# **RAG System Architecture Report**

**Project:** Internal AI-powered Knowledge Management (RAG)  
 **Customer:** Mid-size consulting firm (~500 employees)  
 **Constraints:** data must reside in US regions; initial budget $8,000/month; handle 500 concurrent users; 99.5% uptime; respect document permissions & audits.  
 **Author:** Architecture Team — delivered for Manish jangir

## **1. Executive Summary**

Design a secure Retrieval-Augmented Generation (RAG) system to let employees query the firm’s historical documents in natural language, returning answers with **source citations** while enforcing **document-level permissions**, **audit logs**, and **US-only data residency**. The recommended approach uses a managed ingestion pipeline, a vector database for embeddings, a retrieval layer, an LLM (initially hosted via a vetted API or in-VPC managed inference), and an authenticated frontend.

## **2. High-Level Architecture (data flow)**

[Document Sources] --> [Ingestion Workers] --> [Text Extraction & NLP]

|

v

[Embeddings Service / Embedder]

|

v

[Vector DB (k-NN) Cluster]

|

v

[Authenticated User Query] -> [API Gateway] -> [RAG Orchestrator]

| \

v \

[Retriever (k-NN search)] [Auth & Perms]

| /

v /

[Context Assembler & Reranker] <----

|

v

[LLM / Generation Engine (RAG)]

|

v

[Response + Citations]

|

v

[Audit Log] & [Access Controls Enforced]

Key choices:

* Ingestion handles PDF, Word, PPT, email; extracts text, segments, adds metadata (source, doc-id, section, permissions).
* Each text chunk receives an embedding stored in Vector DB with metadata for permission checks.
* Retrieval returns context+document IDs; orchestrator filters by permission, assembles context, calls LLM, returns answer with citations.

## **3. Technology Stack (recommended, AWS-focused to meet US-residency)**

**Note:** pick a single cloud region (e.g., us-east-2 or us-west-2) for all services and enforce region constraints in IAM & KMS.

* **Compute / Orchestration**
  + AWS ECS Fargate or EKS (managed) for ingestion workers and containers.
  + AWS Lambda for lightweight connectors (webhooks, event triggers).
* **Storage**
  + Amazon S3 (US region) for raw documents + signed URLs; S3 object metadata holds doc-id & coarse permission tags.
  + AWS RDS (Postgres) (multi-AZ) for metadata/catalog (document records, user mapping, permission model).
* **Vector Database / k-NN**
  + Options:  
    - **Pinecone** (managed) — fast, simple, but must confirm US-residency & enterprise contract.
    - **AWS OpenSearch with k-NN plugin** — can run in VPC, US-region, simpler for compliance.
    - **Milvus on EKS** — self-managed, cost-effective at scale.
  + Recommendation for MVP: **Pinecone** (if contract ensures US-hosting) or **OpenSearch k-NN** if you need everything in-VPC.
* **Embeddings & LLM**
  + **Option A (Faster MVP):** Use enterprise LLM provider (OpenAI/Anthropic) with enterprise contract guaranteeing US-hosting, and use their embeddings API + model API.
  + **Option B (Data-residency-first):** Self-host open weights (Llama 2, Mistral, etc.) on EC2 GPU + Triton / containerized inference inside a secured VPC.
  + **Recommendation:** Start with hosted embeddings (enterprise contract) + hosted LLM for rapid launch; plan Phase-2 self-hosting if cost or residency demands it.
* **API / Gateway / Auth**
  + AWS API Gateway + ALB for endpoints.
  + Use SSO: Okta / Azure AD / AWS IAM Identity Center — integrate SAML/OIDC for company-only access.
* **Security & Encryption**
  + AWS KMS (regional) for envelope encryption of S3/RDS.
  + Private VPC endpoints for S3, OpenSearch, and LLM infra.
  + AWS WAF + Security Groups + NACLs.
* **Observability & Auditing**
  + CloudWatch for metrics & alarms; CloudTrail + S3 access logs for audit trail.
  + Central SIEM (Splunk/Elastic) for long retention of logs.
* **Frontend**
  + React web app + native mobile (React Native) served from S3/CloudFront with short-lived auth tokens.

## **4. Security Architecture (authentication, authorization, data protection)**

**Authentication**

* Company SSO (Okta/Azure AD) with enforced MFA.
* Short-lived OAuth2 tokens (JWT) issued to frontend clients.

**Authorization**

* Document-level authorization model:  
  + Maintain an RLS-like mapping in metadata store: (doc-id → ACL groups) and per-section sensitivity tags.
  + At retrieval time, all candidate vectors are filtered by ACL membership before assembling context. This prevents leaking content from confidential docs.
* Token introspection + role claims used by the Orchestrator to allow/deny document access.

**Data Protection**

* Encryption at rest (S3 server-side with KMS) and in transit (TLS).
* Use VPC endpoints and restrict management consoles to admin IPs.
* No plaintext secrets in code; secrets in AWS Secrets Manager or Parameter Store, with strict IAM.

**Audit & Compliance**

* Log each query with user-id, returned document-ids, timestamp, and decision (why a doc was excluded/included).
* Retain logs per company policy (e.g., 1 year hot, 5 years cold archive).
* Provide admin dashboards for audit review.

## **5. Indexing & Real-time Updates**

**Ingestion pipeline**

* Watch S3 buckets or connectors (SharePoint/Drive) for new/changed docs — trigger ingestion via SQS/Lambda or EventBridge.
* Workers extract text, chunk, compute embeddings, store vectors with metadata and permission tags.
* Support incremental reindex and soft-delete handling.

**Realtime updates**

* Use event-driven ingestion; mark vectors with version. Retrievers check versioning to avoid stale context.
* Use small, frequent batches for new documents to keep near-real-time indexing.

## **6. Retrieval, Reranking and Generation**

* **Retriever:** k-NN search returning top-K candidates by vector similarity.
* **Reranker:** Lightweight cross-encoder reranker (if budget allows) to re-score top results using a small transformer to improve precision.
* **Context assembler:** Trims context to token budget, preserves highest-confidence and permissioned snippets, returns a set of citations (doc-id, section, score).
* **LLM:** Takes assembled context + user question; LLM returns answer and supporting references. If hallucination risk is high, return source-excerpts verbatim and mark uncertain answers.

## **7. Scaling Strategy & Sizing (to satisfy 500 concurrent users & $8k/mo)**

**MVP (cost-conscious)**

* Embeddings & LLM use hosted API with pay-as-you-go; vector DB: single-node managed (Pinecone starter / OpenSearch t3.medium cluster).
* Ingestion: a small ECS Fargate cluster (2-3 tasks).
* Frontend: S3 + CloudFront.
* Expected recurring costs estimate (ballpark):  
  + Vector DB: $200–1,000 / month (varies by vendor & scale)
  + LLM API usage: $1,000–4,000 / month (depends on traffic & model)
  + Storage (S3), RDS metadata, compute, monitoring, CDN: $1,000–2,500 / month
  + Total MVP: aim to stay ≤ $8k by throttling heavy model calls & caching responses.

**Scaling approach**

* Use autoscaling groups/ECS task autoscaling for ingestion & API workers.
* Scale vector DB shards/replicas as corpus grows; plan for horizontal scaling.
* Cache popular query answers and embeddings to reduce LLM calls.
* For higher scale, self-host LLM inference on GPU clusters to reduce per-call costs.

## **8. Cost Optimization Techniques**

* Cache retrievals & generated responses (TTL-based) to avoid repeated LLM calls.
* Use smaller, faster reranker models to reduce top-LLM calls.
* Use sampling or offload cold data to cheaper storage tiers.
* Commit to reserved instances / savings plans for predictable baseline compute.
* Monitor and set budgets/alerts; implement rate-limiting for heavy users.

## **9. Implementation Phases & Timeline (high-level)**

**Phase 0 — Discovery & MVP (4–8 weeks)**

* Goals: ingest a sample corpus, vector indexing, simple chat UI, use hosted embedding + LLM APIs, show citations, basic SSO.
* Deliverables: working demo, audit log pipeline, permission model prototype.

**Phase 1 — Production Hardening (4–6 weeks)**

* Goals: permission enforcement across full corpus, retention & backup, compliance checks, monitoring & alerts, enforce region constraints.
* Deliverables: production deployment in chosen US region, SLA testing.

**Phase 2 — Scale & Cost Optimization (6–10 weeks)**

* Goals: caching, reranker, autoscaling, cost reductions; add advanced retrieval & logging dashboards.
* Deliverables: ability to handle 500 concurrent users with load testing & budget alignment.

**Phase 3 — Optional: Self-hosted LLM (8–12 weeks)**

* Goals: move to in-VPC LLM inference (if needed for residency/cost), fine-tune models on internal data, implement feedback loop.
* Deliverables: GPU inference cluster + model operations.

## **10. Business Risks & Mitigations**

* **Hallucinations:** Always return source citations; provide confidence flags and UI to view source excerpts.
* **Data leaks / permission bypass:** Strict metadata-based filtering before text assembly; external audits.
* **Costs runaway:** Set rate limits, budgets, and caching. Use monitoring to detect abnormal usage.
* **Regulatory / residency changes:** Keep all resources in identified US regions; use provider contracts.

## **11. Metrics to Track**

* Query latency (P95)
* LLM calls per minute & cost per query
* Retrieval precision@k and user satisfaction
* Audit log completeness
* Uptime & error rates
* Index freshness lag (time from doc upload → queryable)

## **12. Deliverables & Artefacts to Hand Over**

* Architecture diagram & cloud resource list (IaC recommended: Terraform)
* Ingestion & indexing runbooks
* SSO / IAM config templates
* Monitoring & alerting playbooks
* SLA / compliance evidence pack

## **13. Appendix: Quick vendor comparison (short)**

* **Pinecone:** Easy to use, high performance. Check US-only hosting & enterprise SLA.
* **Milvus:** Open-source, self-hosted on EKS, more operational effort.
* **OpenSearch k-NN:** Can be run in-VPC, simpler ops if already using AWS.

Signed,  
 Manish jangir