ClarifyAI — Company Document Insight Assistant

GROUP & TEAM MEMBERS

1726 GROUP 2

700771333 - GHATTAMANENI LIKHITHA

700773763 - KOMATLAPALLI VENKATA NAGA SRI

700772413 - NIDHIN NINAN

700763677 - ROHINI PATTURAJA

Abstract

ClarifyAI is an internal, role-aware question-answering system designed for organizations that maintain controlled knowledge bases. The application allows employees to pose natural-language questions and receive precise, citation-backed responses derived exclusively from sanctioned documents. The solution adopts a Retrieval-Augmented Generation (RAG) pattern on AWS, coupling managed retrieval (Bedrock Knowledge Bases over S3 Vector Store Serverless) with lightweight generation via Bedrock foundation models. A React interface, hosted on Amplify and secured with Amazon Cognito, ensures simple access control (Admin vs Employee), rapid deployment, and low operational overhead.

List of Figures

1. Introduction

In many companies, employees waste time trying to hunt for the right document or the correct answer. Search tools often give confusing or incomplete results, and it is not always clear which version of a document is the latest or most authoritative. ClarifyAI was created to fix this problem by giving clean answers directly from the company's documents. It uses a modern RAG system on AWS to make the process reliable, secure, and easy to maintain.

ClarifyAI is an internal Question-Answering (Q&A) assistant for organizations that need trustworthy, citation-backed answers sourced only from their approved documents. It pairs a simple React UI with a retrieval-augmented backend so employees can ask natural questions and immediately see the sources that support each response.

1.1 Existing Systems

Most legacy enterprise bots rely on keyword search or rule-based patterns to match questions to predefined answers. Some newer systems call a general LLM directly without constraining it to internal knowledge.

Drawbacks

Both approaches have serious limitations:

- **Prompt fragility** – Users must guess the "right" wording; minor phrasing changes produce inconsistent results.
- **No provenance** – Answers rarely include citations, reducing trust and auditability.
- **Domain drift and hallucinations** – General LLMs may invent facts or pull from non-approved sources.
- **Operational friction** – Pipelines are not designed for frequent document updates and re-indexing.

2. Proposed System

ClarifyAI is a serverless, role-aware, retrieval-augmented assistant that confines answers to an approved document corpus. Curated files live in Amazon S3 and are indexed by Bedrock Knowledge Bases into S3 Vector Store Serverless for semantic retrieval. A React UI hosted on Amplify authenticates users with Amazon Cognito and calls a secured API (API Gateway → Lambda). Lambda orchestrates Bedrock's retrieve-and-generate flow to compose grounded responses and returns citation-backed answers. The objective is to deliver reliable, policy-aligned outputs with minimal operational overhead and clear access controls.

2.1 Key Features

- **Grounded responses with citations** – Retrieve relevant passages before generation; show sources for transparency.
- **Role-based access** – Admins can upload and sync; Employees can only query. The UI adapts based on Cognito group.
- **Secure, low-ops stack** – Cognito for auth; API Gateway + Lambda for a thin, auditable API; S3 Vector Store Serverless for vectors; Amplify Hosting for fast deployments.
- **Observability** – CloudWatch logs for API/Lambda and a visible Knowledge Base Sync status for admins.

3. Technical Stack

3.1. Cloud Services

- **Amazon S3** – Document storage and source of truth.
- **S3 Vector Store Serverless** – Vector index for semantic retrieval.
- **Amazon Bedrock** – Knowledge Bases + foundation models for RAG.
- **AWS Lambda (Python 3.11)** – Serverless compute for query and upload workflows.
- **Amazon API Gateway** – Secured REST API front door.
- **Amazon Cognito (User Pool & Identity Pool)** – Authentication and scoped credentials.
- **AWS Amplify Hosting** – Frontend hosting and CI/CD.

3.2. Frontend Stack

- **React** single-page application.
- **aws-amplify** for API and auth integration.
- **@aws-amplify/ui-react** for prebuilt authentication and layout components.

3.3. DevOps

- Git repository integrated with **Amplify CI/CD**.
- Environment variables for endpoints, IDs, and secrets configuration (API URLs, Cognito IDs, Knowledge Base IDs).

4. System Architecture

ClarifyAI follows a layered RAG architecture that separates concerns for robustness and security:

- **User Authentication & UI** — A Custom React app hosted on Amplify uses Cognito for sign-in/sign-out (custom login UI) and a group-aware UI (Admin vs Employee).
- **Secure API Endpoint** — A minimal AWS Lambda function, fronted by API Gateway, validates tokens, orchestrates retrieval-and-generation with Bedrock, and formats the response as { text, citations[] }. Another Lambda function is to collect admin uploaded documents and uploads then to S3 bucket.
- **Orchestrates RAG Process** — Bedrock Knowledge Base ingests from S3, performs chunking and embeddings, and stores vectors in S3 Vector Store Serverless for semantic retrieval. A manual Sync ensures admins control when new content becomes queryable and the sync can also be triggered when the admin uploads the document through the admin dashboard.
- **Source of Truth & Vector Index** — Amazon S3 is the source of truth for curated documents (encrypted at rest). Uploads are intentional and traceable.

Architecture Diagram

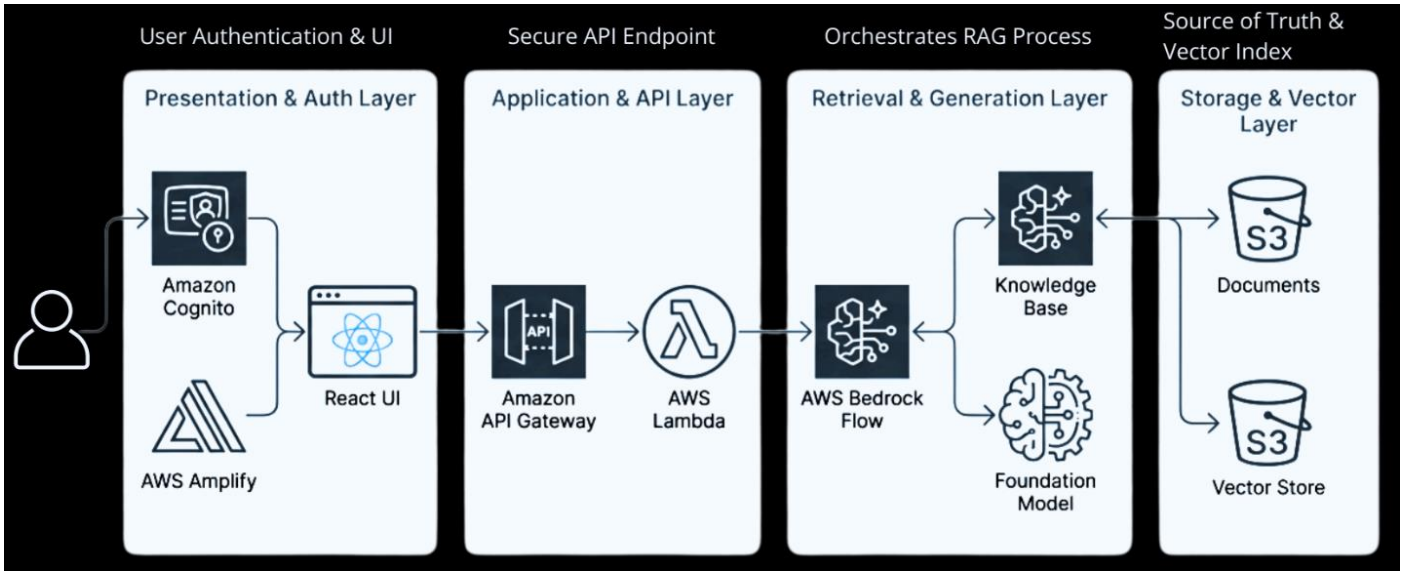


Fig 1. System Architecture Overview

5. System Implementation

ClarifyAI consists of a controlled content onboarding path and a secure query path. Admins onboard content to S3, then explicitly sync the Knowledge Base so vectors in S3 Vector Store reflect the current corpus. End-users authenticate via Cognito; the UI calls the API with an ID token. Lambda invokes Bedrock Flow with the user's question; Bedrock retrieves relevant chunks from S3 Vector Store via the Knowledge Base and generates an answer. This flow balances freshness, control, and transparency.

Application Workflow

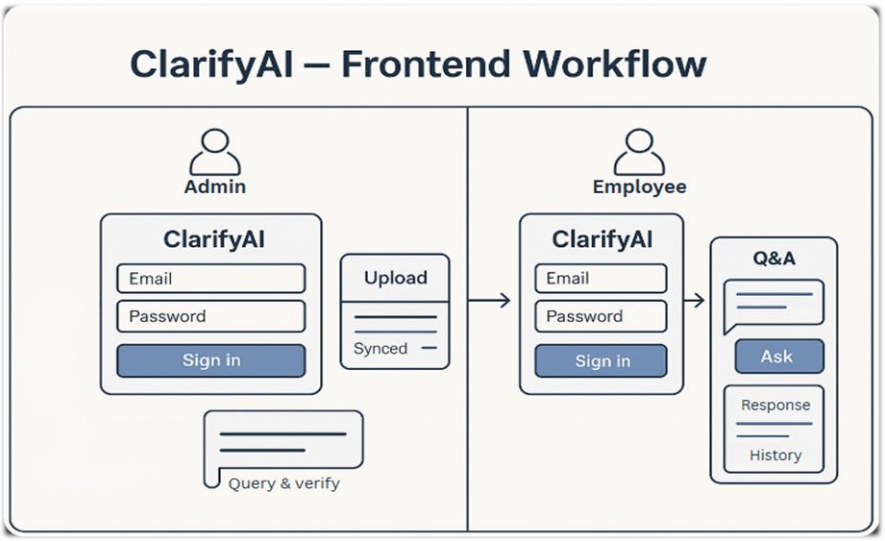


Fig 1. ClarifyAI Application workflow

5.1. Landing and Sign-In

- Landing page with project branding and options to sign in as **Admin** or **User**.
- Sign-in uses Cognito (custom UI styled in React, backed by Cognito Hosted UI and User Pool).
- New users are manually verified and assigned to either the Admin or User group.

5.2. Admin Dashboard Workflow: Securely Managing the Knowledge Corpus

Authenticate & Upload

- Admin signs in via the Amplify UI, authenticated by Amazon Cognito. Cognito Identity Pool provides temporary IAM credentials granting S3 write access.
- Admin uploads new documents (PDF, DOCX, etc.) through the UI to a designated S3 bucket.

Synchronize Knowledge Base

- Admin triggers a 'Sync' operation from the dashboard.
- Bedrock Knowledge Base crawls the S3 bucket, chunks tne new documents, and generates embeddings using Amazon Titan.

Index Vectors

- The new vector embeddings are written to the AWS S3 Vector Store, making the new content available for queries.
- The admin controls exactly when new content becomes queryable.

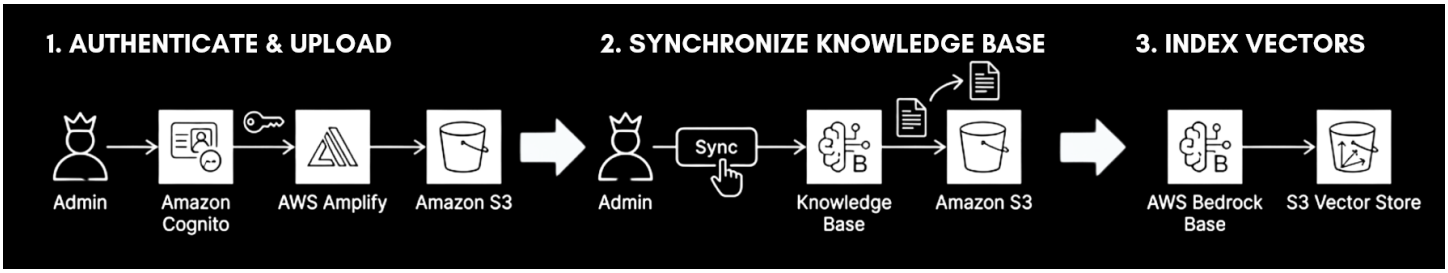


Fig 3. Admin Dashboard Workflow

5.3. Employee Dashboard Workflow: Querying for Trusted, Verifiable Insights

Authenticate & Query

- Employee signs in via the Amplify UI, authenticated by Amazon Cognito.
- The UI presents a simple query interface (no upload functionality). Employee submits a natural-language question.

Secure API Call

- The React app makes a POST request to API Gateway, including the user's Cognito ID token in the Authorization header.
- API Gateway validates the token with its Cognito Authorizer before invoking the Lambda function.

Retrieve, Generate & Respond

- Lambda invokes an AWS Bedrock Flow, which orchestrates the RAG process.
- The flow retrieves the most relevant text chunks from the S3 Vector Store (via the Bedrock Knowledge Base) and provides them as context to a Bedrock Foundation Model to generate an answer.
- The final response, including the generated text and citations, is returned to the user.

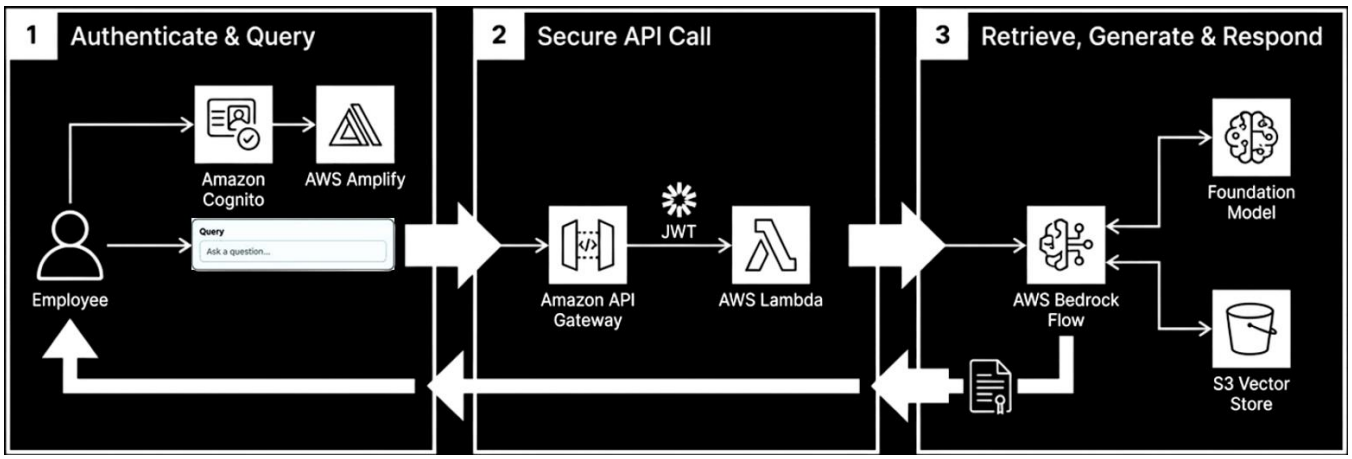


Fig 4. Employee Dashboard Workflow

7. Technology Core: The RAG Engine

The RAG engine is the heart of ClarifyAI.

- **Amazon S3 (Data Source)** – Authoritative, encrypted repository for curated documents; all onboarding flows write here.
- **Bedrock Knowledge Base** – Managed pipeline for ingestion, chunking, and embedding into S3 Vector Store.
- **S3 Vector Store Serverless** – Managed vector store for k-NN semantic search; chosen for lower cost and S3-style pricing.
- **Bedrock Flow** – Visual orchestrator that wires up flow input, Knowledge Base retrieval, prompt, and flow output for retrieve-and-generate.

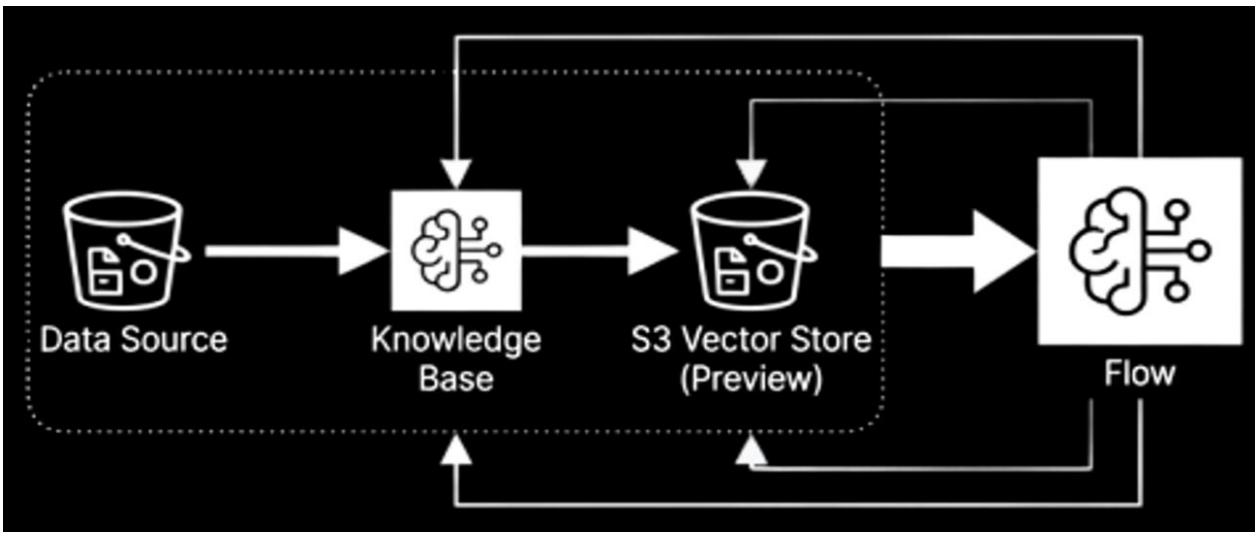


Fig 5. The RAG Engine

7.1. AWS Bedrock Flow

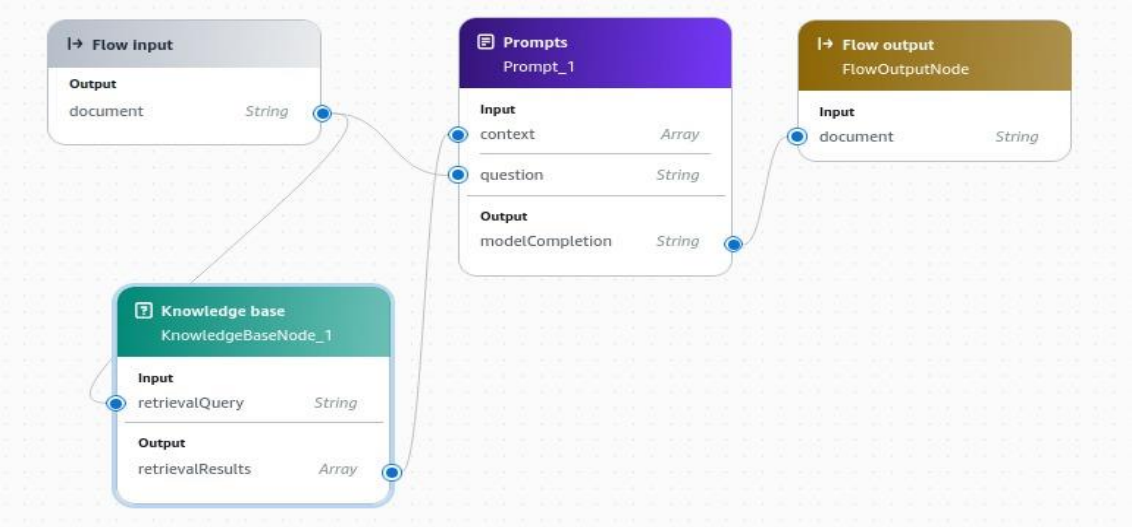


Fig 6. Bedrock Flow

8. Technology Core: Application Backbone

- **AWS Amplify Hosting** – CI/CD pipeline and global CDN for the React frontend; integrates with Git for automated builds and zero-downtime rollbacks.
- **Amazon Cognito** – User Pool manages users and groups; Identity Pool issues scoped AWS credentials.
- **Amazon API Gateway** – REST API exposing /query and /upload endpoints, protected by a Cognito authorizer.
- **AWS Lambda (RAGQueryFunction & DocumentUploadHandler)** – Stateless functions that call Bedrock Flow or handle S3 uploads and ingestion jobs.

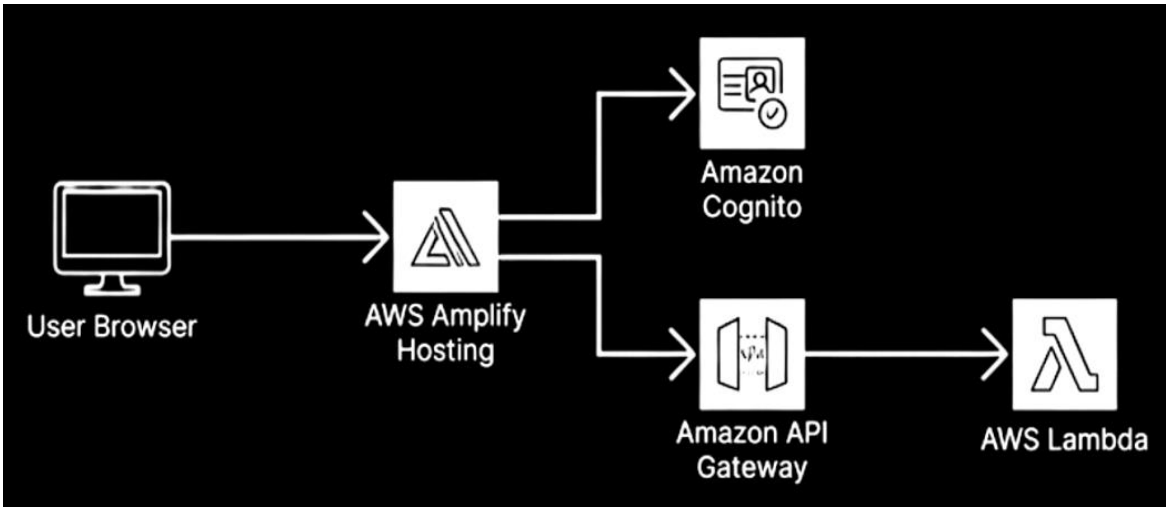


Fig 7. Application Backbone Diagram

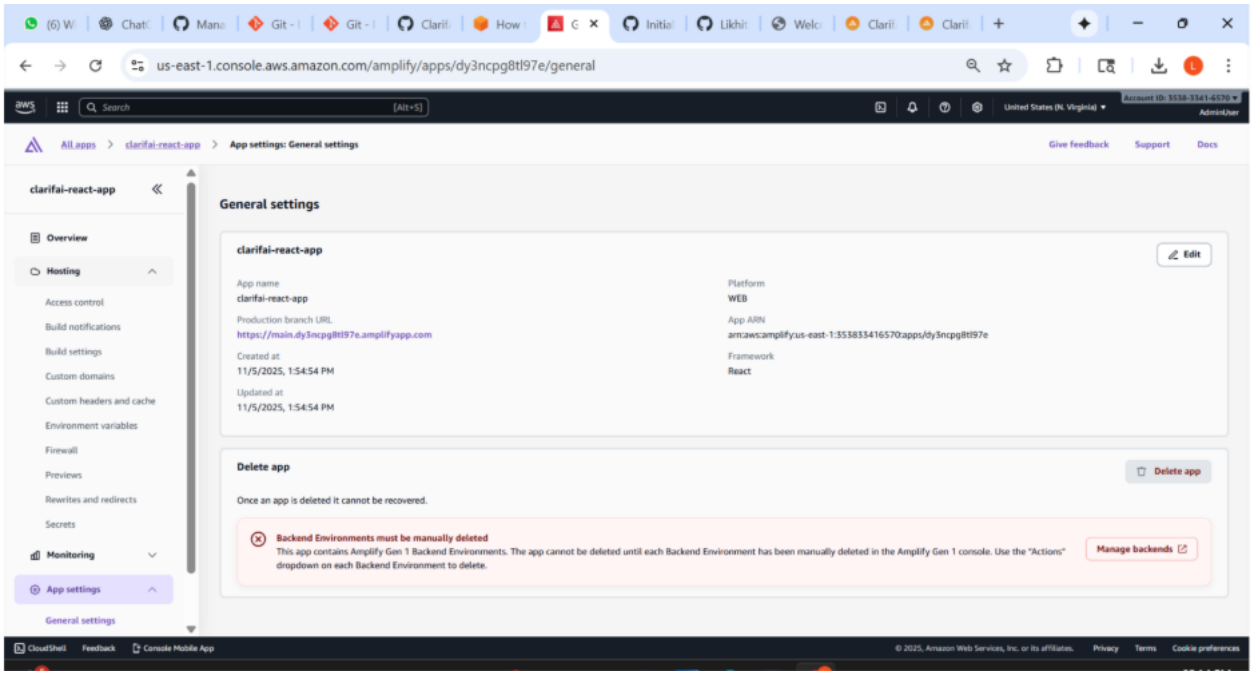
9. Project Modules AWS and Configurations

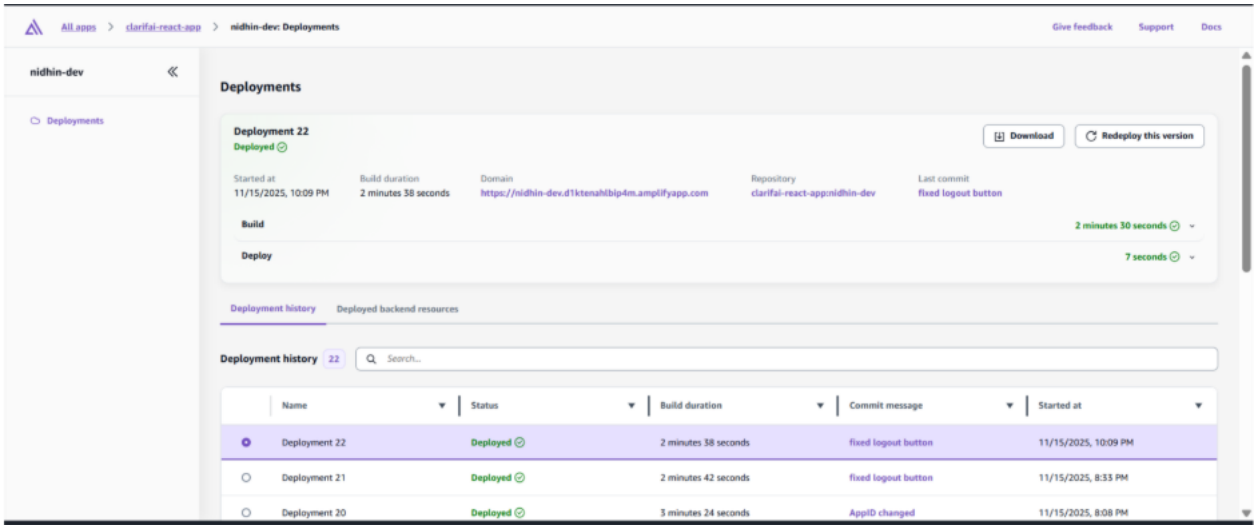
9.1. Amplify-Hosted React UI

An application deployed via **AWS Amplify Hosting** that serves the ClarifyAI frontend. It provides the landing page, sign-in views for Admin/User, the Admin dashboard (upload + test query), and the Employee Q&A screen. It uses **Amplify UI <Authenticator />** and aws-exports.js for a quick, secure integration with Cognito and API Gateway.

Configuration (summary):

1. Pushed the React app to a Git repository (main branch).
2. In **Amplify → Host Web App**, connected the repo and selected the branch.
3. Accepted the default React build settings (install → build) and added environment variables for:
    a. Cognito User Pool ID, App Client ID, Identity Pool ID
    b. API Gateway base URL
    c. Knowledge Base / Flow IDs (if needed on UI side)
4. Configured **rewrites & redirects** for SPA routing (all paths → /index.html).
5. Deployed and verified the live domain:
    a. Example: https://nidhin-dev.d1ktenahlbip4m.amplifyapp.com/
6. Updated aws-exports.js and Amplify.configure(awsExports) in React so <Authenticator /> and API calls use the deployed backend correctly.





9.2 Amazon Cognito (User Pool + Identity Pool)

Cognito provides the managed user directory and token-based authentication, plus temporary AWS credentials via Identity Pool. It powers sign-in/sign-out for both roles and lets us map Admins and Users to different IAM roles.

Purpose in ClarifyAI:

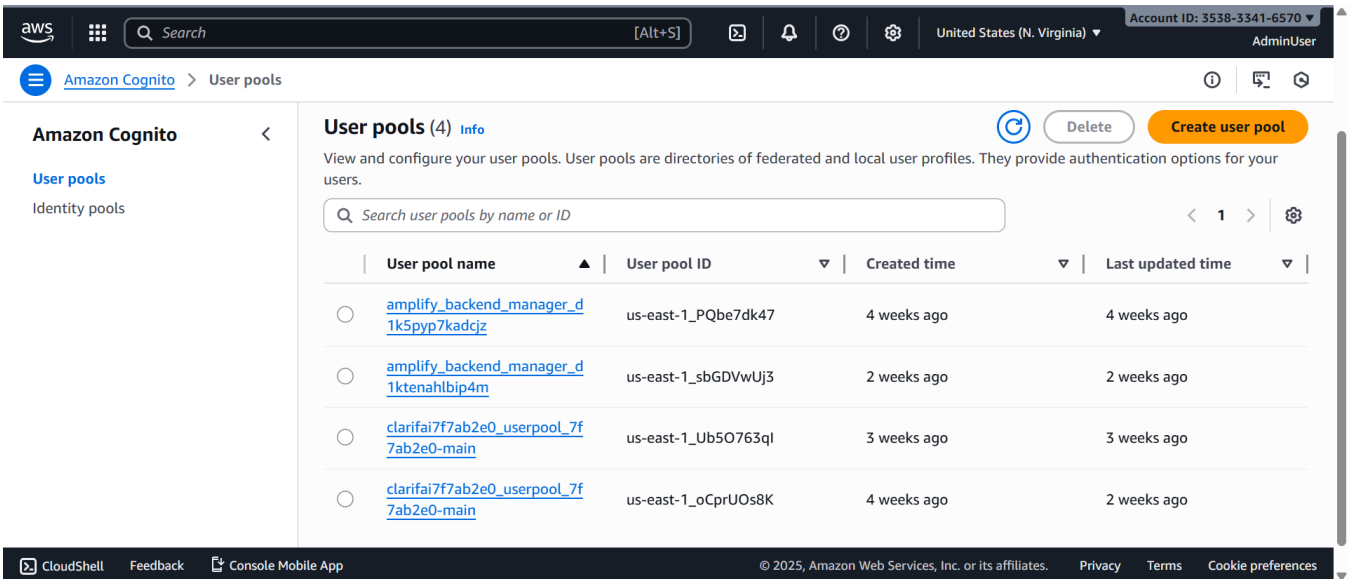
- Authenticate Admin and Employee users.
- Assign users to Admins or Users groups.
- Map Admins to an IAM role with S3 write permission for document uploads.
- Allow both groups to invoke Bedrock Flow via the protected API.

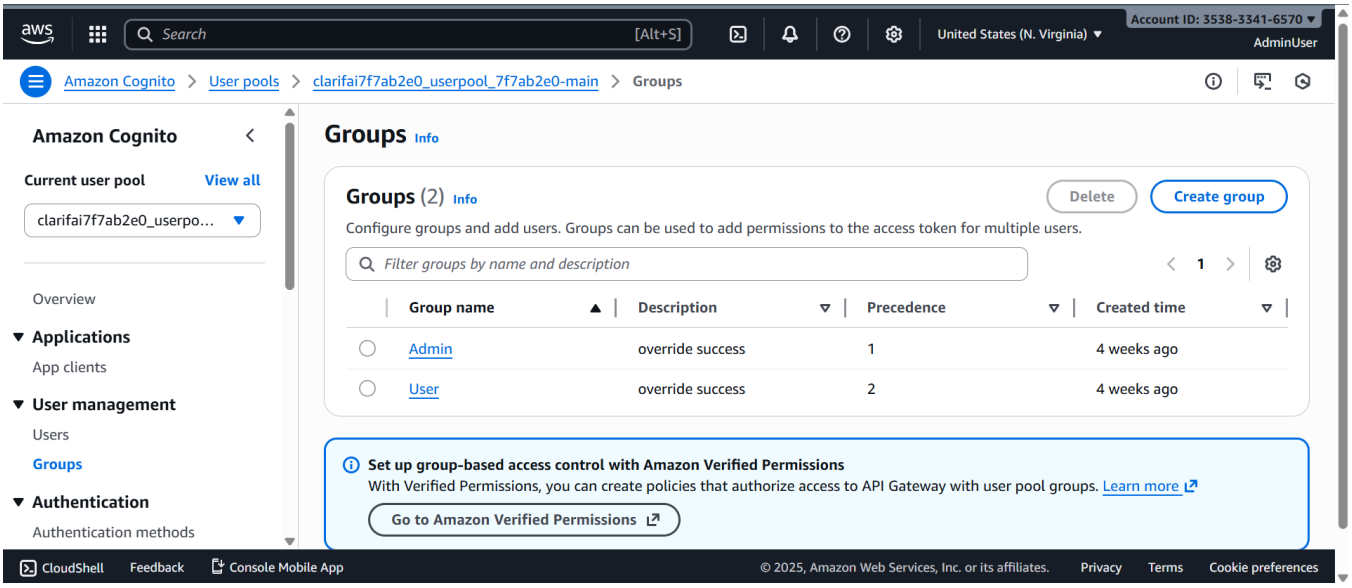
Configuration (summary):

- Created a User Pool for ClarifyAI (email as username, password policy, etc.)
- Created an App Client with no client secret for SPA usage.
- Configured a Hosted UI domain for sign-in/redirect flows.
- Added two groups in the User Pool:
    - Admins – upload + query
    - Users – query-only
- Created an Identity Pool linked to this User Pool/App Client.
- Created IAM roles and mapped them in the Identity Pool:
    - CognitoAdminIAMRole with AdminS3UploadPolicy (write to docs bucket only).
    - Default authenticated role for standard users (no S3 write).
- Updated the React app (via Amplify config) with User Pool ID, App Client ID, Identity Pool ID so <Authenticator /> and Auth/API work end-to-end.

Maintenance note:

When documents are changed or new ones are added, **re-sync the Knowledge Base** to create new embeddings. *Repeat: upload → Sync → quick test.*





9.3 Amazon API Gateway

API Gateway exposes ClarifyAI's backend as a REST API and protects it with a **Cognito authorizer**. It is the "front door" for /query and /upload, enforcing CORS and token validation before any Lambda is run.

Purpose in ClarifyAI:

- Provide POST /query for Bedrock RAG queries.
- Provide POST /upload for Admin document uploads.
- Validate Cognito ID tokens on every request.
- Return a standard HTTP 200 response with a pure JSON body to the dashboard.

Configuration (summary):

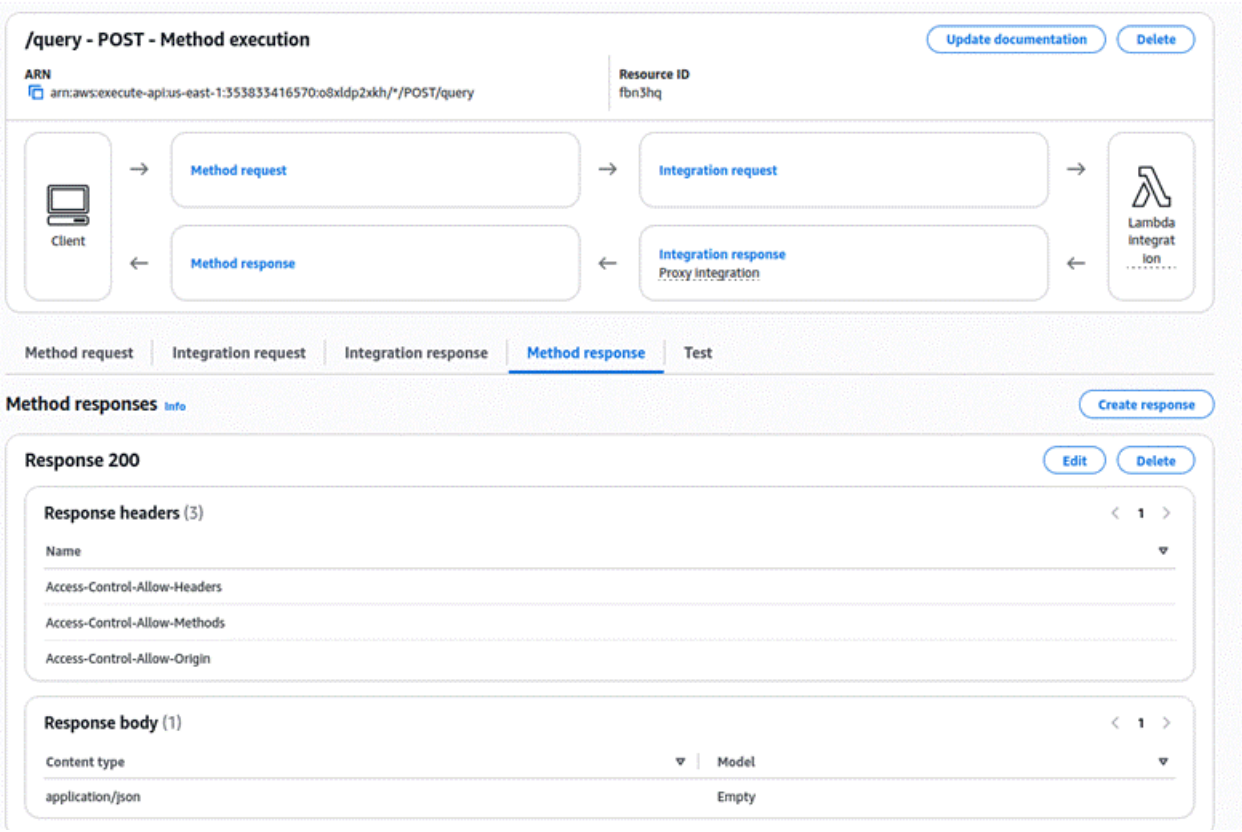
- API Name: **ClarifyAIAPI**
- Region: us-east-1

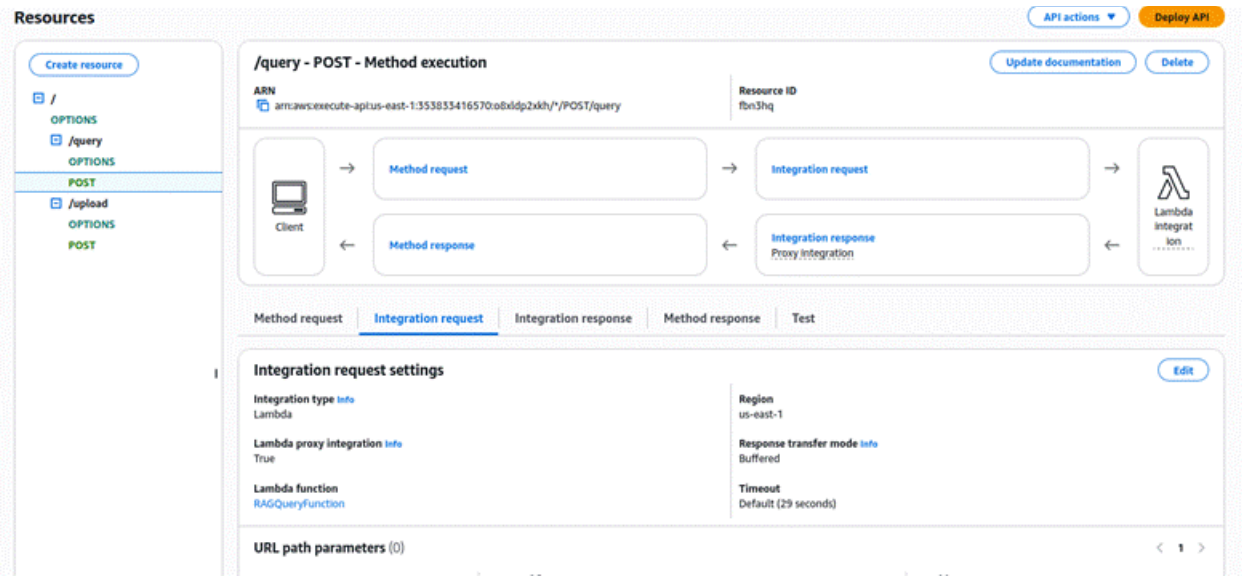
Resources and Methods

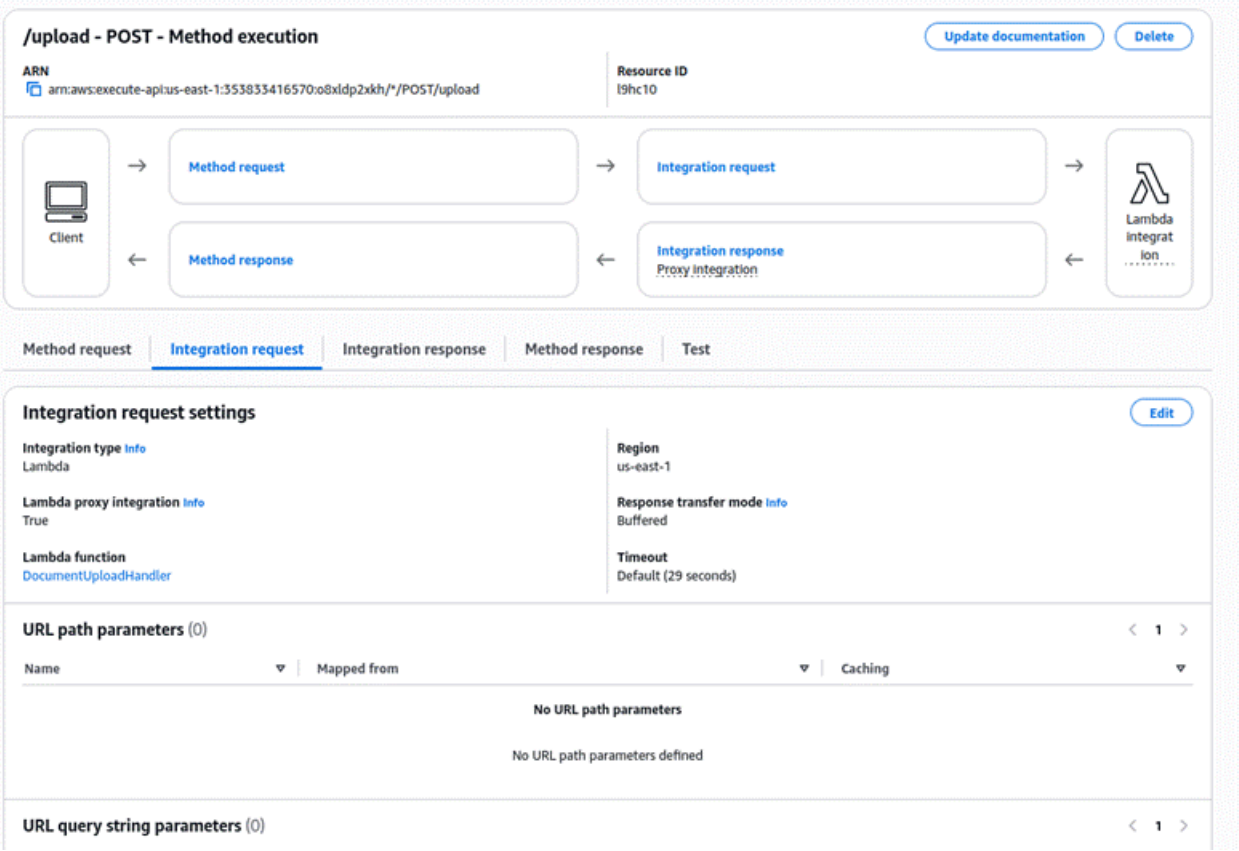
- /query
    - Methods: OPTIONS, POST
    - POST uses **Lambda proxy integration = True**
    - Authorization: **CognitoAuthorizer**
    - Resource ID: e.g., fbn3hq
- /upload
    - Methods: OPTIONS, POST
    - POST uses **Lambda proxy integration = True**
    - Authorization: **CognitoAuthorizer**

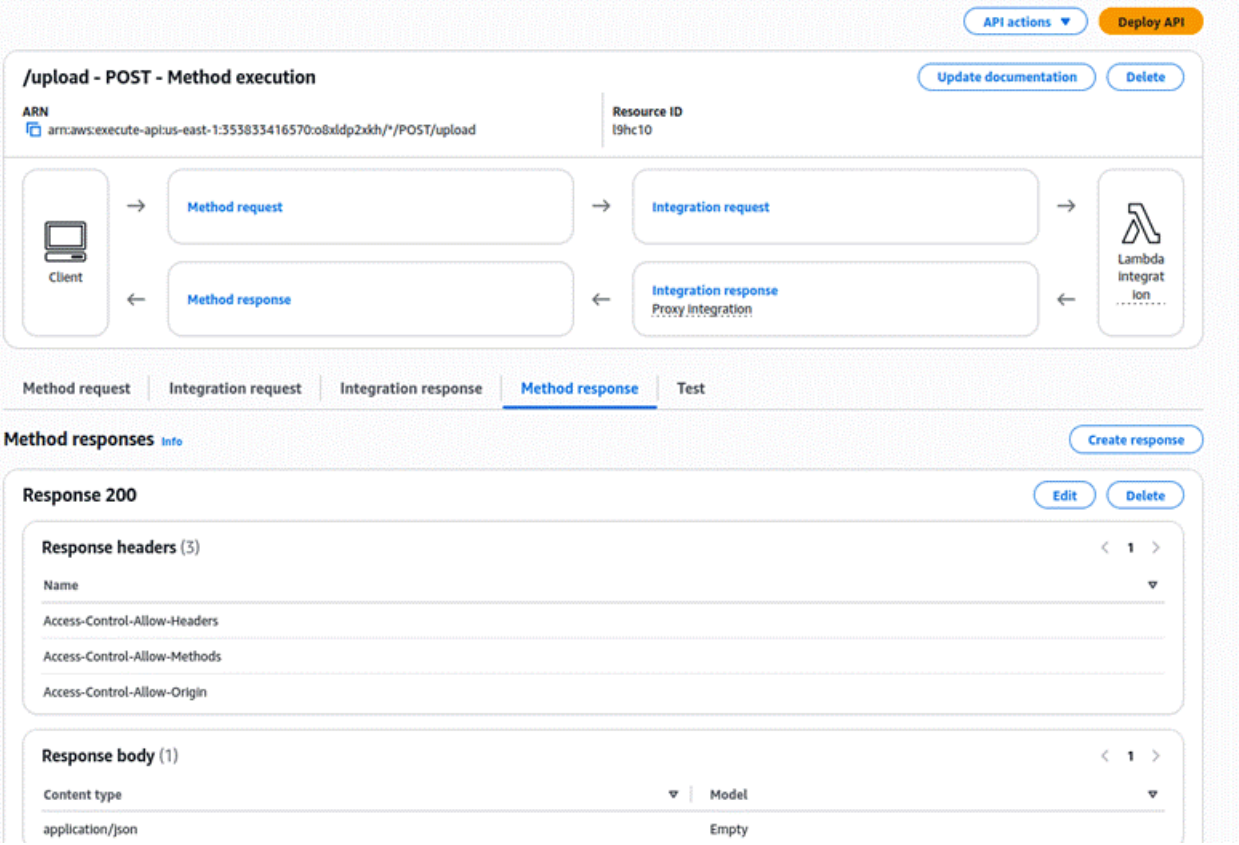
JSON payload contract (Lambda → API response body):

```
{
 "response": "The generated text answer from Bedrock Flow goes here.",
 "execution_id": "exec-123456789",
 "metadata": {
        "completion_reason": "SUCCESS"
 }
}
```


/query - POST - Method execution
Update documentation
Delete
ARN
arn:aws:execute-api:us-east-1:353833416570:o8xldp2xkh/*/POST/query
Resource ID
fbn3hq
Client
Method request
Integration request
Lambda integration
Method response
Integration response
Proxy integration
Method request
Integration request
Integration response
Method response
Test
Method responses Info
Create response
Response 200
Edit
Delete
Response headers (3)
Name
Access-Control-Allow-Headers
Access-Control-Allow-Methods
Access-Control-Allow-Origin
Response body (1)
Content type
Model
application/json
Empty


Resources
API actions
Deploy API
Create resource
/
OPTIONS
/query
OPTIONS
POST
/upload
OPTIONS
POST
/query - POST - Method execution
Update documentation
Delete
ARN
arn:aws:execute-api:us-east-1:353833416570:o8xldp2xkh/*/POST/query
Resource ID
fbn3hq
Client
Method request
Integration request
Lambda integration
Method response
Integration response
Proxy integration
Method request
Integration request
Integration response
Method response
Test
Integration request settings
Edit
Integration type Info
Lambda
Region
us-east-1
Lambda proxy integration Info
True
Response transfer mode Info
Buffered
Lambda function
RAGQueryFunction
Timeout
Default (29 seconds)
URL path parameters (0)


/upload - POST - Method execution
Update documentation
Delete
ARN
arn:aws:execute-api:us-east-1:353833416570:o8xldp2xkh/*/POST/upload
Resource ID
l9hc10
Client
Method request
Integration request
Lambda integration
Method response
Integration response
Proxy integration
Method request
Integration request
Integration response
Method response
Test
Integration request settings
Edit
Integration type Info
Lambda
Region
us-east-1
Lambda proxy integration Info
True
Response transfer mode Info
Buffered
Lambda function
DocumentUploadHandler
Timeout
Default (29 seconds)
URL path parameters (0)
Name
Mapped from
Caching
No URL path parameters
No URL path parameters defined
URL query string parameters (0)


API actions
Deploy API
/upload - POST - Method execution
Update documentation
Delete
ARN
arn:aws:execute-api:us-east-1:353833416570:o8xldp2xkh/*/POST/upload
Resource ID
l9hc10
Client
Method request
Integration request
Lambda integration
Method response
Integration response
Proxy integration
Method request
Integration request
Integration response
Method response
Test
Method responses Info
Create response
Response 200
Edit
Delete
Response headers (3)
Name
Access-Control-Allow-Headers
Access-Control-Allow-Methods
Access-Control-Allow-Origin
Response body (1)
Content type
Model
application/json
Empty

9.4 AWS Lambda (Python 3.11 / 3.13)

Two Lambda functions implement ClarifyAI's backend logic: a **query** function that talks to Bedrock Flow, and an **upload** function that sends files to S3 and triggers ingestion.

9.4.1 RAGQueryFunction

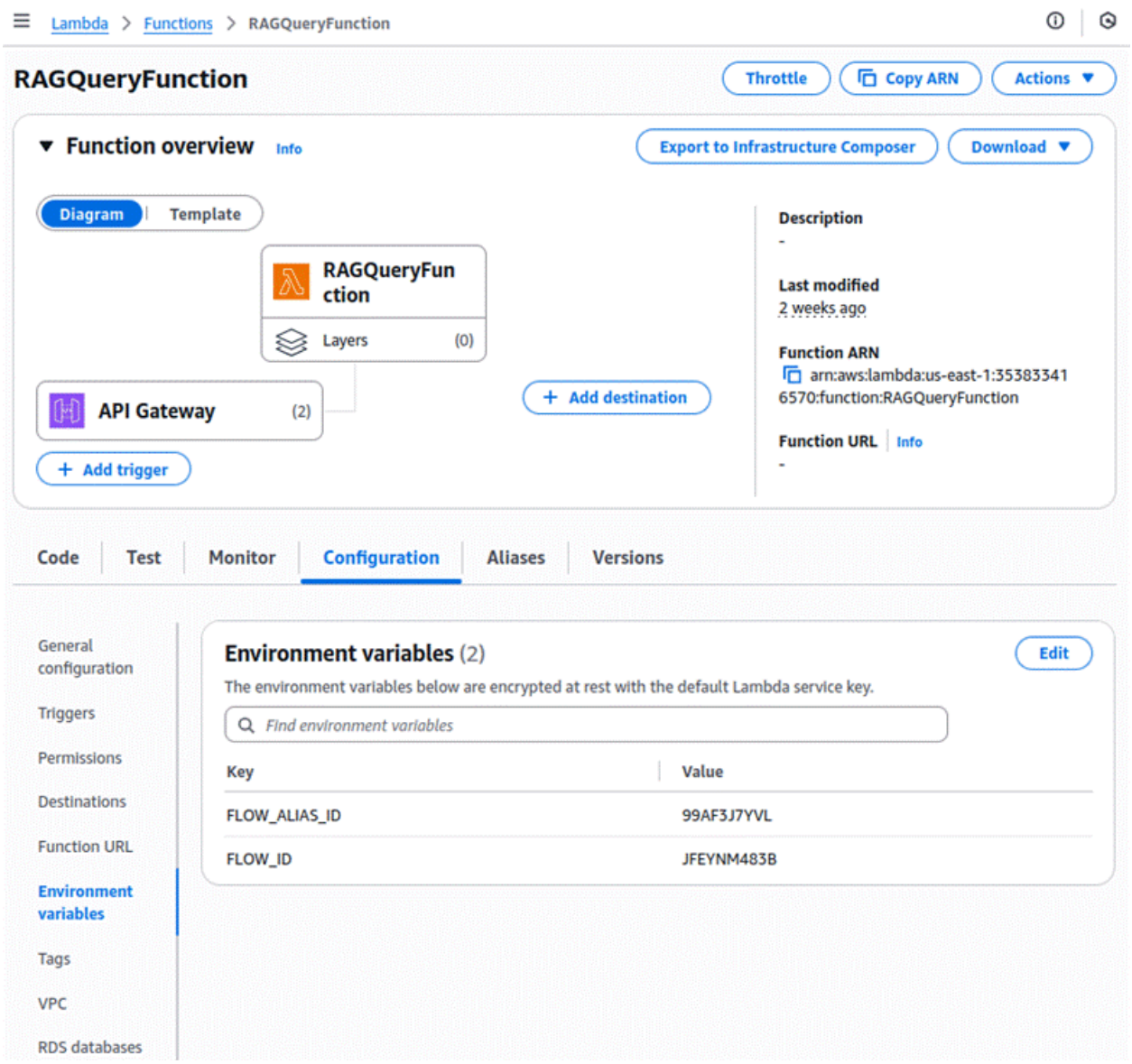
- Name: RAGQueryFunction
- Runtime: Python 3.13 (or 3.11 per environment)
- Purpose: Accepts a question, calls the Bedrock Flow, and returns a structured JSON answer.

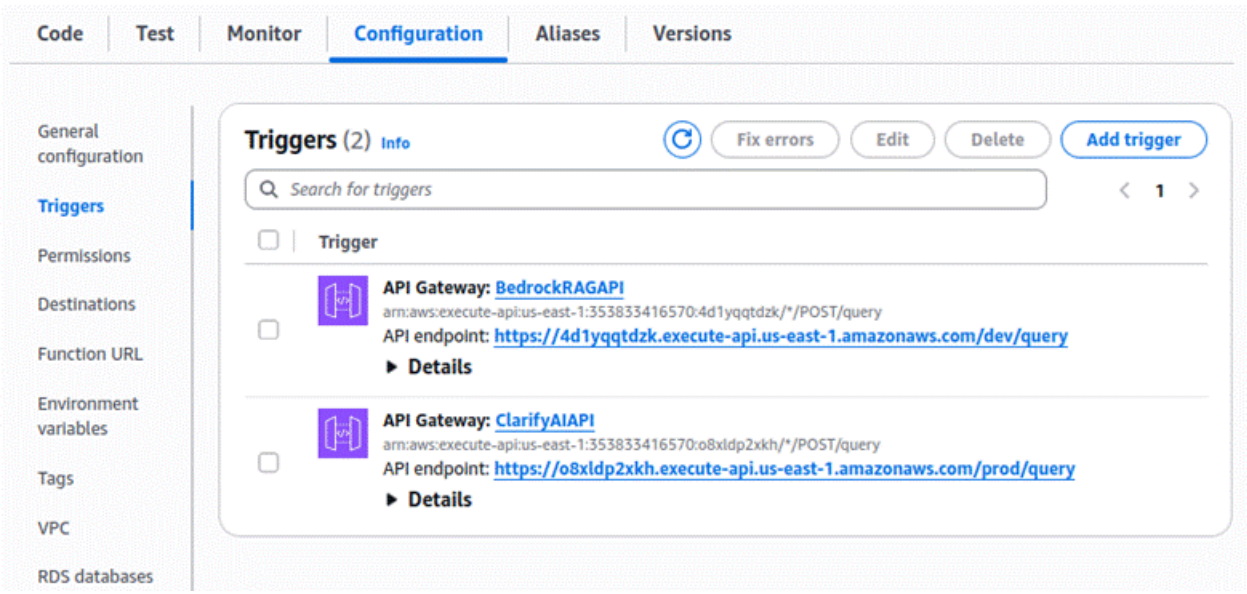
Key configuration:

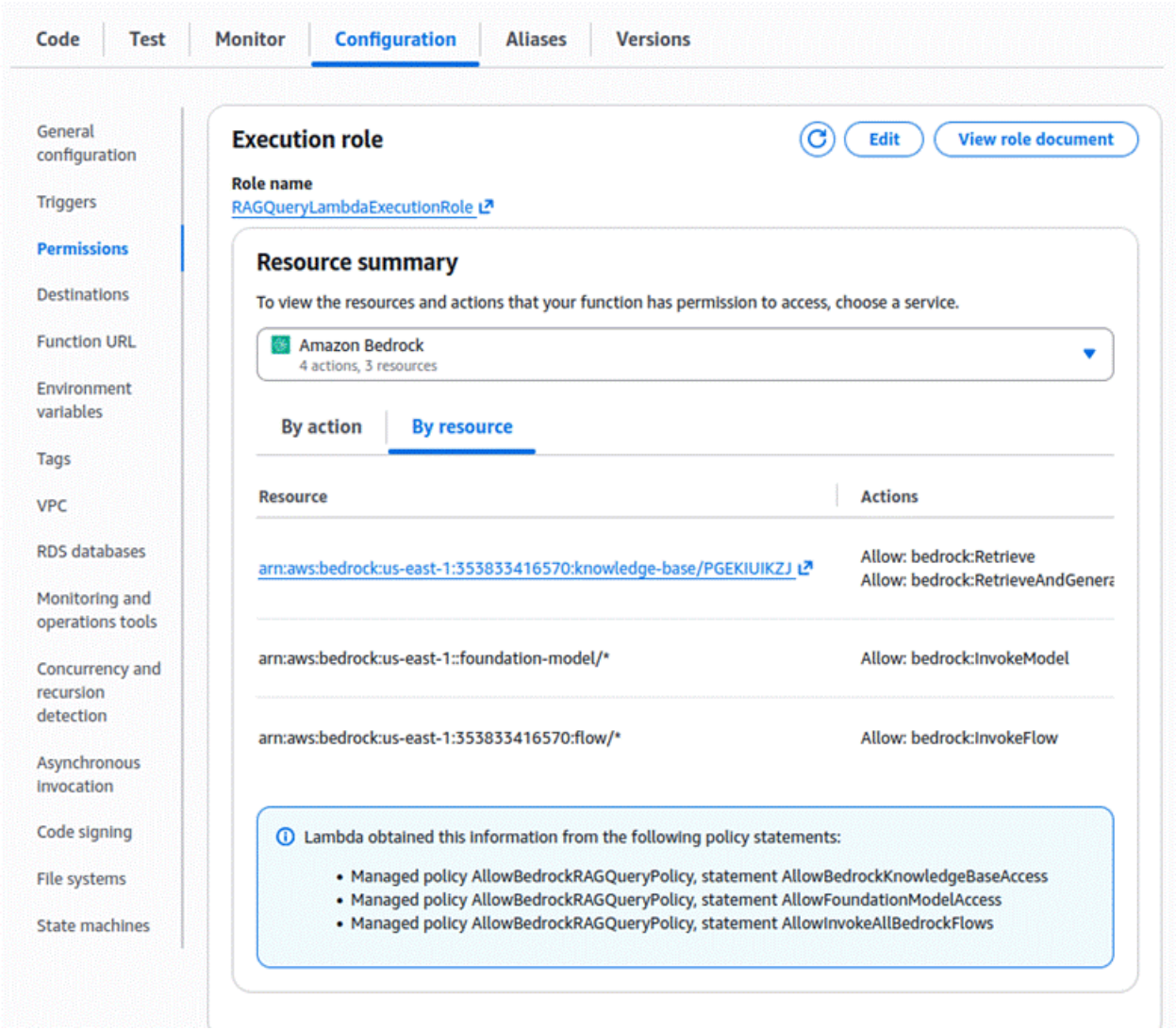
- Execution role allows:
  - bedrock:InvokeFlow on the Flow ARN.
  - CloudWatch Logs actions.
- CORS headers set in the response (e.g., allow Amplify domain, POST,OPTIONS, Authorization, Content-Type).
- Parses event.body JSON from API Gateway, extracts question, invokes Flow, and returns:

```
{
  "statusCode": 200,
  "headers": { ...CORS headers... },
  "body": "{\"response\": \"...\", \"execution_id\": \"...\", \"metadata\": {...}}"
}
```

The flow_response is converted to JSON (answer string, execution ID, metadata such as completion reason).


Lambda > Functions > RAGQueryFunction
RAGQueryFunction
Throttle
Copy ARN
Actions
Function overview Info
Export to Infrastructure Composer
Download
Diagram
Template
RAGQueryFunction
Layers (0)
API Gateway (2)
Add destination
Add trigger
Description
-
Last modified
2 weeks ago
Function ARN
arn:aws:lambda:us-east-1:353833416570:function:RAGQueryFunction
Function URL Info
-
Code
Test
Monitor
Configuration
Aliases
Versions
General configuration
Triggers
Permissions
Destinations
Function URL
Environment variables
Tags
VPC
RDS databases
Environment variables (2)
Edit
The environment variables below are encrypted at rest with the default Lambda service key.
Find environment variables
Key
Value
FLOW_ALIAS_ID
99AF3J7YVL
FLOW_ID
JFEYNM483B


Code
Test
Monitor
Configuration
Aliases
Versions
General configuration
Triggers
Permissions
Destinations
Function URL
Environment variables
Tags
VPC
RDS databases
Triggers (2) Info
Fix errors
Edit
Delete
Add trigger
Search for triggers
1
Trigger
API Gateway: BedrockRAGAPI
arn:aws:execute-api:us-east-1:353833416570:4d1yqqtdzk/*/POST/query
API endpoint: https://4d1yqqtdzk.execute-api.us-east-1.amazonaws.com/dev/query
Details
API Gateway: ClarifyAIAPI
arn:aws:execute-api:us-east-1:353833416570:o8xldp2xkh/*/POST/query
API endpoint: https://o8xldp2xkh.execute-api.us-east-1.amazonaws.com/prod/query
Details


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
Asynchronous invocation
Code signing
File systems
State machines
Execution role
Edit
View role document
Role name
RAGQueryLambdaExecutionRole
Resource summary
To view the resources and actions that your function has permission to access, choose a service.
Amazon Bedrock
4 actions, 3 resources
By action
By resource
Resource
Actions
arn:aws:bedrock:us-east-1:353833416570:knowledge-base/PGEKIUIKZJ
Allow: bedrock:Retrieve
Allow: bedrock:RetrieveAndGenera
arn:aws:bedrock:us-east-1::foundation-model/*
Allow: bedrock:InvokeModel
arn:aws:bedrock:us-east-1:353833416570:flow/*
Allow: bedrock:InvokeFlow
Lambda obtained this information from the following policy statements:
Managed policy AllowBedrockRAGQueryPolicy, statement AllowBedrockKnowledgeBaseAccess
Managed policy AllowBedrockRAGQueryPolicy, statement AllowFoundationModelAccess
Managed policy AllowBedrockRAGQueryPolicy, statement AllowInvokeAllBedrockFlows

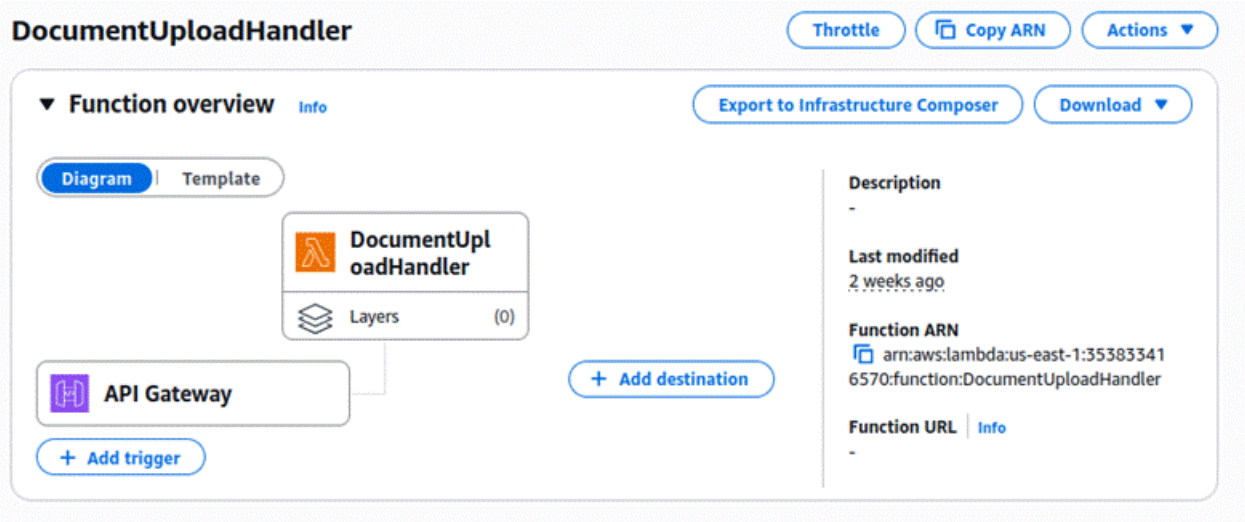
9.4.2 DocumentUploadHandler

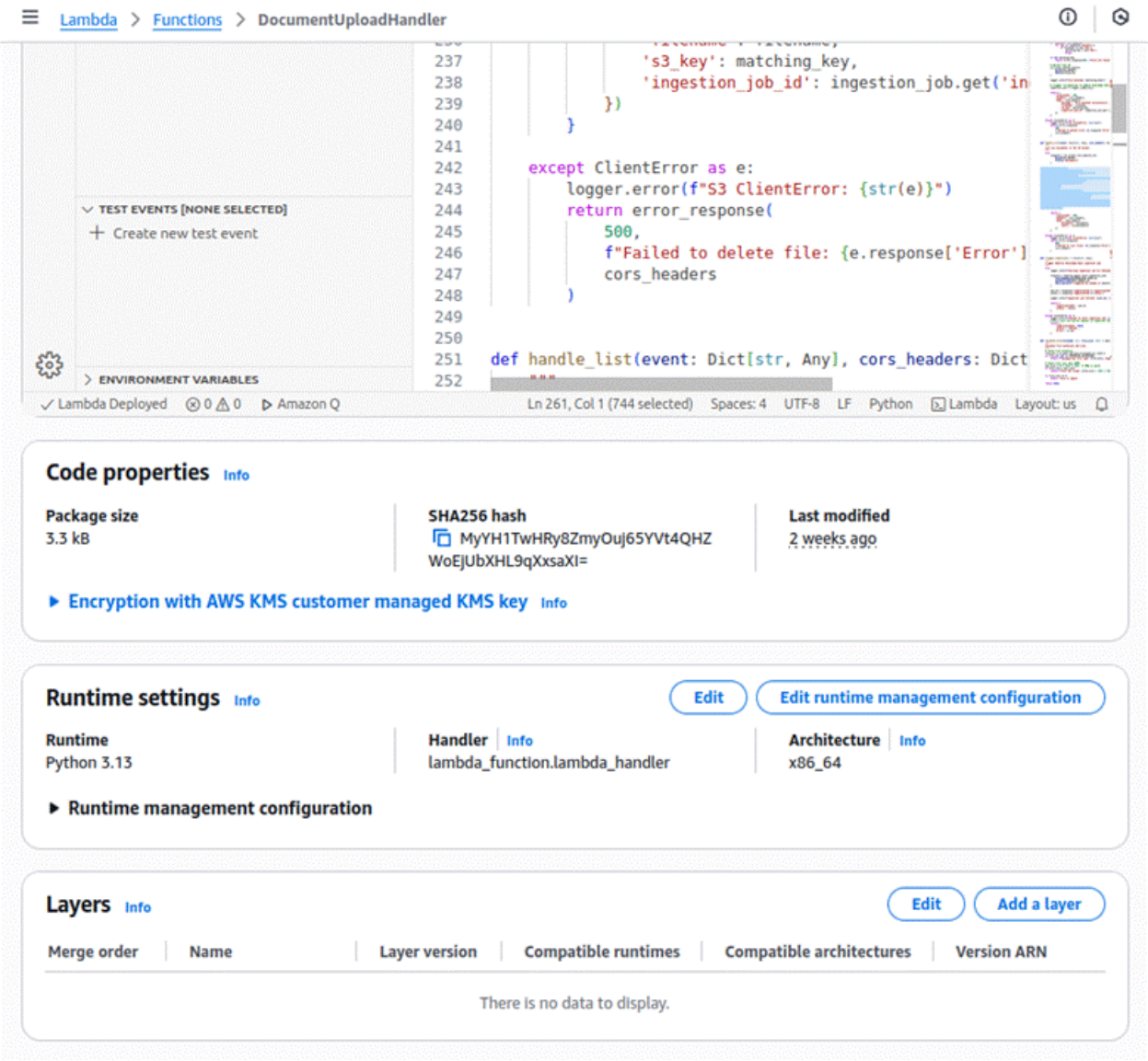
- Name: DocumentUploadHandler
- Runtime: Python 3.13
- Purpose: Handles Admin uploads, writes documents to S3, and triggers a Bedrock KB ingestion job.

Key configuration:

- Environment variables:
    - S3_BUCKET_NAME
    - KNOWLEDGE_BASE_ID
    - DATA_SOURCE_ID
- Parses the request body:
    - Reads fileName, fileContent (base64), and contentType.
    - Stores the file under documents/ prefix with metadata (original filename, upload timestamp).
- Calls start_ingestion_job on the Bedrock KB using the configured KNOWLEDGE_BASE_ID and DATA_SOURCE_ID.
- Returns JSON including upload and ingestion info (job ID, status).

Also exposes a helper path to **list all documents** in the S3 bucket by calling list_objects_v2 on the documents/ prefix and fetching metadata via head_object.





Data Structure returned by Lambda Function

```
{
    'statusCode': 200,
    'headers': cors_headers,
    'body': json.dumps({
        'response': converted to a string using json.dumps(),
        'execution_id': executiondID from Bedrock Flow,
        'metadata': extracts the “completionReason” (e.g., "SUCCESS")
    })
}
```

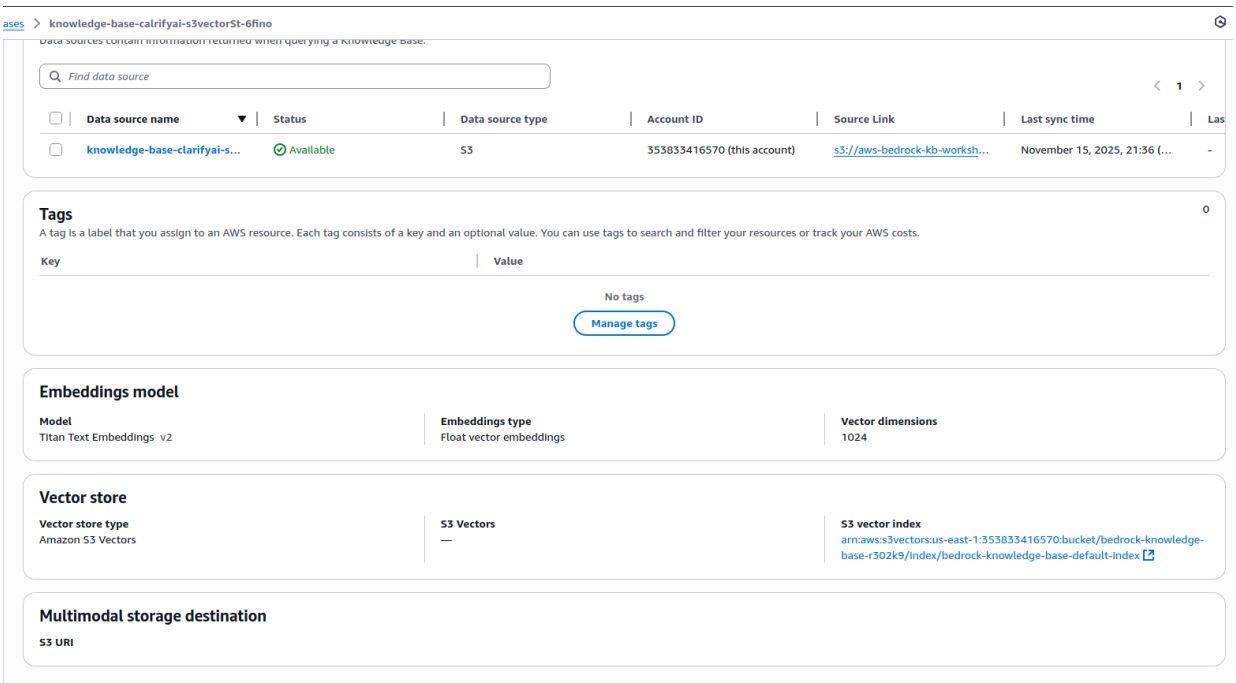
9.5 Bedrock Knowledge Base

The Knowledge Base manages ingestion, chunking, embedding, and retrieval against the S3 document corpus and S3 Vector Store. It is the central RAG retrieval layer for ClarifyAI.

Configuration (core settings):

- Name: knowledge-base-clarifyai-s3vectorSt-6fino
- ID: NKV6MGB8C1
- RAG type: **Vector store**
- Data Source:
    - Type: **Amazon S3**
    - S3 URI: s3://clarifyai-kb-s3vector/... (documents prefix)
    - Chunking: Fixed-size chunking, max **512 tokens** per chunk.
- Embeddings:
    - Model: **Titan Text Embeddings V2**
    - Embedding type: float vectors
    - Vector dimensions: **1024**
- Vector Store:
    - Type: **Amazon S3 Vectors (Preview)**
    - Index ARN: arn:aws:s3vectors:us-east-1:...:bucket/bedrock-knowledge-base-.../index/bedrock-knowledge-base-default-index

After creation, a **Sync** is triggered to ingest all existing documents and generate embeddings. Sync is repeated whenever new content is uploaded.

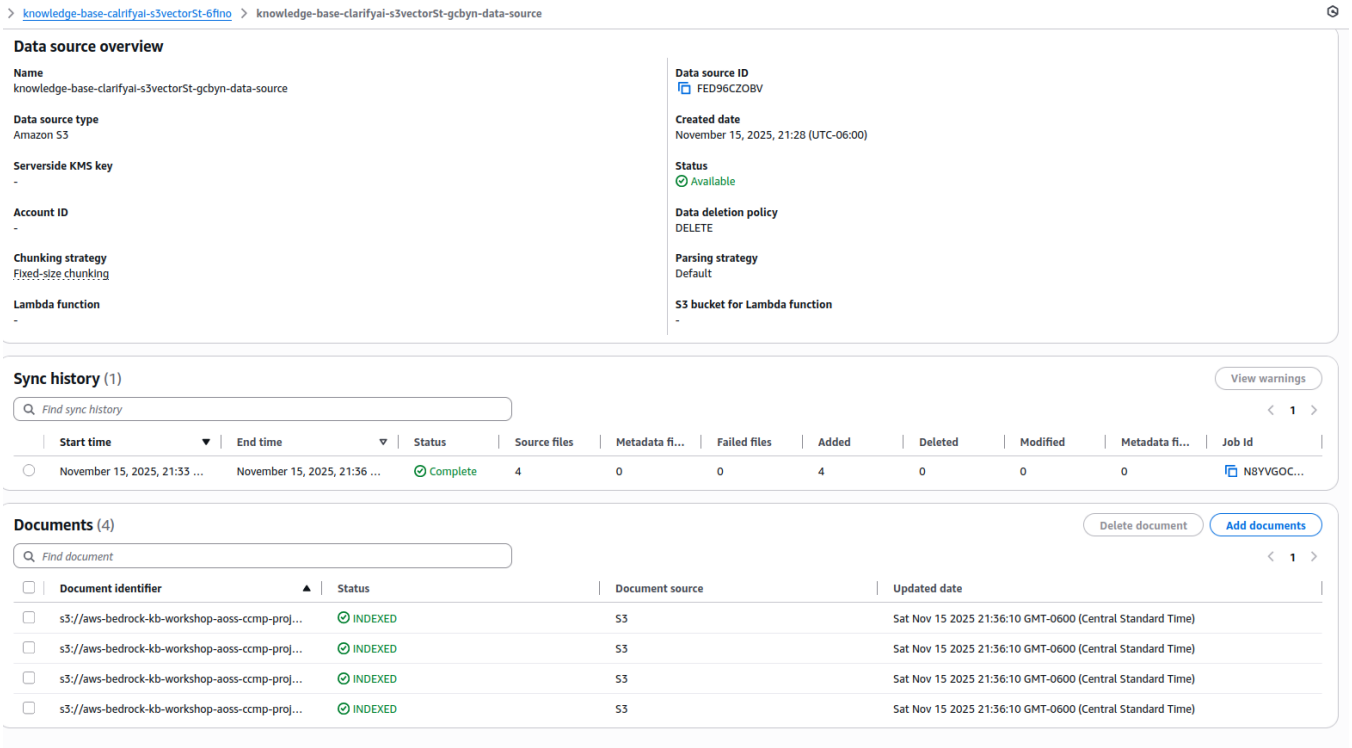


9.6 S3 Vector Store (Vector)

S3 Vector Store Serverless holds the embeddings that power semantic retrieval. The Knowledge Base writes here during Sync, and Bedrock Flow reads from it during queries.

Configuration (summary):

- Vector Store type: **Amazon S3 Vectors (Preview)**
- Vector Store Index ARN:
    - arn:aws:s3vectors:us-east-1:353833416570:bucket/bedrock-knowledge-base-r302k9/index/bedrock-knowledge-base-default-index
- Data source object:
    - Name: knowledge-base-clarifyai-s3vectorSt-gcbyn-data-source
    - Data source ID: FED96CZOBV



9.7 Amazon S3

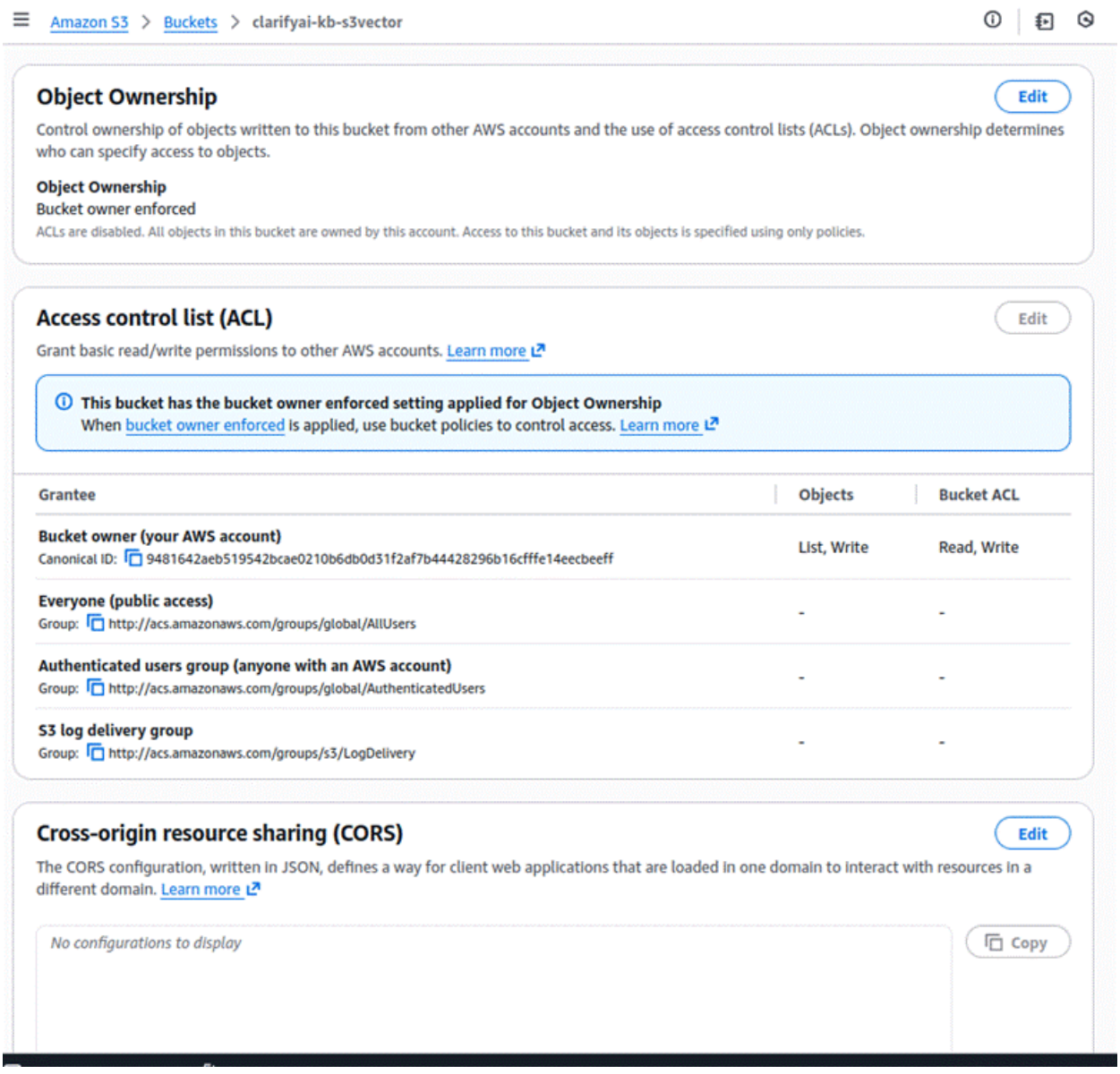
S3 hosts the authoritative document corpus and related KB artifacts. It also receives logs in a dedicated prefix. Access is tightly controlled using IAM and bucket policies.

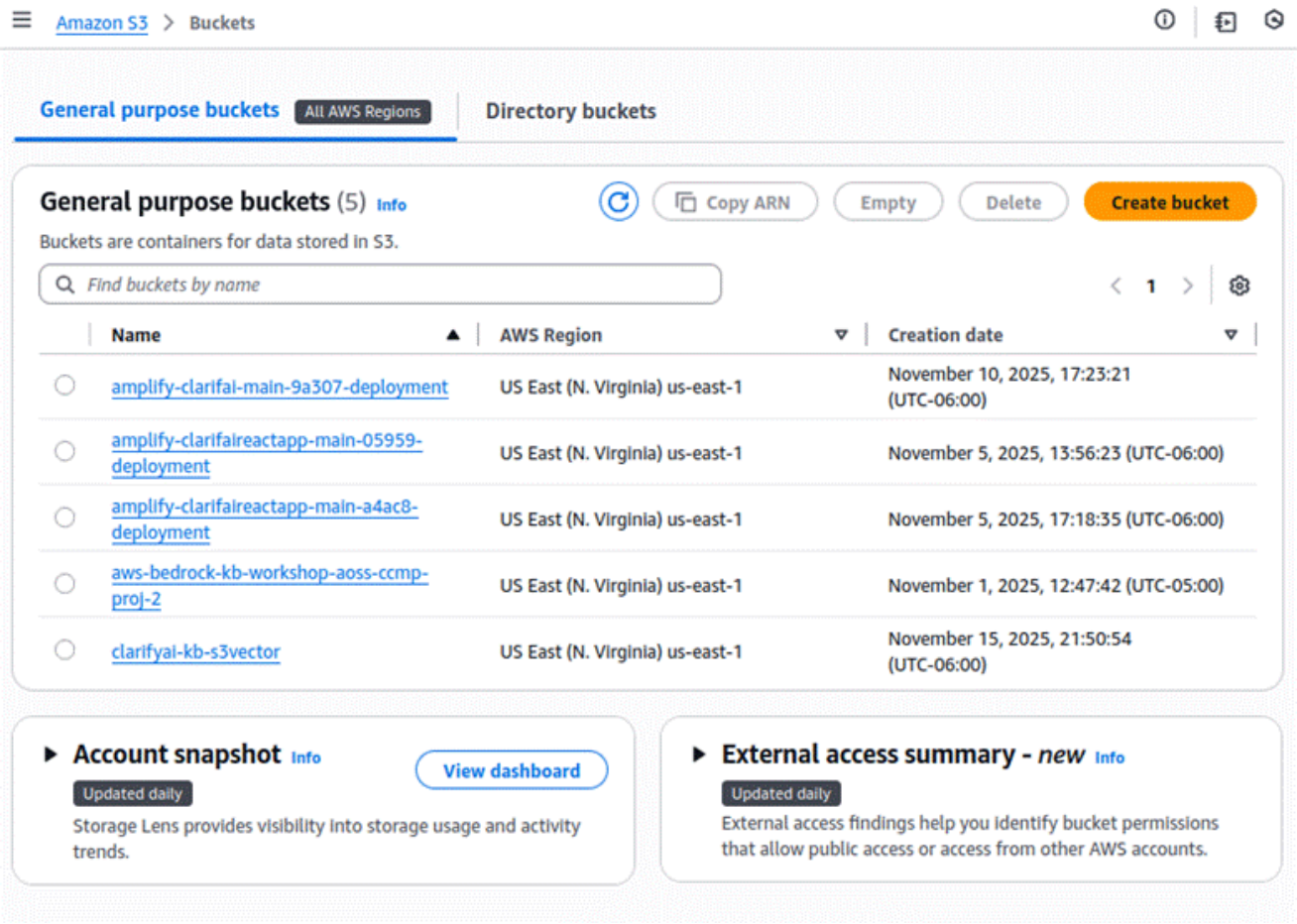
Configuration (summary):

- Bucket name: clarifyai-kb-s3vector
- Region: us-east-1
- Versioning: Disabled (for this project).

- Default encryption: **SSE-S3** with bucket key enabled.
- Public access: **Block all public access = On**.
- Bucket policy: Grants controlled s3:PutObject access for Bedrock log delivery to a specific prefix and account, plus Admin upload permissions via IAM roles.

Documents uploaded by Admin are stored under a documents/ prefix, with metadata (original filename, timestamp) added by DocumentUploadHandler.





10. Security and IAM

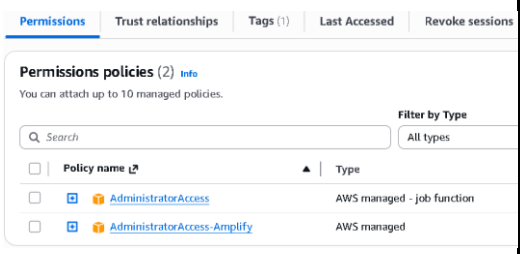
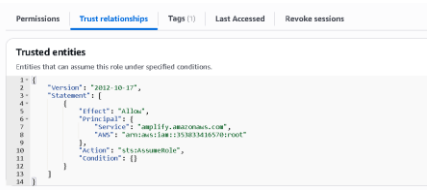
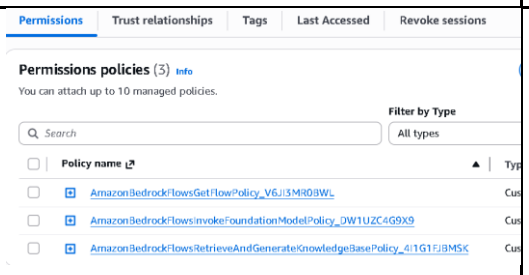
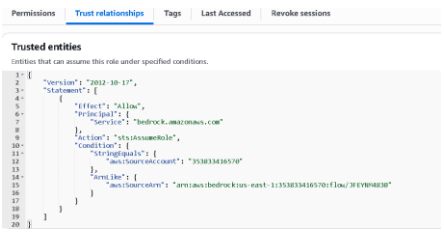
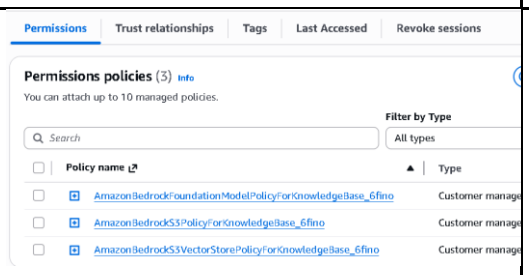
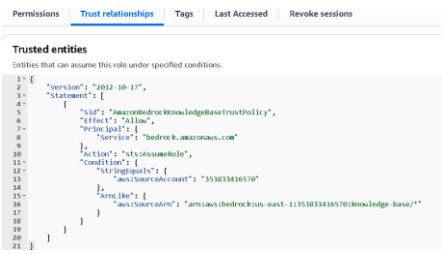
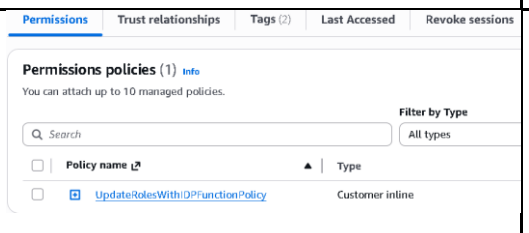
ClarifyAI answers policy-relevant questions and exposes internal documents. We must ensure that only approved users access the app, only Admins can change the corpus, and every API call is authenticated and auditableAWS Identity and Access Management (IAM) defines who can do what on which resources. We couple IAM roles/policies with Cognito groups and an Identity Pool so Admins receive temporary S3 write permissions while Employees do not. API Gateway fronts the API with a Cognito authorizer that validates ID tokens.

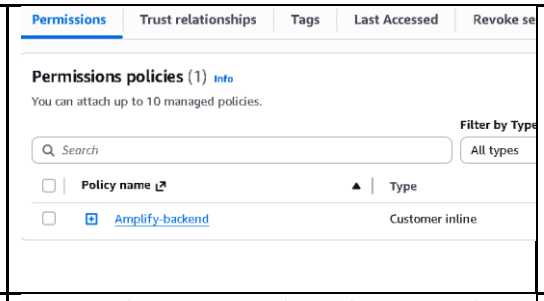
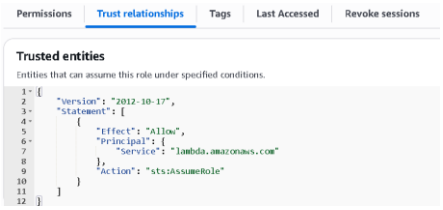
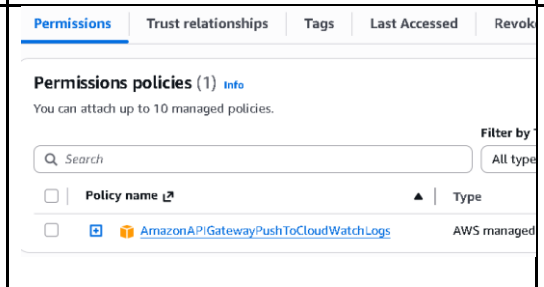
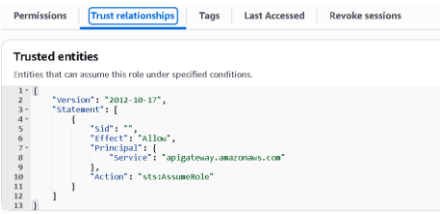
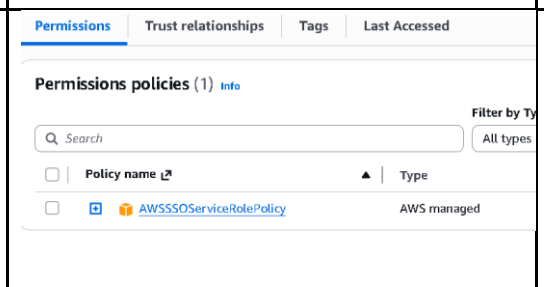
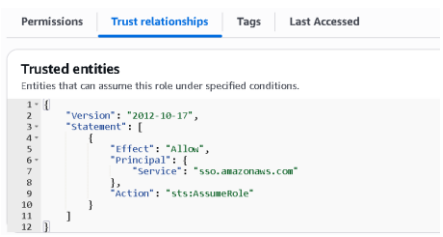
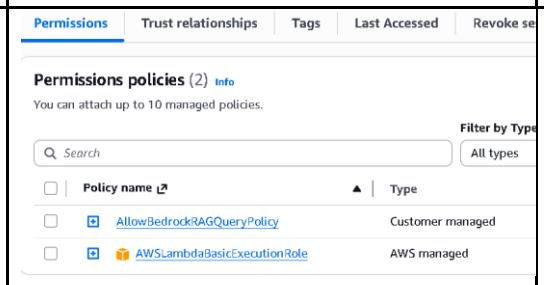
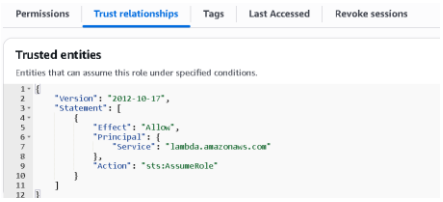
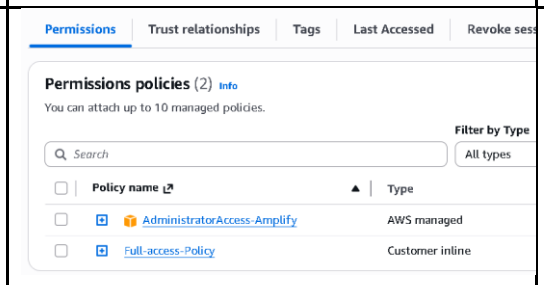
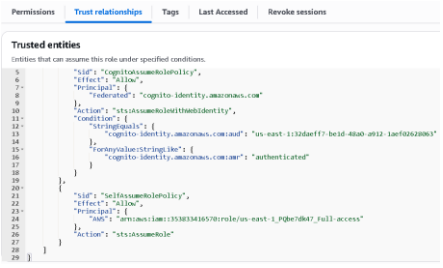
10.1. Controls Implemented

- **Least privilege** – Dedicated roles for Lambda execution, Knowledge Base access, Bedrock flows, API Gateway logging, and S3 operations.
- **Group-based access** – Two Cognito Groups (`Admin`, `User`) clearly define roles. The UI and backend permissions are driven by group membership.
    - *Admins* – upload + query
    - *Users* – query-only

- **Credential isolation** – Identity Pool role mapping grants temporary S3 write permissions only to Admins.
- **API safeguards** – API Gateway Cognito authorizer validates tokens on every request; unauthorized calls return 401/403.
- **Data protection** – S3 server-side encryption by default; TLS in transit; CloudTrail for audit logs.

Table 1: IAM Roles and Policies Summary

Name	Type	Policies	Trusted entities
Admin	Role		amplify.amazonaws.com 
AmazonBedrockExecutionRoleForFlows_WJPB2S22FY	Role		bedrock.amazonaws.com 
AmazonBedrockExecutionRoleForKnowledgeBase_6fino	Role		bedrock.amazonaws.com 
amplify-clarifaireactapp-main-a4ac8-authRole-idp	Role		lambda.amazonaws.com 

amplify-login-lambda-f6e9f59f	Role		lambda.amazonaws.com 
APiGatewayLog_CloudWatch	Role		apigateway.amazonaws.com 
AWSServiceRoleForSSO	Role		sso.amazonaws.com 
RAGQueryLambdaExecutionRole	Role		lambda.amazonaws.com 
us-east-1_PQbe7dk47_Full-access	Role		cognito-identity.amazonaws.com 

11. CI/CD via Amplify

ClarifyAI uses Amplify Hosting to automate builds and deployments from the Git repository. When code is committed:

- **Git Commit & Push** – Developers push changes to the main or feature branch.
- **Amplify Build** – Amplify runs automated build steps (install, test, build React app).
- **Deploy** – Successful builds are deployed to a CloudFront-backed domain with HTTPS.
- **Rollbacks** – If needed, previous builds can be restored with zero downtime.

12. Outcomes

- Built a functioning RAG-based web application where Admins upload documents and Employees query them through a browser.
- Achieved **citation-backed answers** grounded only in S3-hosted company documents.
- Demonstrated **secure, role-aware access** using Cognito groups and IAM S3 permissions.
- Reduced vector database cost by switching from OpenSearch to S3 Vector Store Serverless while keeping the architecture mostly unchanged.

13. Challenges and Solutions

- **Complex IAM configurations** – Many services (Bedrock, S3, Lambda, API Gateway, Amplify) required explicit roles and trust policies.
    - *Solution:* Designed roles up front and used tables/runbooks to track each permission.
- **Vector store cost management** – Persistent OpenSearch Serverless deployments accumulated cost even when idle.
    - *Solution:* Migrated to S3 Vector Store, which charges based on storage and per-query operations, ideal for “cold” corpora.
- **End-to-end integration** – Wiring Cognito tokens through Amplify, API Gateway, and Lambda while debugging CORS and auth errors.
    - *Solution:* Tested layers incrementally (Lambda → API Gateway; then Cognito authorizer; then React integration) and used CloudWatch logs for error tracing.

15. Lessons Learned

- The AWS project required us to setup and connect many different AWS services to build a complete online accessible platform where employees can query the documents that the admin uploads and syncs.
- This project gave us the opportunity to understand how different microservices like knowledge bases, vector databases and S3 storage are interrelated and can't be setup up as standalone units.
- The Bedrock Flow is a great tool to chain prompts, context, knowledge bases and evaluation in an easy-to-understand flowchart type graphical UI.
- The role and permission-based AWS architecture was very well evident across the entire project as almost every aspect of the AWS microservices we setup or used (even AWS terminal access) needed explicit IAM roles and policies associated with each microservice. This results in even more roles than we had initially planned as needed for the setup and operation of the project.
- We had to switch from the AWS OpenSearch vector database and index mid-project because the cost accumulated during persistent long term deployment saw cost accumulating significantly, sometimes even exceeding the AWS free credits (upto $200). This was because unlike S3, OpenSearch deployment was charged for hours and vector search calls like EC2 unlike utility-based pricing based on number of calls and storage sizes like S3. This resulted in cost adding over multiple days of online vector database deployment despite not actively using it.
- We switched to AWS S3 Vector Store (Preview) specifically to overcome this cost overrun despite not actively using the website to query the documents. S3 Vector Store (Preview) is designed to be a much cheaper, "object-storage-style" pay-per-GB option with storage around S3 rates plus per-write and per-query charges. This can be order of magnitude cheaper than running a dedicated vector database or OpenSearch for large, mostly-cold RAG corpora that isn't frequently used.
- S3 Vectors is cost-optimized for massive embeddings you query relatively infrequently, with pros of low storage cost and minimal ops but cons of slower queries and less sophisticated search and analytics.
- AWS Amplify greatly simplified the utilization of AWS Cognito for maintaining the userbase, authentication and security of user and admin login. Just connect the correct Cognito userpool in the Amplify deployment and then we can directly interact with the userpool in the React app by importing the required libraries. Authentication tokens are automatically handled by Amplify, simplifying implementation and execution.

15. Future Roadmap

Planned enhancements include:

- Automated KB Sync triggers on S3 upload events.
- Multi-language support for global teams.
- Fine-grained document-level permissions (per-team or per-department access).
- Integrated cost-anomaly detection and alerts.
- Richer analytics and search insights for admins.
- Integrations with collaboration tools such as Slack or Microsoft Teams.

16. Conclusion

ClarifyAI demonstrates a robust serverless architecture that effectively addresses the critical challenge of providing trustworthy, citation-backed answers from controlled corporate knowledge bases. By leveraging AWS services including Bedrock Knowledge Bases, S3 Vector Store Serverless, Lambda, API Gateway, Cognito, and Amplify, the system implements a complete Retrieval-Augmented Generation (RAG) pipeline with role-based access control. The architecture separates concerns across distinct layers like storage, retrieval, application, and presentation ensuring security through least-privilege IAM policies, group-based permissions, and API token validation. This approach eliminates common enterprise search pitfalls such as prompt fragility, hallucinations, and lack of provenance while maintaining low operational overhead through serverless components and managed services.

The project yielded valuable insights into real-world cloud architecture trade-offs, particularly the significant cost differential between persistent vector databases like OpenSearch and usage-based alternatives like S3 Vector Store, which proved essential for cost-effective deployment of infrequently-queried knowledge bases. The team discovered that AWS IAM's permission architecture requires more granular role definitions than initially anticipated, with explicit policies needed across every microservice interaction. Future improvements could include expanded file format support, incorporate in-document citations, enhanced search analytics, multi-language capabilities, and integration with additional enterprise tools like Slack or Microsoft Teams. Potential use cases extend beyond internal Q&A to compliance documentation systems, customer support knowledge bases, onboarding assistants, legal research tools, and regulatory policy guidance platforms. Essentially, any scenario where organizations need to provide accurate, auditable answers derived exclusively from approved sources while maintaining clear access controls and complete transparency through citations.

Appendix – Technology & Service Guide

A. Amazon S3

Amazon S3 is the storage backbone of ClarifyAI. All approved documents (policies, SOPs, PDFs, DOCX) are stored in a dedicated bucket and treated as the single source of truth for the knowledge base. Bedrock Knowledge Bases read from this bucket during ingestion, and no other storage location is considered authoritative. IAM policies enforce that only Admin users (via Cognito roles) and Bedrock/Lambda service roles can write or read from this bucket.

Setup (ClarifyAI-specific)

1. In the S3 console, create a bucket such as clarifyai-docs-<env>.
2. Turn on **Block public access** for all options.
3. Enable **default encryption** (SSE-S3 or SSE-KMS).
4. Create folder prefixes, for example:
    a. documents/ – raw PDFs/DOCX that the KB ingests.
    b. kb-artifacts/ – optional folder for logs or exports.
5. Create an IAM policy that:
    a. Allows s3:GetObject on documents/* for Knowledge Base and query Lambda roles.
    b. Allows s3:PutObject to documents/* only for the Admin Cognito role and upload Lambda.
6. Attach this policy to the relevant IAM roles (CognitoAdminIAMRole, BedrockKnowledgeBaseServiceRole, upload Lambda role).
7. Upload a few sample documents into documents/ to test end-to-end ingestion.

B. S3 Vector Store Serverless

S3 Vector Store Serverless acts as ClarifyAI's semantic index. When the Knowledge Base syncs, it creates embeddings for document chunks and stores them in this vector store. Bedrock uses this index at query time to find the most relevant passages for a question before the model generates an answer.

Setup (ClarifyAI-specific)

1. From the **Amazon Bedrock console**, go to **Vector Stores** (S3 Vector Store Serverless) and create a new vector store.
2. Choose or create an S3 bucket/prefix where vector metadata and index files will live (e.g., clarifyai-vector-store/).
3. Assign or create a service role that grants:
    a. s3:GetObject, s3:PutObject, s3:ListBucket on the chosen bucket/prefix.
4. Note the **Vector Store ID**; you will reference it when configuring the Knowledge Base.

5. After connecting it to the KB and running a sync, verify via the console that vectors are populated and the status is **Active/Ready**.

C. Amazon Bedrock Knowledge Base

The Knowledge Base is the managed retrieval layer for ClarifyAI. It knows which S3 bucket contains the documents, how to chunk them, which embedding model to use, and which vector store to write to. At query time, ClarifyAI's Flow calls this KB to retrieve top-k passages that are most relevant to the user's question.

Setup (ClarifyAI-specific)

1. In the Bedrock console, navigate to **Knowledge bases → Create knowledge base**.
2. Create or choose a **service role** (e.g., BedrockKnowledgeBaseServiceRole) with permissions to:
    a. Read from the S3 documents bucket (documents/).
    b. Read/write to the S3 Vector Store bucket.
3. Configure a **data source**:
    a. Type: **Amazon S3**.
    b. Bucket: your ClarifyAI docs bucket.
    c. Prefix: documents/.
4. Select an **embeddings model** (e.g., Titan Embeddings V2) appropriate for your language/content.
5. Attach the previously created **S3 Vector Store Serverless** as the vector index.
6. Run an initial **Sync**. When it completes, check that:
    a. Sync status is **Complete/Ready**.
    b. Your documents are visible as ingested items.

Admins will re-trigger this sync when new documents are added.

D. Amazon Bedrock Foundation Models & Flows

ClarifyAI uses a Bedrock Flow to combine retrieval and generation into a single "retrieve-and-generate" pipeline. The Flow receives a question, calls the Knowledge Base to fetch relevant chunks, passes those chunks plus the question into a foundation model, and returns a final answer plus metadata.

Setup (ClarifyAI-specific)

1. In the Bedrock console, first **enable** the foundation model you plan to use (e.g., Nova or Anthropic Claude).
2. Go to **Flows → Create flow** and design a simple flow:
    a. **Input node** with a single field question.

b. **Knowledge Base node** referencing your ClarifyAI KB and configured to return top-k passages.
c. **Model node** with:
    i. System prompt describing ClarifyAI's behavior (only answer from provided context, show citations, etc.).
    ii. Placeholders for question and retrieved context.
d. **Output node** that returns a JSON object with answer and structured citations.

3. Deploy the flow and note the **Flow ARN**.
4. Grant your Lambda execution role permission to invoke this flow via Bedrock.

The Lambda function now only needs to pass the user's question and read back the flow's structured output.

E. AWS Lambda (Python 3.11)

Lambda hosts ClarifyAI's backend code. One function handles user queries, calling Bedrock Flow and shaping the HTTP response. A second function (optional) handles document uploads and Knowledge Base sync tasks.

Setup (ClarifyAI-specific)

1. Create a function **clarifyai-query-lambda** with runtime **Python 3.11**.
2. Assign an execution role that allows:
    a. bedrock:InvokeFlow (for your Flow ARN).
    b. logs:CreateLogGroup, logs:CreateLogStream, logs:PutLogEvents (CloudWatch).
3. Implement the handler to:
    a. Read the incoming event body (e.g., { "question": "..." }).
    b. Call Bedrock Flow (using boto3's bedrock-agent-runtime / bedrock-runtime client).
    c. Parse the flow result (answer text, citation metadata).
    d. Return an HTTP response structure recognized by API Gateway (statusCode, headers with CORS, body).
4. Test the Lambda locally with a sample event to ensure Bedrock integration works.
5. (Optional) Create a second Lambda for **uploads** that writes to S3 and starts a KB sync job via the Bedrock API.

F. Amazon API Gateway

API Gateway exposes ClarifyAI's backend as HTTP endpoints to the React frontend. It validates Cognito tokens and forwards only authorized requests to Lambda.

Setup (ClarifyAI-specific)

1. In API Gateway, create a new **HTTP API** or **REST API** (HTTP API is simpler).
2. Add a **POST /query** route and set its integration target to clarifyai-query-lambda.
3. Configure a **Cognito authorizer** pointing to your ClarifyAI User Pool.
4. Require the authorizer on /query so calls must include a valid Authorization: Bearer <ID token> header.
5. Enable **CORS**:
    a. Allowed origin: your Amplify app URL.
    b. Methods: POST, OPTIONS.
    c. Headers: Authorization, Content-Type.
6. Deploy the API to a stage (e.g., prod) and copy the endpoint URL for your frontend configuration.

G. Amazon Cognito (User Pool & Identity Pool)

Cognito handles user sign-in and group membership for ClarifyAI. It also issues JWT tokens (ID/Access) that the frontend passes to API Gateway, and temporary AWS credentials that map Admins to S3 write permissions.

Setup (ClarifyAI-specific)

1. Create a **User Pool** (email as login).
2. Add an **App Client** (no secret for SPA) and enable the flows you need (e.g., ALLOW_USER_PASSWORD_AUTH, ALLOW_REFRESH_TOKEN_AUTH).
3. Configure a **Hosted UI domain** for the sign-in page if you use the built-in UI.
4. Under **User Pool → Groups**, create:
    a. Admins – for users who can upload/sync documents.
    b. Users – for employees who can only query.
5. Create an **Identity Pool** and select “Use Cognito User Pool”.
6. Define two IAM roles:
    a. CognitoAdminIAMRole – has S3 write permissions to the docs bucket.
    b. CognitoUserRole – read/query only.
7. In Identity Pool **role mapping**, map the Admins group to CognitoAdminIAMRole and default users to CognitoUserRole.
8. In Amplify/React, configure Cognito IDs (aws_user_pools_id, aws_user_pools_web_client_id, identity pool ID) so authentication works end-to-end.

H. AWS Amplify Hosting

Amplify Hosting serves the React application and provides automatic builds on every git push. It is the entry point for both Admin and Employee UIs.

Setup (ClarifyAI-specific)

1. Push your React project to GitHub or another supported repo.
2. In the Amplify console, choose **Host web app → Connect app** and link the repo/branch.
3. Review the generated buildspec (it should run npm ci / npm install and npm run build).
4. Add **environment variables**:
    a. REACT_APP_API_URL – API Gateway endpoint.
    b. REACT_APP_REGION, REACT_APP_USER_POOL_ID, etc.
5. Configure **rewrites & redirects** for SPA (all paths → /index.html).
6. Deploy and note the hosted URL (e.g., https://main.<id>.amplifyapp.com).
7. Add this domain to:
    a. API Gateway CORS allowed origins.
    b. Cognito App Client redirect URLs (for Hosted UI).

I. React (Frontend Application)

The ClarifyAI UI is built as a React SPA. It provides two main experiences: an Admin dashboard and a simplified Q&A view for employees. Routing is generally role-based; the same codebase renders different components depending on the user's Cognito group.

Setup (ClarifyAI-specific)

1. Initialize the project with Create React App or Vite.
2. Install required dependencies:
3. npm install aws-amplify @aws-amplify/ui-react
4. Create top-level components:
    a. AdminDashboard – upload widget, document table, sync button, test query area, session history.
    b. UserChat – question input, "Ask" button, answer panel with citations, session history.
5. Use React Router or simple conditional rendering:
    a. After sign-in, check user.signInUserSession.idToken.payload["cognito:groups"] to decide whether to show Admin or User view.

J. Amplify JS & Amplify UI (aws-amplify, @aws-amplify/ui-react)

Amplify's JS and UI packages glue the frontend to Cognito and API Gateway. They take care of the heavy lifting for authentication and HTTP requests.

Setup (ClarifyAI-specific)

1. Import and configure Amplify in your React entry file:

```
import { Amplify } from 'aws-amplify';
import awsExports from './aws-exports';
```

```
Amplify.configure(awsExports);
```

2. Wrap the app with the **Authenticator** UI component:

```
import { Authenticator } from '@aws-amplify/ui-react';

function App() {
 return (
  <Authenticator>
   {({ signOut, user }) => (
    <MainApp signOut={signOut} user={user} />
   )}
  </Authenticator>
 );
}
```

3. For calling the backend, configure an Amplify API or use the raw endpoint:

```
import { API } from 'aws-amplify';

async function askQuestion(question) {
 const res = await API.post('clarifyApi', '/query', {
  body: { question },
 });
 return res;
}
```

4. Use Amplify UI components (buttons, layout, etc.) to maintain consistent styling and built-in handling for loading/error states.