

Cloud Architecture Design: Enterprise RAG System

1. Assumptions

- **Data Volume:** 10 years of documents implies ~1-5 million documents (PDFs, Word, Emails). Estimated storage: 5-10 TB.
- **Update Frequency:** Daily batch updates (new documents added daily). Real-time ingestion not strictly required but preferred.
- **User Base:** 500+ concurrent users (employees). Internal corporate network access.
- **Compliance:** Data must remain within the corporate VPC or secure cloud environment. SOC2/GDPR compliance likely required.

2. High-Level Architecture

```
graph TD
    subgraph "Client Layer"
        UI[Web UI / Chat Interface]
    end

    subgraph "API Gateway & Security"
        WAF[AWS WAF]
        ALB[Application Load Balancer]
        Auth[Cognito / SSO]
    end

    subgraph "Application Layer (EKS/ECS)"
        API[RAG API Service]
        Ingest[Ingestion Worker]
    end

    subgraph "Data & Storage Layer"
        S3[S3 Bucket (Raw Docs)]
        VDB[(Vector DB - Pinecone/Milvus)]
        RDS[(PostgreSQL - Metadata/Chat History)]
        Cache[(Redis - Semantic Cache)]
    end
```

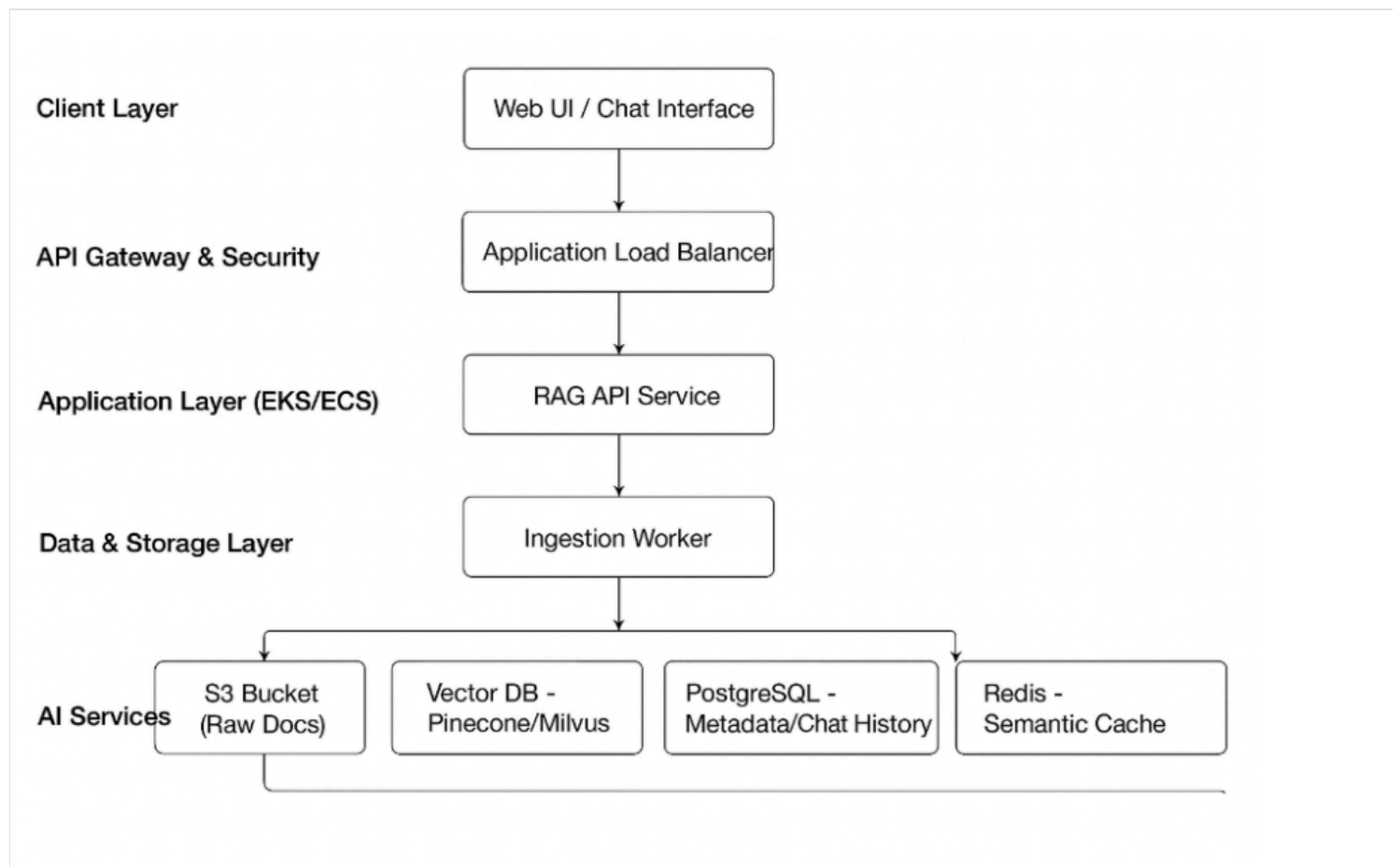
```
subgraph "AI Services"
    LLM[LLM (Bedrock/OpenAI)]
    Embed[Embedding Model]
end

UI --> WAF
WAF --> ALB
ALB --> API
API --> Auth

API --> Cache
API --> Embed
API --> VDB
API --> LLM
API --> RDS

S3 --> Ingest
Ingest --> Embed
Ingest --> VDB
```

Visual Architecture Diagram



3. Ingestion and Indexing Pipeline

1. **Collection:** Documents are uploaded to an **S3 Bucket** (Raw Zone). S3 Event Notifications trigger the pipeline.
2. **Preprocessing:** A Lambda function or Fargate task picks up the file.
 - **OCR:** Use **AWS Textract** or **Unstructured.io** to extract text from PDFs/Images.
 - **Cleaning:** Remove PII (using Macie or regex), headers, and footers.
3. **Chunking:** Split text into chunks of 512-1024 tokens with 10% overlap.
4. **Embedding:** Pass chunks to an Embedding Model (e.g., **Titan Embeddings** or **OpenAI text-embedding-3-small**).
5. **Indexing:** Store vectors in **Pinecone** (managed) or **Milvus** (self-hosted) with metadata (doc_id, author, date, s3_link).

4. RAG Retrieval + Response Logic

1. **Query Analysis:** User query is rewritten to optimize for search (e.g., removing stop words, expanding acronyms).
2. **Hybrid Retrieval:**
 - **Vector Search:** Find top 20 chunks based on semantic similarity (Cosine Similarity).
 - **Keyword Search:** Use BM25 (via OpenSearch or Pinecone's hybrid mode) for exact matches (e.g., "Project Alpha").
3. **Re-ranking:** Pass the top 20 results to a **Cross-Encoder (Cohere Rerank)** to select the top 5 most relevant chunks.
4. **Generation:** Construct a prompt with the top 5 chunks as context.
 - *System Prompt:* "Answer based ONLY on the context. Cite sources using [Doc ID]."
5. **Citations:** The UI renders the [Doc ID] as a clickable link to the S3 presigned URL.

5. User Interface + Application Layer

- **Frontend:** React/Next.js application hosting a Chat Interface.
- **Backend:** Python (FastAPI) service.
- **API Gateway:** Manages rate limiting and routing.
- **Streaming:** Responses are streamed to the client (Server-Sent Events) for low latency perception.

6. Security Architecture

```

graph TD
    User -->|HTTPS| CloudFront
    CloudFront -->|WAF Rules| ALB
    ALB -->|Private Subnet| ECS_Cluster

    subgraph "VPC (Virtual Private Cloud)"
        subgraph "Public Subnet"
            ALB
            NAT[NAT Gateway]
        end

        subgraph "Private Subnet"
            ECS_Cluster[API Containers]
            Lambda[Ingestion Functions]
        end

        subgraph "Data Subnet (Isolated)"
            RDS
            VPC_Endpoint[VPC Endpoints for S3/Bedrock]
        end
    end

    ECS_Cluster -->|IAM Role| VPC_Endpoint

```

- **Authentication:** Integration with Corporate SSO (Okta/Azure AD) via AWS Cognito.
- **Authorization: Document-Level Security (DLS).** Each vector in the DB stores an `access_group` field. Queries filter results by the user's group (e.g., `filter={group: "HR"}`).
- **Encryption:** AES-256 for S3/RDS (At Rest). TLS 1.3 for all transit.
- **Network:** API and DB sit in Private Subnets. No direct internet access.

7. Scaling Strategy

- **Ingestion:** Event-driven (SQS + Lambda). Scales to zero when idle; scales up to thousands of concurrent Lambda executions during bulk uploads.
- **Retrieval:** The Vector DB (e.g., Pinecone) is managed and auto-scales based on QPS (Queries Per Second).
- **API Layer:** Run on AWS Fargate (Serverless Containers) with Auto Scaling based on CPU/Memory usage.
- **Caching:** Use Redis to cache frequent queries (Semantic Caching). If a user asks "What is the vacation policy?" twice, the second answer comes from Cache (0ms latency).

8. Cost Strategy

- **Compute:** Use **Spot Instances** for the Ingestion workers (fault-tolerant). Use **Savings Plans** for the API Fargate tasks.
- **Storage:** Use **S3 Intelligent-Tiering** to move old documents to cheaper storage classes.
- **LLM:** Use **Bedrock** (pay-per-token) instead of hosting 24/7 GPU instances. Use smaller models (Haiku/GPT-3.5) for simple queries and larger ones (Sonnet/GPT-4) only for complex reasoning.

9. Risks and Tradeoffs

- **Tradeoff: Managed vs. Self-Hosted.** We chose Managed Services (Pinecone, Bedrock) to reduce operational overhead, even though it costs slightly more than self-hosting.
- **Risk: Hallucinations.** Even with RAG, the LLM might lie. *Mitigation:* Strict system prompts and UI warnings.
- **Risk: Latency.** Reranking adds latency. *Mitigation:* Only rerank for complex queries.