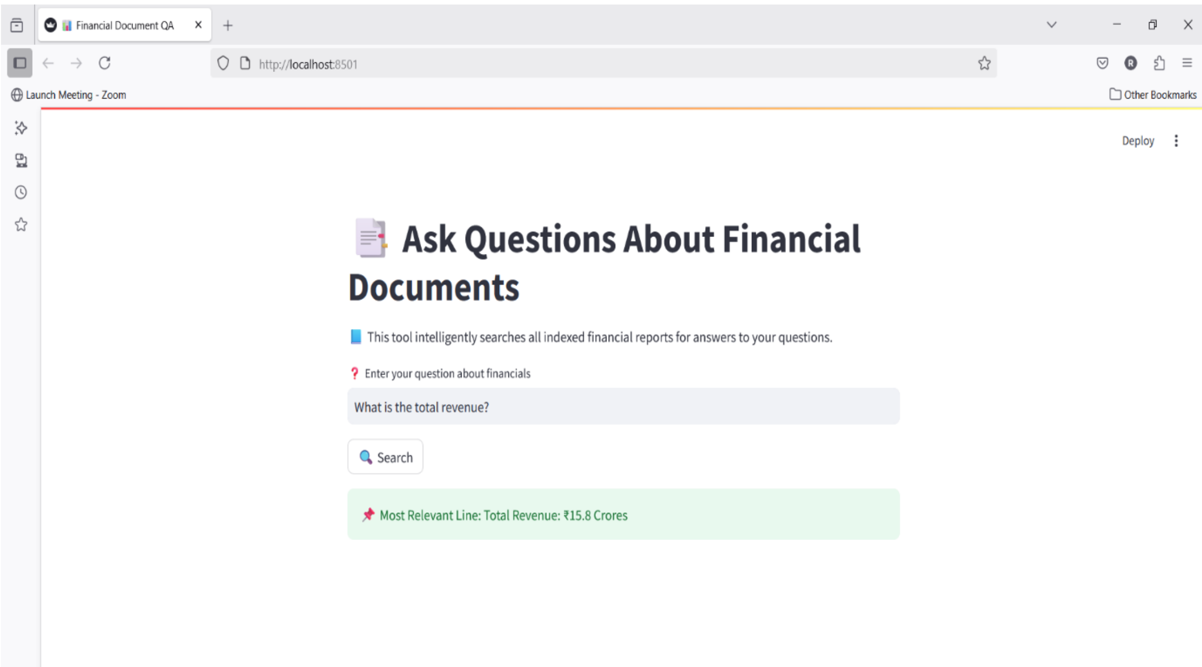
- ☑ SageMaker embedding generation outputs.



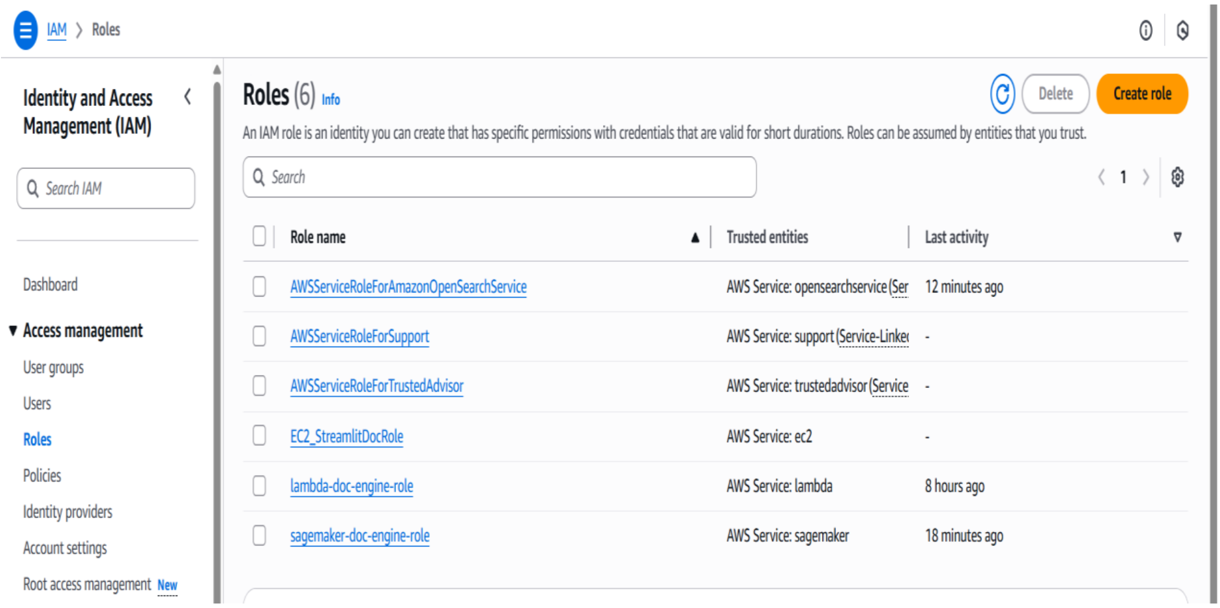
- ☑ OpenSearch dashboard (index + knn).



- ☑ Streamlit UI answering questions.



- ☑ IAM Roles configuration



Hosting the Streamlit App on AWS EC2

To make the application accessible publicly and ensure scalability, I deployed the Streamlit-based Q&A interface on an **Amazon EC2 instance** using a **Linux AMI (Amazon Machine Image)** with the **Free Tier t2.micro** configuration.

EC2 Configuration Details:

Configuration	Value
AMI	Amazon Linux 2
Instance Type	t2.micro (Free Tier eligible)
Storage	8 GB EBS (General Purpose SSD)
Security Group	Allowed inbound traffic on ports **8501** (Streamlit), **22** (SSH), and optionally **80** for HTTP
Key Pair	Generated and used for SSH access securely
Public IP Assignment	Enabled (auto-assigned)

App Deployment Steps on EC2

Here's how I deployed the Streamlit application:

1. **SSH into the EC2 instance:**

   ```
   ssh -i my-key.pem ec2-user@your-ec2-public-ip
   ```

2. **Installed required system packages:**

   ```
   sudo yum update -y
   ```

```
sudo yum install python3 git -y
```

3. **Created a virtual environment:**

```
python3 -m venv venv
source venv/bin/activate
```

4. **Cloned the GitHub repo containing my Streamlit app:**

```
git clone https://github.com/rishit911/document_engine_proj.git
```

5. **Installed Python dependencies:**

```
pip install -r requirements.txt
```

6. **Configured AWS credentials using aws configure** to authenticate with SageMaker, OpenSearch, and S3.

7. **Ran the Streamlit application:**

```
streamlit run app.py --server.port 8501 --server.enableCORS false
```

8. **Kept the Streamlit app running persistently using nohup:**

```
nohup streamlit run app.py &
```

9. **Accessed the app via browser using:**

   http://3.110.245.87:8501

Deployment Observations (EC2 Hosting Experience)

After developing and testing the complete system locally, I deployed the Streamlit app to an AWS EC2 instance for public access. This phase added a new layer of complexity and learning.

EC2 Instance Configuration

- **Instance Type: t2.micro (AWS Free Tier)**
- **AMI: Amazon Linux 2023 AMI**

- **Storage Volume: Initially 8 GiB (default)**
- **Security Group: Opened inbound rules for port 8501 (Streamlit) and 22 (SSH)**
- **Dependencies Installed:**
    - **Python 3.10**
    - **streamlit, boto3, opensearch-py, sentence-transformers, requests-aws4auth**
    - **Model downloaded from S3 inside EC2 at runtime**

Issue #1: Cold Start Latency

On each EC2 boot, downloading the SentenceTransformer model from S3, initializing the OpenSearch client with SigV4, and loading dependencies led to a cold start delay (30–50 seconds). This was acceptable for light usage, but for faster production-grade deployments, the following improvements are recommended:

- **Cache model in EBS volume between reboots**
- **Containerize and pre-load using Docker on boot**
- **Use AWS Lambda with provisioned concurrency (if architecture is serverless)**

Issue #2: Default Volume Too Small

When installing large packages like sentence-transformers, torch, and transformers, I encountered No space left on device errors due to the default 8 GiB volume limit.

✅ **Resolution:**

- **Stopped EC2**
- **Went to Volumes > Actions > Modify Volume**
- **Increased size to 20 GiB**
- **Rebooted EC2 and confirmed new size with df -h**

Personal Learnings & Project Reflections

What I Learned

Working on this project was a deep dive into real-world **cloud engineering**, **MLOps**, and **AI automation**. It helped me:

- ✅ Understand **end-to-end data pipelines** from ingestion (S3) to processing (Lambda & Textract) to storage (DynamoDB) and retrieval (OpenSearch).
- ✅ Gain hands-on experience with **semantic search** using vector databases and **SageMaker** model serving.
- ✅ Learn to orchestrate multiple AWS services securely using **IAM roles**, **boto3**, and **SigV4 authentication**.
- ✅ Improve my debugging and deployment skills — from local dev on Streamlit to hosting on **EC2 (free-tier t2.micro)**.
- ✅ Structure and document a scalable, modular, and production-grade cloud-native application.

Mistakes I Made (and Fixed)

- ❌ **Initial Misconfigurations** in OpenSearch domain endpoint caused timeout errors — I learned how to use correct hostname syntax and adjusted connection retries.
- ❌ I tried to **load heavy ML models inside Lambda**, which exceeded size limits. I pivoted to **SageMaker inference endpoints** for scalability.
- ❌ **Model saving/loading errors** (due to meta tensors and PyTorch 2.x issues) taught me the value of version compatibility and lazy loading models from S3 dynamically.
- ❌ Initially added dropdown document selection in Streamlit — later realized **aggregating search across all documents** was more useful and user-friendly.

How This Project Prepares Me for a Cloud Engineer Role

- Showcases my **ability to integrate multiple AWS services** into a functional, cloud-native solution.
- Demonstrates my understanding of **event-driven architectures** using Lambda and Textract.
- Proves my skills in **deploying scalable apps on EC2**, including dependency handling, model loading, and real-time querying.
- Highlights **secure authentication practices** using AWS4Auth and boto3.Session.
- Combines **machine learning**, **data engineering**, and **cloud operations** in a unified solution.