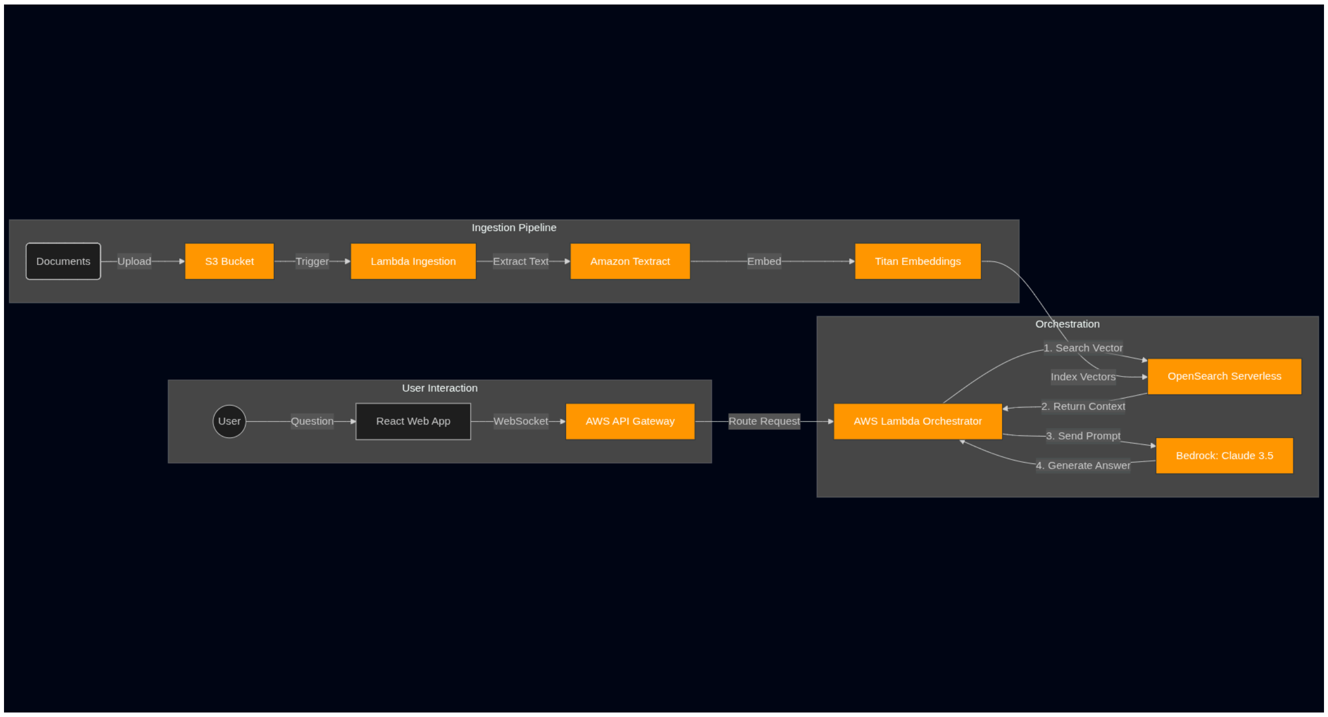
Cloud Architect Solution: Internal AI Knowledge System

1. Assumptions

To design a system optimized for performance and cost, I have made the following assumptions:

- **Document Volume:** Approximately 100,000 existing documents (PDF, Word, PPT) averaging 5MB each, with ~100 new documents added daily.
- **User Access:** 500 total employees, but "500 concurrent users" implies a peak usage scenario (e.g., Monday morning reporting) where everyone is online simultaneously.
- **Data Structure:** Documents reside in existing repositories (SharePoint/Drive) and have existing Access Control Lists (ACLs) that must be preserved.
- **Latency Target:** Users expect an answer within 5-8 seconds.

2. High-Level Architecture



Data Flow Summary:
The architecture uses an Event-Driven Serverless model on AWS.

1. **Ingestion Track:** Documents uploaded to **Amazon S3** trigger an **AWS Lambda** pipeline. **Textract** extracts text, which is chunked and embedded by **Amazon Titan**, then stored in

   OpenSearch Serverless.
2. **Query Track:** Users interact via a **React Web App** (hosted on Amplify). API Gateway routes the request to Lambda, which converts the question to a vector, retrieves relevant chunks from OpenSearch, and generates an answer using **Claude 3.5 Sonnet** on **Amazon Bedrock**.

3. Ingestion and Indexing Pipeline

This pipeline ensures documents are searchable within minutes of upload.

- **Collection:** A connector script pulls documents from the firm's file servers and uploads them to an **S3 Bucket (Ingestion Zone)**.
- **Conversion (OCR):** An S3 Event Notification triggers a Lambda function. If the file is a scanned PDF or image, **Amazon Textract** is used to extract raw text. Standard Word/PPT files are parsed using Python libraries.
- **Chunking & Embedding:**
  - **Chunking:** Text is split into 500-token chunks with a 10% overlap to preserve context.
  - **Embedding:** Chunks are sent to **Amazon Titan Text v2** to generate vector embeddings.
- **Storage:** Vectors are indexed in **Amazon OpenSearch Serverless**. Crucially, we store metadata with every chunk: { "source": "project_alpha.pdf", "allowed_groups": ["HR", "Managers"] }.

4. RAG Retrieval + Response Logic

We prioritize accuracy and security over raw speed.

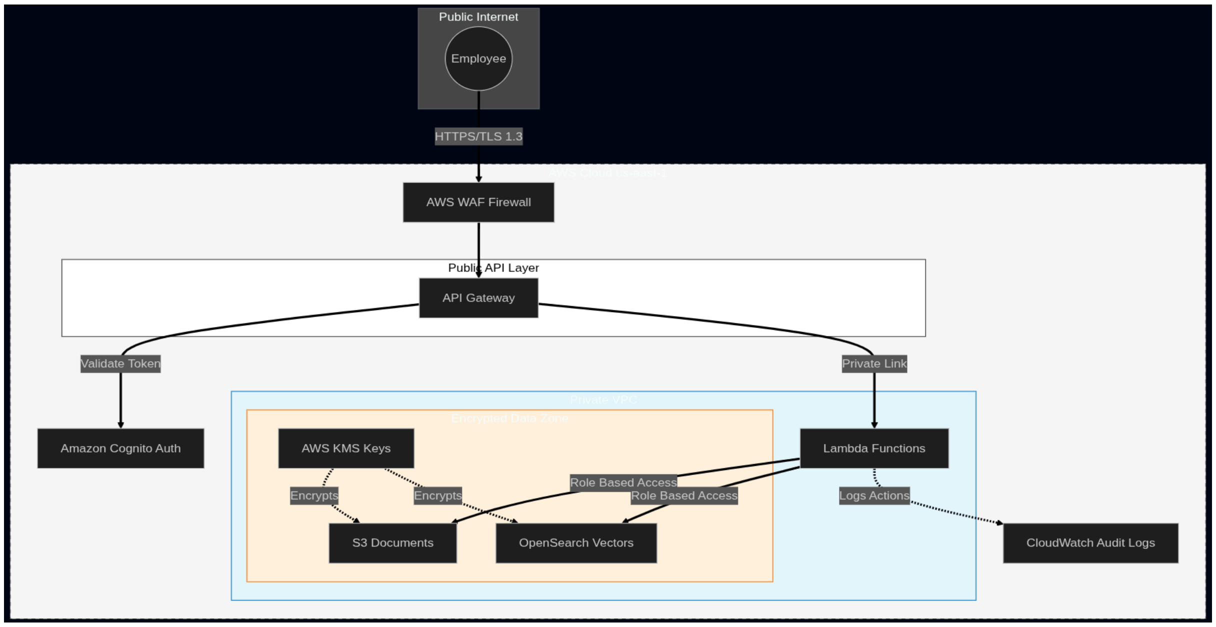
- **Vector DB: Amazon OpenSearch Serverless** (k-NN search).
- **Embedding Model: Amazon Titan Text v2** (Cost-effective and optimized for RAG).
- **Retrieval Strategy:**
  - **Hybrid Search:** We perform a vector search (for meaning) AND a keyword search (for specific project names).
  - **Security Filtering:** The query includes a hard filter matching the user’s role (e.g., filter: { user_groups: ["Junior_Associate"] }). This ensures users never retrieve chunks they aren't allowed to see.
  - **k-Value:** We retrieve the top k=10 chunks.
- **Generation:** The top 5 filtered chunks are fed into **Claude 3.5 Sonnet** with a system prompt enforcing citation: *"Answer solely based on the provided context and cite the document name for every fact."*

5. User Interface + Application Layer

- **Frontend:** A clean interface built with **React** and deployed via **AWS Amplify**. It supports real-time streaming of the answer so users don't stare at a loading spinner.

- **Backend: Amazon API Gateway** (WebSocket API) maintains a persistent connection to handle the streaming response.
- **Orchestration: LangChain** (running on Lambda) manages the conversation history and prompt engineering logic.

6. Security Architecture



- **Authentication: Amazon Cognito** integrated with the firm's Corporate Active Directory (SSO). Only employees with valid corporate email can log in.
- **Authorization:** implemented at the **Database Row Level**. Every document chunk in OpenSearch has an Access Control List (ACL) field. Queries are automatically filtered by the user's JWT token claims.
- **Encryption:**
  - **At Rest:** S3 buckets and OpenSearch indices are encrypted using **AWS KMS** with Customer Managed Keys (CMK).
  - **In Transit:** TLS 1.3 is enforced for all API endpoints.
- **Audit Logging: AWS CloudTrail** logs all infrastructure changes. A separate "Query Log" (stored in CloudWatch) records *who* asked *what* and *which* documents were accessed, satisfying compliance.

7. Scaling Strategy

To handle 500 concurrent users:

- **Compute: AWS Lambda** is configured with Provisioned Concurrency during business hours (8 AM - 6 PM) to eliminate cold starts.

- **Database: OpenSearch Serverless** automatically scales compute units (OCUs) for search independently of ingestion. Heavy querying will not slow down document indexing.
- **Rate Limiting:** API Gateway implements per-user throttling (e.g., 50 requests/minute) to prevent abuse.

8. Cost Strategy

Target: < $8,000/month.

- **Model Optimization:** Simple queries will be routed ("What is the holiday policy?") to **Claude 3 Haiku** (cheaper/faster) and complex queries ("Analyze risk in these 5 contracts") to **Claude 3.5 Sonnet**.
- **Storage Tiering:** Documents older than 3 years are moved to **S3 Infrequent Access (IA)**, saving 40% on storage.
- **Vector Storage:** We use OpenSearch "Warm" nodes for older project archives, keeping only active projects in "Hot" memory.
- **Dev Costs:** Development environments use **LocalStack** to mock AWS services, reducing cloud spend to near zero during coding.

9. Risks, Tradeoffs, and Alternatives

- **Optimized For: Security and Compliance**. OpenSearch was chosen over cheaper options (like Pinecone) to keep data strictly inside the AWS VPC boundary.
- **Deprioritized: Global Latency**. All infrastructure is in us-east-1 (US Data Residency). International employees may experience slightly higher latency, but this is a necessary tradeoff for compliance.
- **Future Improvement:** Implement a "Feedback Loop" (Thumbs Up/Down) in the UI to fine-tune retrieval relevance using Reinforcement Learning (RLHF).