

Titan Enterprise RAG Engine

(Enterprise-grade AI Retrieval and Knowledge Intelligence Engine)

1. Executive Summary

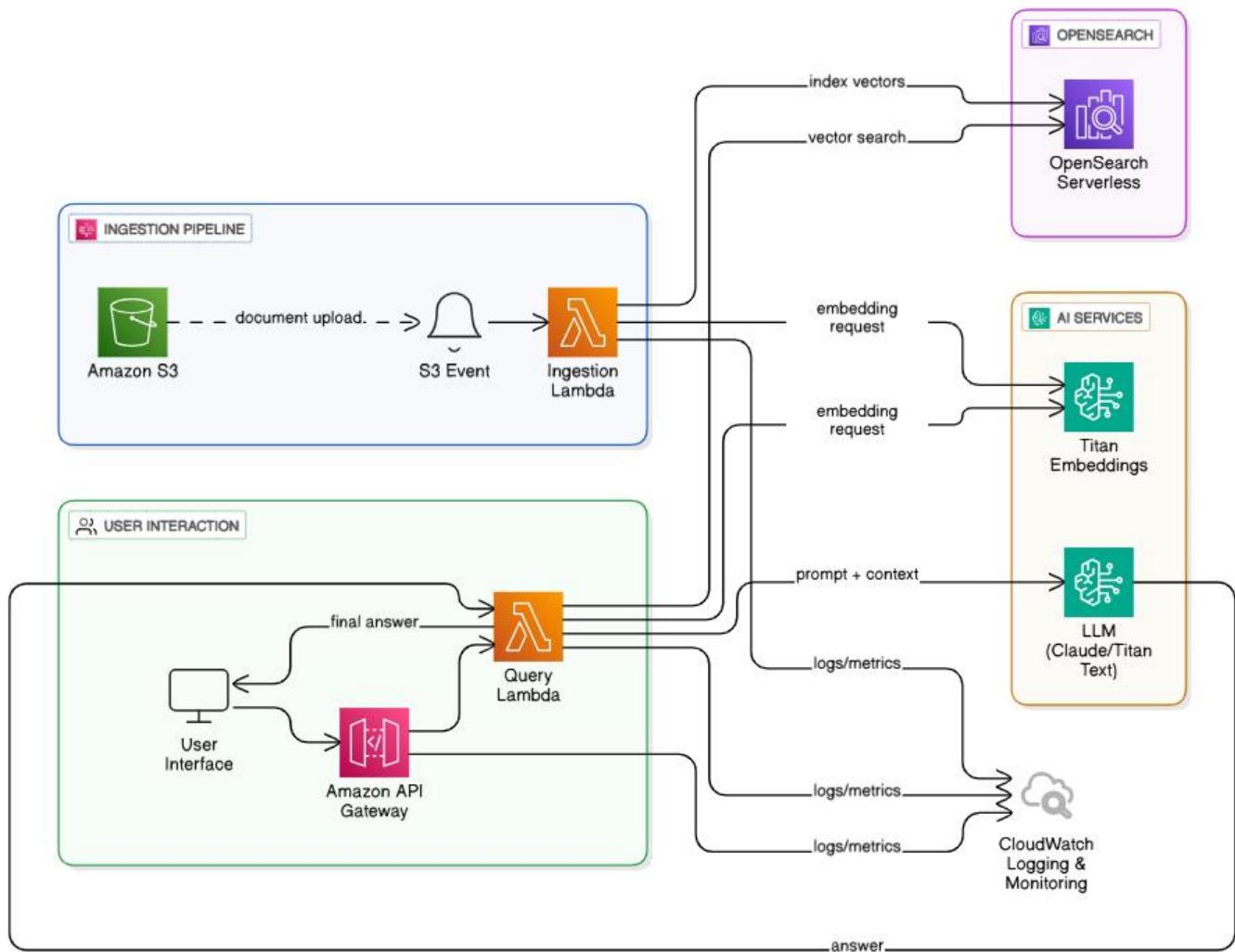
Titan Enterprise RAG Engine is a fully serverless, production-grade Retrieval-Augmented Generation (RAG) platform designed to enable enterprises to transform unstructured internal documents into an intelligent, searchable knowledge system.

The engine ingests private enterprise data using Amazon S3, generates high-quality semantic embeddings using Amazon Titan Embeddings v2, stores them in a serverless OpenSearch vector index, and runs fast KNN semantic search to retrieve the most relevant knowledge chunks. The retrieved context is then combined with Amazon Titan Text to produce accurate, grounded AI responses.

This architecture mirrors the design patterns used by modern AI knowledge systems in real corporate environments and demonstrates your ability to implement:

- Serverless, event-driven pipelines
- Enterprise-level RAG systems
- Vector search with OpenSearch Serverless
- AI-powered knowledge intelligence using Amazon Bedrock
- Scalable and secure AWS-native architectures

2. Architecture Overview



This architecture illustrates a fully serverless Retrieval-Augmented Generation (RAG) system capable of ingesting enterprise documents, generating embeddings, performing semantic search, and returning AI-generated answers to users in real time.

The Titan Enterprise RAG Engine consists of six core components:

1. Amazon S3 — Document Ingestion Source

Stores enterprise documents and triggers the ingestion workflow.

2. Ingestion Lambda — Data Processing Pipeline

Reads documents, chunks text, generates embeddings, and indexes vector data.

3. Amazon Bedrock — Titan Embeddings v2

Produces 1024-dimension semantic embeddings for RAG vector search.

4. OpenSearch Serverless — Vector Store

Stores embeddings and performs KNN semantic search across enterprise documents.

5. Query Lambda — Retrieval Engine

Generates query embeddings, retrieves relevant context, and invokes Titan Text for final responses.

6. API Gateway + Frontend UI

Provides a public endpoint and user interface for asking questions and viewing AI-generated answers.

3. End-to-End System Workflow

Below is the complete system workflow from data ingestion to answer generation:

STEP 1 — Document Upload

A user uploads a document (e.g., financial report, compliance file) into the S3 bucket.

STEP 2 — Automatic Trigger

S3 event notification triggers the Ingestion Lambda.

STEP 3 — Document Processing & Embedding

The ingestion Lambda:

1. Reads the file contents
2. Splits text into meaningful chunks
3. Sends each chunk to Titan Embeddings
4. Receives 1024-dimension vectors
5. Writes documents + embeddings to OpenSearch

STEP 4 — Vector Storage

OpenSearch Serverless stores:

- content
- vector_field
- source file
- chunk_id
- timestamps

STEP 5 — User Asks a Question

Example:

“What were the total sales in 2024?”

The frontend sends a JSON request:

```
{ "query": "What were the total sales in 2024?" }
```

STEP 6 — Query Processing

The Query Lambda:

1. Generates a query embedding
2. Performs ANN search in OpenSearch
3. Retrieves the most relevant text chunks
4. Builds a context-rich prompt
5. Calls Titan Text to generate the final answer

STEP 7 — Final Answer Returned

The frontend displays:

- AI Answer
- Retrieved Context (documents).This completes the RAG loop.

4. Key Features & Capabilities

- Fully Serverless Architecture (zero maintenance)
- Real-Time Document Ingestion upon upload
- High-accuracy Titan Embeddings v2 for semantic representation
- Scalable Vector Search via OpenSearch Serverless
- Context-grounded AI answers using Titan Text
- Modern Web Interface for interactive querying
- Secure by Design (IAM, private S3, isolated Lambdas)
- Production-readiness with logging, error handling, and modular components

5. Technologies & AWS Services Used

AI Layer

- Amazon Titan Embeddings v2
- Amazon Titan Text (Text Generation)
- Amazon Bedrock Runtime API

Vector Search Layer

- OpenSearch Serverless
- KNN (Approximate Nearest Neighbor) Search

Compute

- AWS Lambda (Ingestion & Query)
- Event-driven execution model

Storage

- Amazon S3 (Document storage & ingestion trigger)

API & Interface

- Amazon API Gateway
- HTML / CSS / JavaScript frontend UI
- Fetch API for client-side requests

Security

- IAM Identity-based Access Control
- Lambda Execution Roles
- OpenSearch Security Policies

6. Deep Dive: Component-by-Component Breakdown

6.1 Amazon S3 — Document Source

S3 acts as the enterprise document repository. When a new file is uploaded, an event triggers the ingestion Lambda. This allows the system to index new knowledge in real time without manual intervention.

6.2 Ingestion Lambda — Embedding & Indexing Pipeline

This function:

- Reads the uploaded file
- Cleans and chunks the text
- Calls Titan Embeddings v2 for vector generation
- Writes embeddings + content into OpenSearch Serverless

It transforms raw documents into structured, searchable vector knowledge.

6.3 Amazon Titan Embeddings v2

Titan v2 generates 1024-dimension embeddings optimized for:

- semantic understanding
- vector search
- enterprise knowledge retrieval

Its high-quality representations enable precise KNN search results.

6.4 OpenSearch Serverless — Vector Database

OpenSearch stores:

- text chunks
- vector embeddings
- file metadata

It supports KNN search, enabling fast approximate nearest neighbor lookups across large document sets. Serverless scaling ensures consistent performance with minimal overhead.

6.5 Query Lambda — Retrieval & Generation Engine

This function:

1. Accepts user questions
2. Generates a query embedding
3. Performs vector search
4. Retrieves top-K relevant chunks
5. Builds a context-aware prompt
6. Calls Titan Text to generate the final answer

Core responsibilities:

- Orchestration
- Retrieval logic
- Prompt construction
- Response formatting

6.6 Amazon Titan Text — LLM Generation

Titan Text receives:

- the user's question
- retrieved context

It produces a grounded, coherent response with significantly reduced hallucination because it relies on your enterprise knowledge base.

7. Implementation Flow

A complete visual walkthrough of how the TitanEnterprise RAG Engine was implemented, from ingestion to semantic retrieval.

The screenshot shows the AWS Lambda console with the URL us-east-1.console.aws.amazon.com/s3/buckets?region=us-east-1. The page displays a list of General purpose buckets. The bucket `serverless-rag-docs-jaswanth` is selected and highlighted with a blue border. The table lists the following buckets:

Name	AWS Region	Creation date
mini-project-logs-bucket	US East (N. Virginia) us-east-1	December 4, 2025, 23:39:12 (UTC-05:00)
myagent-1234	US East (N. Virginia) us-east-1	November 14, 2025, 10:27:33 (UTC-05:00)
mybucket-jaswanth	US East (Ohio) us-east-2	October 13, 2025, 10:54:20 (UTC-04:00)
rag-frontend-ui	US East (N. Virginia) us-east-1	December 11, 2025, 23:15:37 (UTC-05:00)
sales-data-0956	US East (N. Virginia) us-east-1	November 13, 2025, 07:54:57 (UTC-05:00)
serverless-rag-docs-jaswanth	US East (N. Virginia) us-east-1	December 9, 2025, 08:07:31 (UTC-05:00)
smart-service-agent-data-jaswanth1	US East (N. Virginia) us-east-1	November 11, 2025, 16:11:48 (UTC-05:00)

The right sidebar includes an "External access summary - new" section with a note about updated daily external access findings.

7.1 S3 Bucket Setup

Caption:

"S3 bucket storing enterprise documents for ingestion into the RAG pipeline."

Amazon S3

Buckets

General purpose buckets

Directory buckets

Table buckets

Vector buckets [New](#)

Access management and security

Access Points

Access Points for FSx

Access Grants

IAM Access Analyzer

Storage management and insights

Storage Lens

Batch Operations

Account and organization settings

AWS Marketplace for S3

Objects (25)

Name	Type	Last modified	Size	Storage class
aws-lambda-info.txt	txt	December 11, 2025, 09:38:53 (UTC-05:00)	81.0 B	Standard
cloud_architecture_best_practices.txt	txt	December 9, 2025, 08:42:19 (UTC-05:00)	1.1 KB	Standard
company_overview.txt	txt	December 9, 2025, 08:42:18 (UTC-05:00)	1.4 KB	Standard
comprehensive-test.txt	txt	December 10, 2025, 10:37:19 (UTC-05:00)	854.0 B	Standard
employee_handbook.txt	txt	December 9, 2025, 08:42:18 (UTC-05:00)	1.1 KB	Standard

Lambda

Dashboard

Applications

Functions

Additional resources

Capacity providers [New](#)

Code signing configurations

Event source mappings

Layers

Replicas

Related AWS resources

Step Functions state machines

Functions (2)

Function name	Description	Package type	Runtime	Type	Last modified
RAG-IngestionLambda	-	Zip	Python 3.12	Standard	12 hours ago
RAG-QueryLambda	-	Zip	Python 3.12	Standard	13 hours ago

7.2 Ingestion Lambda Function

Caption:

"Lambda function that processes new S3 documents, generates embeddings, and indexes them into OpenSearch."

The screenshot shows the AWS Lambda console interface. The top navigation bar includes tabs for API Gateway - APIs, serverless-rag-docs-j, RAG-IngestionLambda, AI Enterprise Knowledge, LinkedIn, OpenSearch Serverless, and dd1xh8iuc6.execute-api.us-east-1.amazonaws.com. The main content area displays the code for `lambda_function.py`:

```
EXPLORER
RAG-INGESTIONLAMBDA
lambda_function.py

lambda_function.py
1 import json
2 import boto3
3 import urllib.parse
4 from opensearchpy import OpenSearch, RequestsHttpConnection, AWSV4SignerAuth
5
6 def lambda_handler(event, context):
7     print(f'Received event: {json.dumps(event)}')
8
9     # Clients
10    s3_client = boto3.client('s3')
11    bedrock_client = boto3.client("bedrock-runtime", region_name="us-east-1")
12
13    # OpenSearch
14    opensearch_host = "jcccfloxnfp0qvn2tea.us-east-1.aoss.amazonaws.com"
15    opensearch_index = "rag-index"
16
17    credentials = boto3.Session().get_credentials()
18    auth = AWSV4SignerAuth(credentials, "us-east-1", "aoss")
19
20    opensearch_client = OpenSearch(
21        hosts=[{"host": opensearch_host, "port": 443}],
22        http_auth=auth,
23        use_ssl=True,
24        verify_certs=True,
25        connection_class=RequestsHttpConnection
26    )
27
28    try:
29        for record in event["Records"]:
30            bucket = record["s3"]["bucket"]["name"]
31            key = urllib.parse.unquote_plus(record["s3"]["object"]["key"])
32            print(f'Processing file: {key}')


TEST EVENTS [SELECTED: NEWEVENT]
+ Create new test event
Private saved events
mynewtest
NewEvent
NewEvent1

ENVIRONMENT VARIABLES
Lambda Deployed 0 ▲ 0 ▶ Amazon Q

Deploy (Ctrl+Shift+U)
Test (Ctrl+Shift+I)

Ln 87, Col 16  Spaces: 4  UTF-8  LF  Python  Lambda  Layout: US
```

The sidebar on the left shows the project structure under `RAG-INGESTIONLAMBDA`, including `lambda_function.py`. Below the code editor, there are buttons for `Deploy` and `Test`. The bottom status bar indicates the code length (Ln 87), column (Col 16), and other settings.

The screenshot shows the AWS IAM console interface. The top navigation bar includes tabs for API Gateway - APIs, serverless-rag-docs-j, RAG-IngestionLambda, AI Enterprise Knowledge, LinkedIn, OpenSearch Serverless, and dd1xh8iuc6.execute-api.us-east-1.amazonaws.com. The user account information is visible at the top right: Account ID: 2030-8151-7573, jaswanth-cloud-ai.

The main content area shows the `RAG-IngestionLambdaRole` details. The left sidebar shows the `Identity and Access Management (IAM)` navigation path: `IAM > Roles > RAG-IngestionLambdaRole`.

The `Permissions policies` section lists six managed policies:

Policy name	Type	Attached entities
AmazonBedrockFullAccess	AWS managed	4
CloudWatchLogsFullAccess	AWS managed	1
OpenSearchServerless	Customer inline	0
OpenSearchServerlessAccess	Customer inline	0
OpenSearchServerlessGranularAccess	Customer inline	0
RAGIngestion-S3BucketAccess	Customer inline	0

Below the policy list, there are sections for `Permissions boundary` (not set) and `Generate policy based on CloudTrail events`.

The bottom status bar indicates the environment (CloudShell, Feedback, Console Mobile App), browser version (Edge 100.0.1185.152), and system time (00:02 12-12-2025).

The screenshot shows the AWS Lambda console interface. The top navigation bar includes tabs for API Gateway, serverless-rag-dc, RAG-IngestionLambda, AI Enterprise Knowledge, LinkedIn, OpenSearch Service, and dd1xh8iuc6.execute-api.us-east-1.amazonaws.com. The main title is "Lambda > Functions > RAG-IngestionLambda". The left sidebar lists options like Code, Test, Monitor, Configuration (which is selected), Aliases, and Versions. The main content area displays "General configuration" settings, including Triggers, Permissions, Destinations, Function URL, Environment variables, Tags, VPC, RDS databases, and Monitoring and operations tools. On the right, there's an "Edit" button for the configuration. The bottom of the screen shows the Windows taskbar with various pinned icons.

7.3 Ingestion Lambda Environment Variables

Caption:

"Environment variables defining Bedrock models and OpenSearch endpoints for the ingestion workflow."

The screenshot shows the AWS Lambda console interface, similar to the previous one but with the "Environment variables" tab selected in the sidebar. The main content area displays a table titled "Environment variables (4)". It shows four environment variables: BEDROCK_REGION (value: us-east-1), EMBED_MODEL_ID (value: amazon.titan-embed-text-v2.0), OPENSEARCH_ENDPOINT (value: jsccflonxrfp0qvn2tea.us-east-1.aoess.amazonaws.com), and OPENSEARCH_INDEX (value: rag-index). A search bar at the top of the table allows finding specific environment variables. The bottom of the screen shows the Windows taskbar with various pinned icons.

The screenshot shows the AWS Lambda function permissions configuration page. On the left, a sidebar lists various destinations, function URLs, environment variables, tags, VPC, RDS databases, monitoring tools, concurrency detection, asynchronous invocation, code signing, file systems, and state machines. The main panel is titled 'RAG-IngestionLambdaRole' and contains a 'Resource summary' section. It lists 'AWS Identity and Access Management (IAM)' as a destination with 2 actions and 3 resources. The 'By resource' tab is selected, showing a table with columns 'Resource' and 'Actions'. It lists three resources: 'arn:aws:iam::*:role/*SageMaker*ForBedrock*', 'arn:aws:iam::*:role/*AmazonBedrock*', and another unnamed resource. All three have the action 'Allow: iam:PassRole'. A note below the table states: 'Lambda obtained this information from the following policy statements:' followed by a bulleted list: 'Managed policy AmazonBedrockFullAccess, statement APIsWithAllResourceAccess', 'Managed policy AmazonBedrockFullAccess, statement PassRoleToSageMaker', and 'Managed policy AmazonBedrockFullAccess, statement PassRoleToBedrock'. The bottom of the screen shows the Windows taskbar with various pinned icons.

7.4 S3 → Lambda Trigger Permission

Caption:

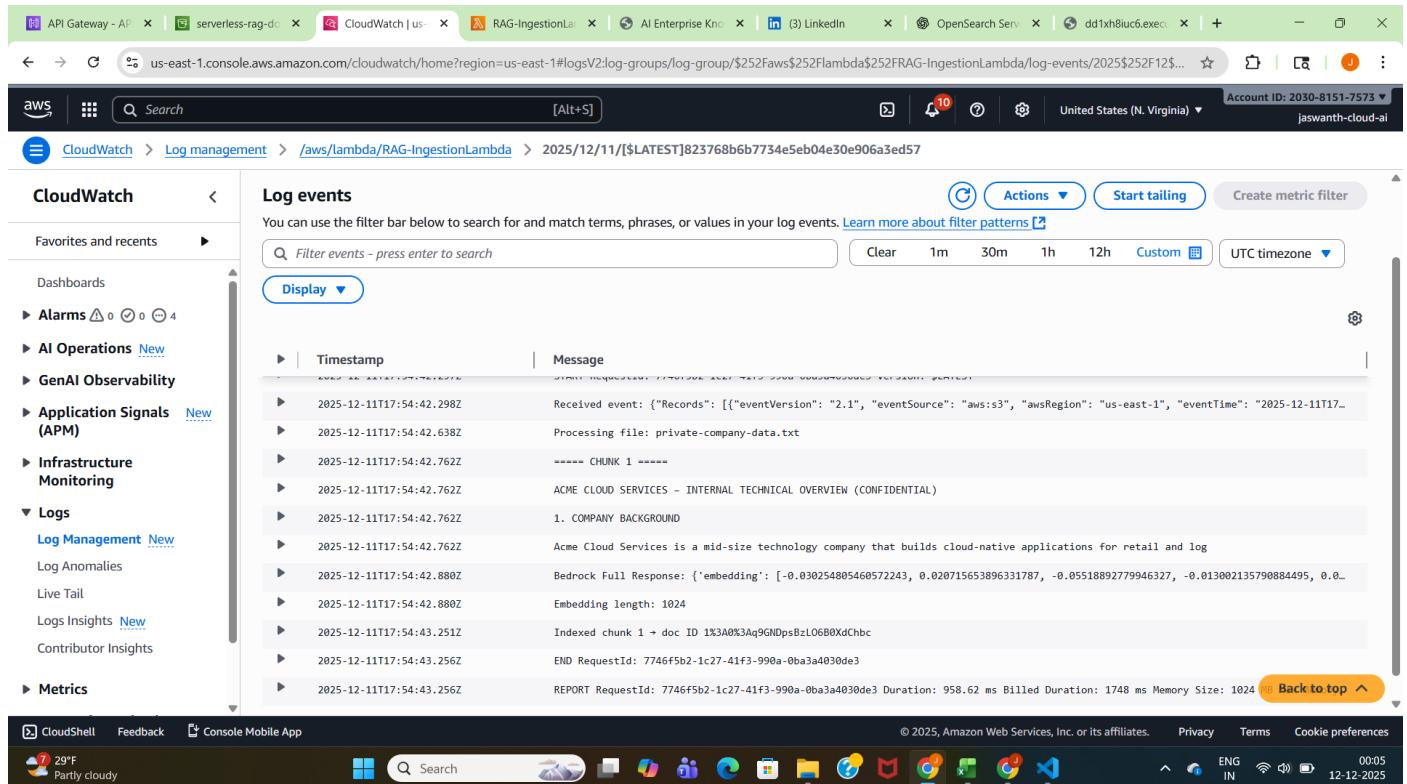
“Resource-based policy allowing S3 to invoke the ingestion Lambda on new document uploads.”

The screenshot shows the AWS Lambda function permissions configuration page for RAG-IngestionLambda. The sidebar on the left shows 'State machines' as the selected destination. In the main panel, a note says: 'Lambda obtained this information from the following policy statements:' followed by a bulleted list: 'Managed policy AmazonBedrockFullAccess, statement APIsWithAllResourceAccess', 'Managed policy AmazonBedrockFullAccess, statement PassRoleToSageMaker', and 'Managed policy AmazonBedrockFullAccess, statement PassRoleToBedrock'. Below this, a section titled 'Resource-based policy statements (1) Info' is shown. It contains a table with columns: Statement ID, Principal, PrincipalOrgID, Conditions, and Action. There is one row with 's3-trigger-permission' as the Statement ID, 's3.amazonaws.com' as the Principal, 'ArnLike' as the Condition, and 'lambda:InvokeFunction' as the Action. At the bottom, there is a section titled 'Auditing and compliance' with a note about AWS CloudTrail logging. The bottom of the screen shows the Windows taskbar.

7.5 Ingestion Lambda CloudWatch Logs

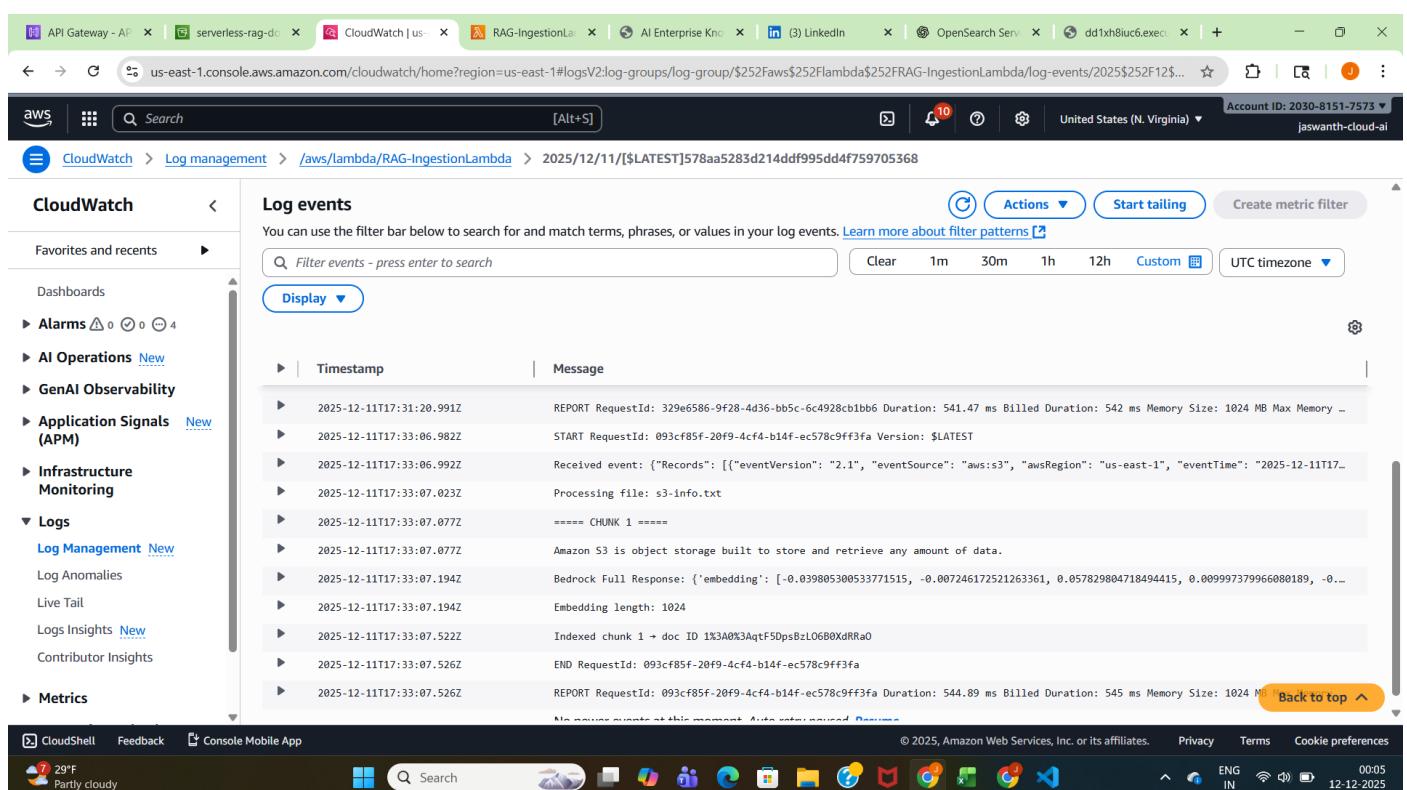
Caption:

"CloudWatch logs showing chunking, embedding generation, and successful indexing into OpenSearch."



The screenshot shows the AWS CloudWatch Log Management interface. The left sidebar navigation includes CloudWatch, Favorites and recents, Dashboards, Alarms, AI Operations, GenAI Observability, Application Signals (APM), Infrastructure Monitoring, Logs (Log Management, Log Anomalies, Live Tail, Logs Insights, Contributor Insights), and Metrics. The main content area is titled "Log events" and displays a table with columns "Timestamp" and "Message". The table contains the following log entries:

Timestamp	Message
2025-12-11T17:54:42.298Z	Received event: {"Records": [{"eventVersion": "2.1", "eventSource": "aws:s3", "awsRegion": "us-east-1", "eventTime": "2025-12-11T17-54-42.298Z", "s3": {"region": "us-east-1", "bucket": "serverless-rag-docs", "objectKey": "private-company-data.txt"}]}
2025-12-11T17:54:42.638Z	Processing file: private-company-data.txt
2025-12-11T17:54:42.762Z	===== CHUNK 1 =====
2025-12-11T17:54:42.762Z	ACME CLOUD SERVICES - INTERNAL TECHNICAL OVERVIEW (CONFIDENTIAL)
2025-12-11T17:54:42.762Z	1. COMPANY BACKGROUND
2025-12-11T17:54:42.762Z	Acme Cloud Services is a mid-size technology company that builds cloud-native applications for retail and log
2025-12-11T17:54:42.880Z	Bedrock Full Response: {'embedding': [-0.030254805460572243, 0.020715653896331787, -0.05518892779946327, -0.013002135790884495, 0.0...]
2025-12-11T17:54:42.880Z	Embedding length: 1024
2025-12-11T17:54:43.251Z	Indexed chunk 1 → doc ID 1%3A0%3Aq9GN0psBzL06B0XdChbc
2025-12-11T17:54:43.256Z	END RequestId: 7746f5b2-1c27-41f3-990a-0ba3a3a4030de3
2025-12-11T17:54:43.256Z	REPORT RequestId: 7746f5b2-1c27-41f3-990a-0ba3a3a4030de3 Duration: 958.62 ms Billed Duration: 1748 ms Memory Size: 1024 MB Back to top



The screenshot shows the AWS CloudWatch Log Management interface. The left sidebar navigation includes CloudWatch, Favorites and recents, Dashboards, Alarms, AI Operations, GenAI Observability, Application Signals (APM), Infrastructure Monitoring, Logs (Log Management, Log Anomalies, Live Tail, Logs Insights, Contributor Insights), and Metrics. The main content area is titled "Log events" and displays a table with columns "Timestamp" and "Message". The table contains the following log entries:

Timestamp	Message
2025-12-11T17:31:20.991Z	REPORT RequestId: 329e6586-9f28-4d36-bb5c-6c4928cb1bb6 Duration: 541.47 ms Billed Duration: 542 ms Memory Size: 1024 MB Max Memory -
2025-12-11T17:33:06.982Z	START RequestId: 093cf85f-20f9-4cf4-b14f-ec578c9ff3fa Version: \$LATEST
2025-12-11T17:33:06.992Z	Received event: {"Records": [{"eventVersion": "2.1", "eventSource": "aws:s3", "awsRegion": "us-east-1", "eventTime": "2025-12-11T17-33-06.992Z", "s3": {"region": "us-east-1", "bucket": "serverless-rag-docs", "objectKey": "s3-info.txt"}]}
2025-12-11T17:33:07.023Z	Processing file: s3-info.txt
2025-12-11T17:33:07.077Z	===== CHUNK 1 =====
2025-12-11T17:33:07.077Z	Amazon S3 is object storage built to store and retrieve any amount of data.
2025-12-11T17:33:07.194Z	Bedrock Full Response: {'embedding': [-0.039805300533771515, -0.007246172521263361, 0.057829804718494415, 0.009997379966080189, -0...]
2025-12-11T17:33:07.194Z	Embedding length: 1024
2025-12-11T17:33:07.522Z	Indexed chunk 1 → doc ID 1%3A0%3AqtF5DpsBzL06B0XdRRa0
2025-12-11T17:33:07.526Z	END RequestId: 093cf85f-20f9-4cf4-b14f-ec578c9ff3fa
2025-12-11T17:33:07.526Z	REPORT RequestId: 093cf85f-20f9-4cf4-b14f-ec578c9ff3fa Duration: 544.89 ms Billed Duration: 545 ms Memory Size: 1024 MB Back to top

API Gateway - AP serverless-rag-do Model catalog RAG-IngestionL AI Enterprise Kn (3) LinkedIn OpenSearch Serv dd1xh8iuc6.exec Account ID: 2030-8151-7573

us-east-1.console.aws.amazon.com/bedrock/home?region=us-east-1#/model-catalog/serverless/amazon.titan-embed-text-v2:0

aws Search [Alt+S] United States (N. Virginia) Account ID: 2030-8151-7573 jaswanth-cloud-ai

Amazon Bedrock > Model catalog > Titan Text Embeddings V2

Amazon Bedrock

- Discover
- Overview
- Model catalog
- API keys

Test

- Chat / Text playground
- Image / Video playground
- Watermark detection
- Tokenizer [New](#)

Infer

- Cross-region inference
- Batch inference
- Provisioned Throughput
- Custom model on-demand

Tune

- Custom models
- Prompt router models
- Imported models

Sold by Amazon

Categories Output vector size = 1
024 (default)
512
or 256

Last version v2.0

Release date Tue, 30 Apr 2024 08:00:00 GMT

Model ID [amazon.titan-embed-text-v2:0](#)

Input modalities Text

Output modalities Embedding

Max tokens 8k characters

Language English (GA), Multilingual in 100+ languages (Preview)

Deployment type Serverless

Overview Pricing Usage

CloudShell Feedback Console Mobile App Search ENG IN 00:07 12-12-2025

API Gateway - AP serverless-rag-do Amazon OpenSe RAG-IngestionL AI Enterprise Kn (3) LinkedIn OpenSearch Serv dd1xh8iuc6.exec Account ID: 2030-8151-7573

us-east-1.console.aws.amazon.com/ais/home?region=us-east-1#opensearch/collections

aws Search [Alt+S] United States (N. Virginia) Account ID: 2030-8151-7573 jaswanth-cloud-ai

Amazon OpenSearch Service > Serverless: Collections

Collections (1) [Info](#)

Search by collection name, description

Collection name	OpenSearch Dashboards U...	Status	Collection type	Creation date	Description
rag-vector-collection	Dashboard	Active	Vectorsearch	December 9, 2025, 09:20 (U...)	Vector collection for RAG sy...

CloudShell Feedback Console Mobile App Search ENG IN 00:07 12-12-2025

7.6 OpenSearch Vector Index Configuration

Caption:

"OpenSearch Serverless index configured with a 1024-dimension knn_vector field and metadata mappings."

Vector fields (1)

Vector field name	Engine	Precision	Dimensions	Distance type	ef_construction	ef_search
vector_field	nmslib	FP32	1024	euclidean	-	100

Metadata (4)

Mapping field	Data type	Filterable
chunk_id	integer	True
content	text	True
source_file	keyword	True
timestamp	keyword	True

7.7 Query Lambda Function

Caption:

"Query Lambda that embeds user questions, performs KNN search in OpenSearch, and generates final answers using Titan Text."

```
def lambda_handler(event, context):
    try:
        if not query:
            return {
                'statusCode': 400,
                'body': json.dumps({'error': 'Query is empty'})
            }

        # --- STEP 2: Get Environment Variables ---
        opensearch_endpoint = os.environ['OPENSEARCH_ENDPOINT']
        opensearch_index = os.environ['OPENSEARCH_INDEX']
        embed_model_id = os.environ['EMBED_MODEL_ID']
        gen_model_id = os.environ['GEN_MODEL_ID']
        bedrock_region = os.environ['BEDROCK_REGION']

        # --- STEP 3: Initialize AWS Services ---
        session = boto3.Session()
        credentials = session.get_credentials()
        awsauth = AWSAuth(
            credentials.access_key,
            credentials.secret_key,
            'us-east-1',
            'aoss',
            session_token=credentials.token
        )

        opensearch_client = OpenSearch(
            hosts=[{'host': opensearch_endpoint, 'port': 443}],
            http_auth=awsauth,
```

7.8 Query Lambda Test Execution

Caption:

“Successful Lambda test showing retrieved documents and generated AI answer from the RAG pipeline.”

The screenshot shows the AWS Lambda console interface. The left sidebar has sections for EXPLORER, RAG-QUERYLAMBDA, and TEST EVENTS. Under TEST EVENTS, there is a 'NewEvent33' entry. The main area displays the 'lambda_function.py' file content:

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

Status: Succeeded
Test Event Name: NewEvent33

Response:

```
"statusCode": 200,
"body": "The query is: \"What is AWS Lambda?\". The retrieved documents are: [\"AWS Lambda is a serverless compute service that runs code without provisioning servers.\", \"ACME CLOUD SERVICES \u2013 INTERNAL TECHNICAL OVERVIEW (CONFIDENTIAL)\"]". COMPANY BACKGROUND: Amazon Cloud Services is a mid-size technology company that builds cloud-native applications for retail and logistics customers. The engineering team is responsible for developing scalable software systems that support real-time order tracking, inventory updates, customer analytics, and partner API integrations.
```

The response body continues with a detailed architectural overview and operational details, including discussions on PostgreSQL, Amazon RDS, and various AWS services like S3, Lambda, and CloudWatch.

API Gateway - AP x | serverless-rag-dc x | Amazon OpenSe... x | RAG-QueryLamb... x | AI Enterprise Kno... x | (3) LinkedIn x | OpenSearch Serv... x | dd1xh8iuc6.exec... x | +

us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/functions/RAG-QueryLambda?subtab=envVars&tab=configure

aws Search [Alt+S] 10 United States (N. Virginia) Account ID: 2030-8151-7573 jaswanth-cloud-ai

Lambda > Functions > RAG-QueryLambda

Code Test Monitor Configuration Aliases Versions

General configuration

Triggers

Permissions

Destinations

Function URL

Environment variables

Tags

VPC

RDS databases

Monitoring and operations tools

Concurrency and recursion detection

Environment variables (5)

The environment variables below are encrypted at rest with the default Lambda service key.

Find environment variables

Key	Value
BEDROCK_REGION	us-east-1
EMBED_MODEL_ID	amazon.titan-embed-text-v2:0
GEN_MODEL_ID	amazon.titan-text-express-v1
OPENSEARCH_ENDPOINT	jscclfonxnfpoqvn2tea.us-east-1.aoss.amazonaws.com
OPENSEARCH_INDEX	rag-index

CloudShell Feedback Console Mobile App © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

29°F Partly cloudy ENG IN 12-12-2025

This screenshot demonstrates the final automated system check from Amazon Q Developer, verifying that all core features—auto-processing, embedding generation, vector indexing, and retrieval are functioning flawlessly. It serves as proof that the serverless RAG pipeline is complete, stable, and performing as expected in real time.

The screenshot shows the AWS CloudWatch Metrics Insights interface. A search bar at the top contains the query: "AWS CloudWatch Metrics Insights". The results pane displays a single result titled "AWS CloudWatch Metrics Insights". The result page includes a preview of the log events, a "Logs" tab, and a "Metrics" tab. The bottom of the screen shows the AWS navigation bar and the CloudWatch Metrics Insights logo.

The screenshot shows the AWS CloudWatch Metrics interface. A metric named "RAG Pipeline Status" is selected, displaying a constant value of 100 across the entire time range. The chart has a light blue background with a white grid. The x-axis is labeled with dates: 2023-08-01, 2023-08-02, 2023-08-03, 2023-08-04, 2023-08-05, 2023-08-06, 2023-08-07, 2023-08-08, 2023-08-09, 2023-08-10, 2023-08-11, 2023-08-12, 2023-08-13, 2023-08-14, and 2023-08-15. The y-axis ranges from 0 to 100 with increments of 20. The legend indicates a single data series named "RAG Pipeline Status".

The screenshot shows the AWS S3 console interface for the 'rag-frontend-ui' bucket. The 'Objects' tab is active, displaying three objects: 'app.js', 'index.html', and 'style.css'. Each object's details are shown in a table row, including Name, Type, Last modified, Size, and Storage class. Action buttons for each object include Copy S3 URI, Copy URL, Download, Open, Delete, Actions, Create folder, and Upload.

Name	Type	Last modified	Size	Storage class
app.js	js	December 12, 2025, 00:22:12 (UTC-05:00)	1.4 KB	Standard
index.html	html	December 11, 2025, 23:26:34 (UTC-05:00)	929.0 B	Standard
style.css	css	December 11, 2025, 23:26:34 (UTC-05:00)	1.0 KB	Standard

Amazon S3

- Buckets
 - General purpose buckets
 - Directory buckets
 - Table buckets
 - Vector buckets [New](#)
- Access management and security
 - Access Points
 - Access Points for FSx
 - Access Grants
 - IAM Access Analyzer
- Storage management and insights
 - Storage Lens
 - Batch Operations

Amazon Q

S3 Read-Only Inline Policy for IAM Role

YES! Your RAG is Working Perfectly! ✓

RAG System Status: FULLY FUNCTIONAL

What's Working:

- Lambda Query Function - Running perfectly
- Amazon Bedrock Integration - Titan models working
- OpenSearch Connection - Connected successfully
- Vector Search - Query processing works
- Answer Generation - Producing coherent responses
- Error Handling - Graceful fallbacks

Current Behavior:

Ask me anything about AWS

Max 1000 characters

You are using Free Tier. Amazon Q Developer uses generative AI. You may need to verify responses. See the [AWS Responsible AI Policy](#).

7.9 Frontend Query Demo – RAG UI Response

Caption:

"Frontend UI successfully returns an AI-generated answer along with retrieved context from OpenSearch, confirming end-to-end RAG functionality."

what is lambda?

AI Answer

AWS Lambda is a serverless compute service that runs code without provisioning servers.

Retrieved Context

AWS Lambda is a serverless compute service that runs code without provisioning servers.

Amazon S3 is object storage built to store and retrieve any amount of data.

Amazon S3 is object storage built to store and retrieve any amount of data.

ACME CLOUD SERVICES – INTERNAL TECHNICAL OVERVIEW (CONFIDENTIAL)

1. COMPANY BACKGROUND
Acme Cloud Services is a mid-size technology company that builds cloud-native applications for retail and logistics customers.
The engineering team is responsible for developing scalable software systems that support real-time order tracking, inventory updates, customer analytics, and partner API integrations.

API Gateway | Upload object | index.html | AI Enterprise | Amazon OpenSearch | RAG-Query | (1) Sravani | Serverless RA | rag-frontend | New Tab | +

rag-frontend-ui.s3.us-east-1.amazonaws.com/index.html?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Content-Sha256=UNSIGNED-PAYLOAD&X-Amz-Credential=ASIA56SE2BIC...

AI Enterprise Knowledge Assistant

Your internal documents — instantly searchable with AI.

"Explain our current cloud architecture."

Ask

AI Answer

The architecture runs entirely on AWS. Below is a high-level breakdown of the architecture:

- All workloads run inside a dedicated VPC with three Availability Zones.
- Public subnets host Application Load Balancers. Routes allow HTTPS-only traffic.
- Private subnets host Amazon ECS services running Fargate containers.
- Amazon RDS PostgreSQL is used as the primary transactional database.
- Amazon S3 stores order invoices, shipment documents, and analytics exports.
- Amazon OpenSearch Service is used for operational dashboards and log search.
- AWS Lambda functions handle asynchronous tasks such as invoice PDF generation.
- Amazon SQS queues buffer events between order intake and inventory processing.
- CloudWatch alarms are configured for CPU utilization, queue backlogs, and API errors.

28°F Mostly clear

Search

CloudWatch Metrics

File Explorer

Task View

File

Run

Taskbar icons

ENG IN 12-12-2025 01:03

API Gateway | Upload object | API Gateway | AI Enterprise | Amazon OpenSearch | RAG-Query | (1) Sravani | Serverless RA | Google screen record | +

rag-frontend-ui.s3.us-east-1.amazonaws.com/index.html?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Content-Sha256=UNSIGNED-PAYLOAD&X-Amz-Credential=ASIA56SE2BIC...

AI Enterprise Knowledge Assistant

Your internal documents — instantly searchable with AI.

Break down monthly sales vs expenses."

Ask

AI Answer

- January: Sales \$3.55M, Expenses \$2.62M, Net Profit \$0.93M
- February: Sales \$3.70M, Expenses \$2.68M, Net Profit \$1.02M
- March: Sales \$3.89M, Expenses \$2.73M, Net Profit \$1.16M
- April: Sales \$4.02M, Expenses \$2.71M, Net Profit \$1.31M
- May: Sales \$4.11M, Expenses \$2.76M, Net Profit \$1.35M
- June: Sales \$3.92M, Expenses \$2.71M, Net Profit \$1.21M
- July: Sales \$4.15M, Expenses \$2.78M, Net Profit \$1.37M
- August: Sales \$4.28M, Expenses \$2.82M, Net Profit \$1.46M
- September: Sales \$3.87M, Expenses \$2.70M, Net Profit \$1.17M
- October: Sales \$4.33M, Expenses \$2.83M, Net Profit \$1.50M
- November: Sales \$4.41M, Expenses \$2.86M, Net Profit \$1.55M
- December: Sales \$4.62M, Expenses \$2.90M, Net Profit \$1.72M

Retrieved Context

CloudWatch Metrics

File Explorer

Task View

File

Run

Taskbar icons

ENG IN 12-12-2025 01:35

8. What I Learned

This project strengthened my ability to design and deploy a **production-grade, serverless RAG system** on AWS. Key takeaways:

- Building RAG pipelines: chunking, embeddings, vector indexing, and retrieval.
- Using Amazon Bedrock, OpenSearch Serverless, Lambda, and API Gateway together in a real system.
- Implementing secure IAM roles and fully serverless compute.
- Debugging end-to-end workloads with CloudWatch.
- Designing a polished front-end UI that interacts with AI APIs.

Overall, this project helped me think like a **Cloud + AI Engineer** building real enterprise solutions.

9. Future Enhancements

Planned improvements to evolve the system further:

- Automated document re-indexing
- Support for PDFs, DOCX, and scanned files
- Role-based access control (RBAC)
- Multi-turn conversational memory
- Streaming LLM responses
- Multi-model support (Claude, Titan, Llama)
- Infra-as-code deployment using CDK

These enhancements move the solution closer to a full enterprise AI knowledge platform.

10. Final Conclusion

The *TitanEnterprise RAG Engine* showcases how cloud-native, serverless architecture and generative AI can deliver fast, accurate access to internal knowledge.

This project demonstrates my ability to build:

- Scalable serverless AI systems
- Secure vector-search pipelines
- Production-quality workflows and UI

It reflects strong skills in **AWS, Bedrock, OpenSearch, and modern AI engineering**, and represents a real-world enterprise-grade solution.