Cloud Test: Enterprise RAG System Architecture

Cloud Architecture Design Submission

November 2025

1. Assumptions

Document Volume & Characteristics

- **Total Documents**: ~100,000 documents accumulated over 10+ years
- **Document Types**: 60% PDFs, 25% Word documents, 10% PowerPoint, 5% emails/other
- **Average Document Size**: 2-5 MB per document
- **Total Storage**: ~300-500 GB of raw documents
- **Growth Rate**: ~10,000 new documents per year (~800/month)

Update Frequency & Access Patterns

- **Batch Uploads**: Weekly bulk uploads of 50-200 documents
- **Real-time Indexing**: Required for newly added documents within 15 minutes
- **Query Volume**: ~2,000-5,000 queries per day (avg 3 queries/user/week)
- **Peak Usage**: Business hours (9 AM - 5 PM EST), 500 concurrent users during peaks
- **Query Distribution**: 70% simple lookups, 30% complex multi-document queries

User Access Patterns

- **Total Users**: 500 employees with authenticated access
- **Active Users**: ~300 users querying monthly, ~80 users daily
- **Document Permissions**: 70% accessible to all, 20% department-restricted, 10% confidential
- **Mobile Access**: 30% of queries from mobile devices

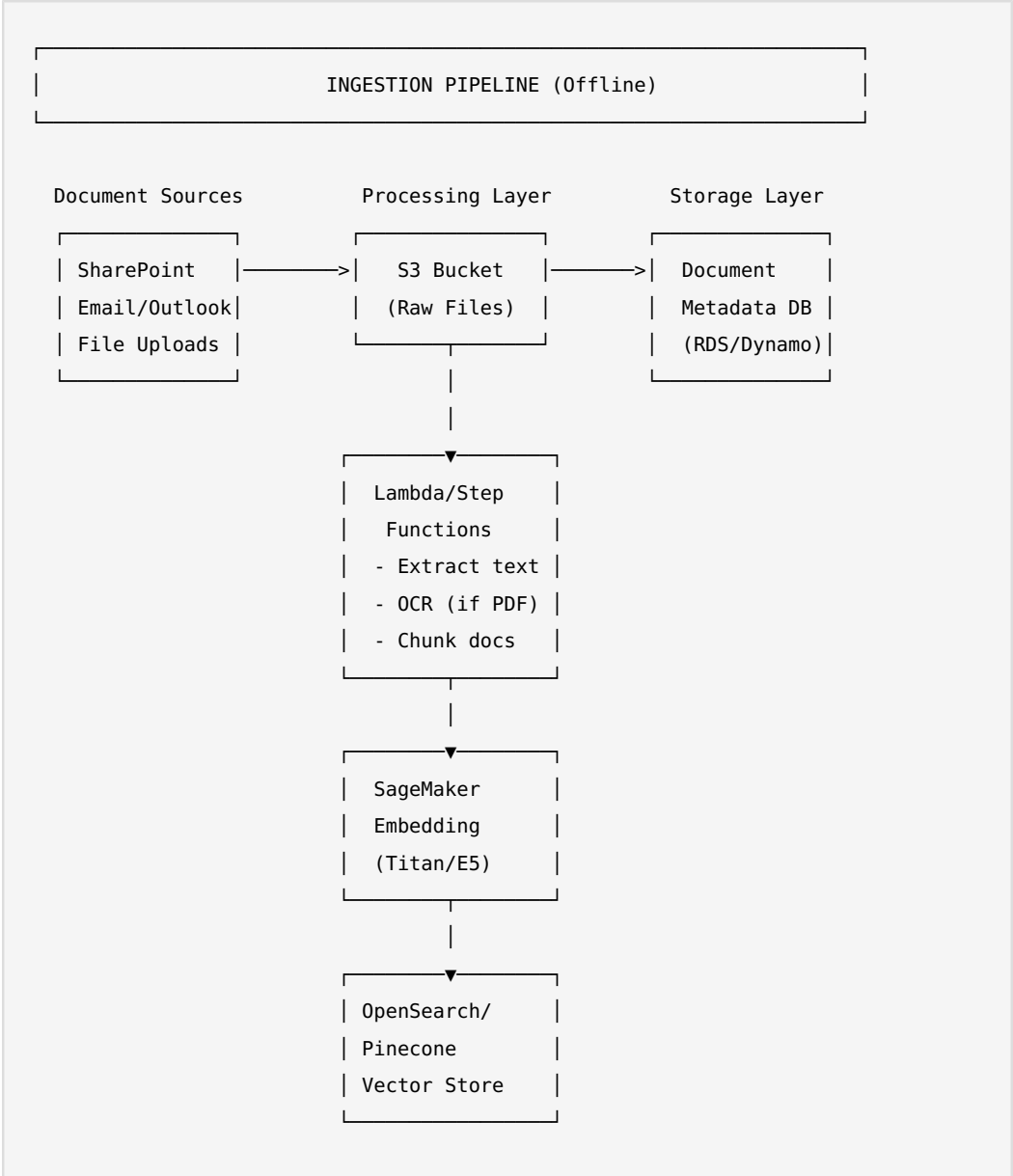
Technology Stack Familiarity

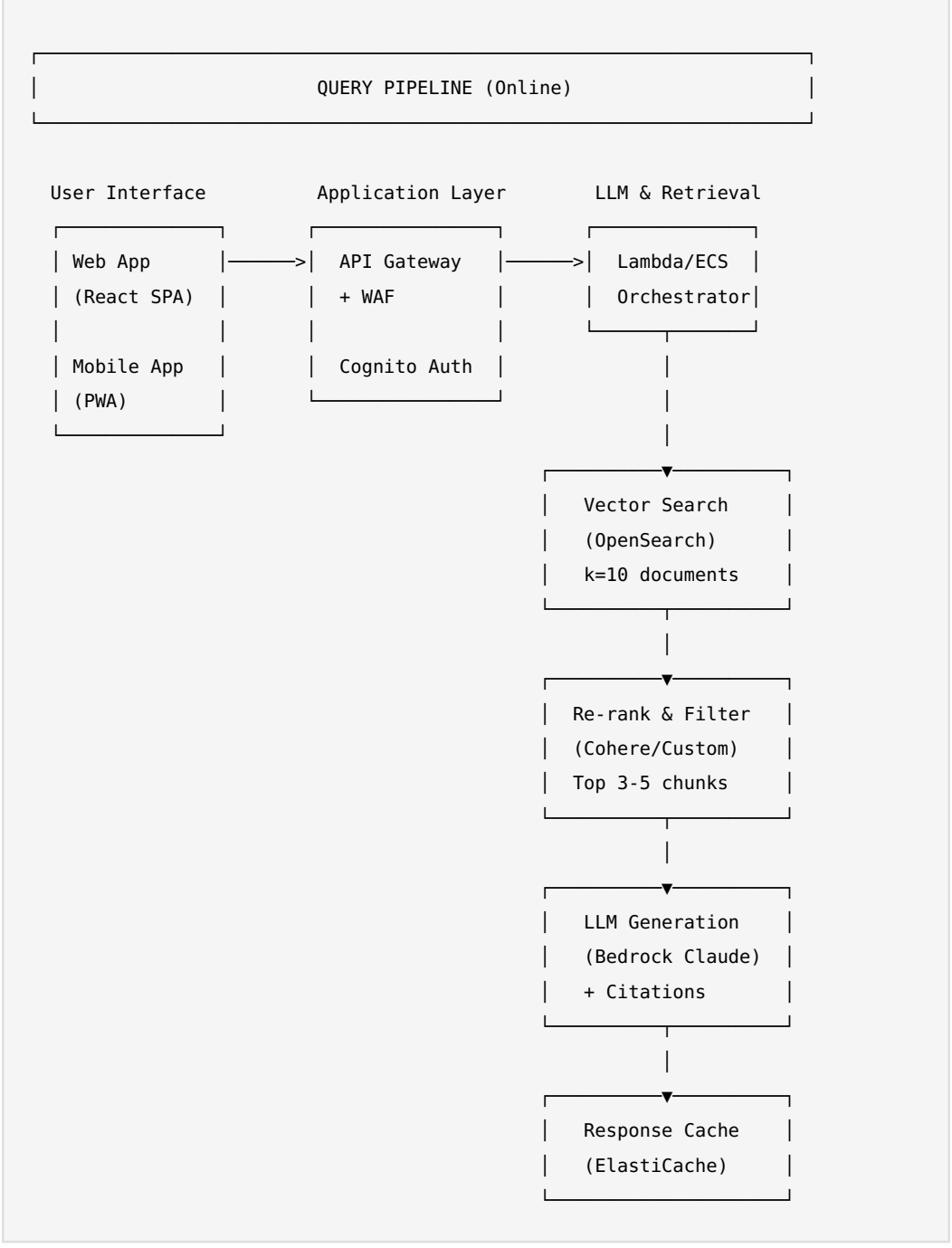
- **Cloud Provider**: AWS (assumed based on enterprise adoption, US data residency)
- **Development Team**: 3-5 engineers familiar with Python, React, cloud services
- **Maintenance**: Preference for managed services over custom infrastructure
- **Compliance**: SOC 2, basic data residency requirements (US-only)

2. High-Level Architecture

System Overview

The RAG system follows a dual-pipeline architecture: (1) an offline ingestion pipeline for document processing and indexing, and (2) an online query pipeline for retrieval and generation.





Data Flow Summary

Ingestion Path: Document → S3 → Text Extraction → Chunking → Embedding → Vector Store + Metadata DB

Query Path: User Query → Auth → Embedding → Vector Search → Re-rank → LLM Context → Generated Response + Citations → User

3. Ingestion and Indexing Pipeline

Document Collection

Source Integration

- **Primary Sources**:
  - SharePoint/OneDrive API integration for existing documents
  - Email integration via Microsoft Graph API for project emails
  - Web upload interface for ad-hoc documents
- **Implementation**:
  - AWS Lambda functions triggered by S3 events or scheduled EventBridge rules
  - Initial bulk migration: AWS DataSync or custom batch scripts
  - Ongoing sync: Daily scheduled Lambda pulling from SharePoint API

File Storage

- **Raw Document Storage**: S3 Standard tier
  - Bucket structure: `s3://company-docs-raw/{year}/{department}/{document-id}`
  - Lifecycle policy: Move to S3 Glacier after 1 year (documents rarely re-processed)
  - Versioning enabled for document updates

Document Preprocessing

Text Extraction Pipeline

```
Lambda Function (Python 3.11, 10GB RAM, 15min timeout)
├── PDF Processing: PyPDF2 + Tesseract OCR for scanned documents
├── Word/PPT: python-docx, python-pptx libraries
├── Email: email.parser library for .eml files
└── Output: Extracted plain text + metadata (author, date, department, access level)
```

Chunking Strategy

- **Chunk Size**: 512 tokens (~400 words) with 50-token overlap
- **Method**: Semantic chunking using LangChain's RecursiveCharacterTextSplitter
  - Split on paragraphs first, then sentences
  - Preserve section headers with chunks for context
- **Metadata Preservation**: Each chunk tagged with:
  - `document_id`, `chunk_index`, `source_file`, `page_number`
  - `department`, `created_date`, `author`, `access_level`
  - `document_type`, `project_id` (if applicable)

OCR Handling

- **Scanned PDFs**: Detected via PDF metadata or text extraction failure
- **OCR Engine**: Amazon Textract for production quality
  - Cost optimization: Use async API for batch processing
  - Fallback: Tesseract OCR for cost-sensitive scenarios

Embedding Generation

Model Selection: Amazon Titan Embeddings G1 or E5-large-v2

- **Reasoning**:
  - Titan: Fully managed, AWS-native, good cost/performance
  - E5-large: Higher quality, can self-host on SageMaker for cost control
- **Dimensions**: 1024 (Titan) or 768 (E5)
- **Batch Processing**: 100 chunks per API call via SageMaker batch transform

Implementation

```
        # Pseudo-code for embedding pipeline
        def process_document(s3_key):
            text = extract_text(s3_key)
            chunks = chunk_document(text, size=512, overlap=50)

            embeddings = sagemaker_batch_embed(chunks)

            for chunk, embedding in zip(chunks, embeddings):
                metadata = extract_metadata(s3_key, chunk)
                store_vector(embedding, chunk.text, metadata)
                store_metadata_db(chunk.id, metadata)
```

Vector Store: Amazon OpenSearch Serverless

Configuration

- **Index Settings**:
  - Engine: k-NN with HNSW algorithm

    - Similarity: Cosine similarity
    - Dimensions: 1024
    - M (neighbors): 16, ef_construction: 512
- **Sharding**: Auto-scaled based on data volume
- **Replicas**: 2 for high availability

Alternative: Pinecone (if cost-effective at scale)

- Serverless tier for auto-scaling
- Built-in filtering for metadata queries
- Simpler management, potential cost savings at 100k+ vectors

4. RAG Retrieval + Response Logic

Query Processing Flow

1. Query Embedding

- Same embedding model as documents (Titan/E5) for consistency
- Real-time inference via SageMaker endpoint (1-2 instances, auto-scaling)
- Latency target: <100ms for embedding generation

2. Vector Search Strategy

Initial Retrieval: k=20 documents - Cosine similarity threshold: >0.7 (configurable) - Metadata pre-filtering for access control: `python filter_query = { "bool": { "must": [{"term": {"access_level": user.allowed_levels}}], "should": [{"term": {"department": user.department}}] } }`

Hybrid Search (Optional Enhancement): - Combine vector search (semantic) + BM25 (keyword) with 0.7:0.3 weighting - Improves recall for specific terms (contract IDs, project names)

3. Re-ranking Strategy

Model: Cohere Rerank API or custom cross-encoder - Input: Query + top 20 chunks - Output: Rescored and sorted top 5-10 chunks - Latency: ~200-500ms (acceptable for quality improvement)

Criteria: - Semantic relevance to query - Recency (boost documents from recent years) - Document authority (boost official templates, final reports)

4. Context Assembly

Selected Chunks: Top 3-5 after re-ranking - **Context Window**: ~3000 tokens for LLM (Claude 3 Sonnet: 200k context) - **Template**: ``` Context Documents: [Document 1: {title} - {date}] {chunk_text} Source: {file_name}, Page {page_number}

[Document 2: ...]

User Question: {query}

Instructions: Answer based solely on the provided context. Always cite sources by document title and page number. ```

5. LLM Generation

Model: **Amazon Bedrock - Claude 3.5 Sonnet** - **Reasoning**: - Best-in-class reasoning for complex queries - Strong citation accuracy - 200k context window handles large document sets - AWS-native, simplified deployment

Inference Configuration: - Temperature: 0.1 (low for factual accuracy) - Max tokens: 1024 - Stop sequences: None (allow full responses)

Prompt Engineering:

```
You are a knowledge assistant for {Company}. Answer the question
using ONLY the provided documents. Always cite your sources.

If the answer is not in the documents, say "I don't have information
```

```
about that in the available documents."

Format citations as: [Document Title, Page X]
```

6. Citation Attachment

Post-processing: - Parse LLM response for citation markers - Validate citations against retrieved chunks - Enrich with clickable links to original documents in S3 - Format: `[Healthcare Project Summary 2022, p.5](https://link-to-doc)`

7. Response Caching

Cache Layer: Amazon ElastiCache (Redis) - Key: Hash of (query + user_access_level) - TTL: 1 hour for frequently asked questions - Cache hit rate target: 20-30% (saves LLM costs)

5. User Interface + Application Layer

Front-End Experience

Web Application (Primary)

Technology Stack: - **Framework**: React 18 with TypeScript - **UI Library**: Material-UI or Tailwind CSS - **State Management**: React Query for API calls, Zustand for local state - **Hosting**: AWS Amplify or S3 + CloudFront

Features: - Clean chat interface (ChatGPT-like UX) - Real-time typing indicators - Inline citation cards with document previews - Search history and saved queries - Dark/light mode

Key Interactions:

```
┌─────────────────────────────────────────┐
│  Search: "Healthcare projects 2022"     │
├─────────────────────────────────────────┤
│                                         │
│  💬 Assistant:                          │
│  In 2022, we completed 8 healthcare     │
│  projects focusing on EHR integrations  │
│  and telemedicine platforms.            │
│                                         │
│  Key challenges included:               │
│  • HIPAA compliance requirements        │
│  • Legacy system integrations           │
│                                         │
│  📄 Sources:                            │
│  [1] Healthcare Summary 2022.pdf, p.3   │
│  [2] Project Retrospective Q4.docx      │
│                                         │
```

```
│  [View Documents] [Follow-up Question]  │
└─────────────────────────────────────────┘
```

Mobile Application (Progressive Web App)

Technology: Same React app with responsive design - PWA manifest for “Add to Home Screen” - Offline support for recent queries (service workers) - Touch-optimized interface

Document Preview

- Embedded PDF viewer (PDF.js) highlighting relevant sections
- Deep links to specific pages
- Download options (with audit logging)

Backend Services

API Gateway

AWS API Gateway (REST API): - Rate limiting: 100 requests/minute per user - Request validation and sanitization - CORS configuration for web/mobile clients - Custom domain: `https://api.knowledge.company.com`

Endpoints: - `POST /query` - Main RAG query endpoint - `GET /documents/{id}` - Document retrieval - `GET /history` - User query history - `POST /feedback` - Response quality feedback

Application Orchestrator (Lambda/ECS)

Option 1: AWS Lambda (Recommended for cost) - Language: Python 3.11 - Memory: 3GB - Timeout: 30 seconds - Concurrency: 500 reserved (matches user count)

Option 2: ECS Fargate (For complex workflows) - Container: Python FastAPI application - Auto-scaling: 2-10 tasks based on CPU/memory - Use case: If adding complex multi-turn conversations

Orchestration Logic:

```
async def handle_query(query, user_id):
    # 1. Authenticate and get user permissions
    user = get_user_context(user_id)

    # 2. Check cache
    cached = check_cache(query, user.access_level)
    if cached:
        return cached

    # 3. Embed query
    query_vector = await embed_query(query)

    # 4. Vector search with access filters
    candidates = search_vectors(query_vector, user.filters)

    # 5. Re-rank
    top_chunks = rerank(query, candidates)

    # 6. Generate response
    response = await llm_generate(query, top_chunks)

    # 7. Log and cache
    log_query(user_id, query, response)
    cache_response(query, response)

    return response
```

Asynchronous Processing

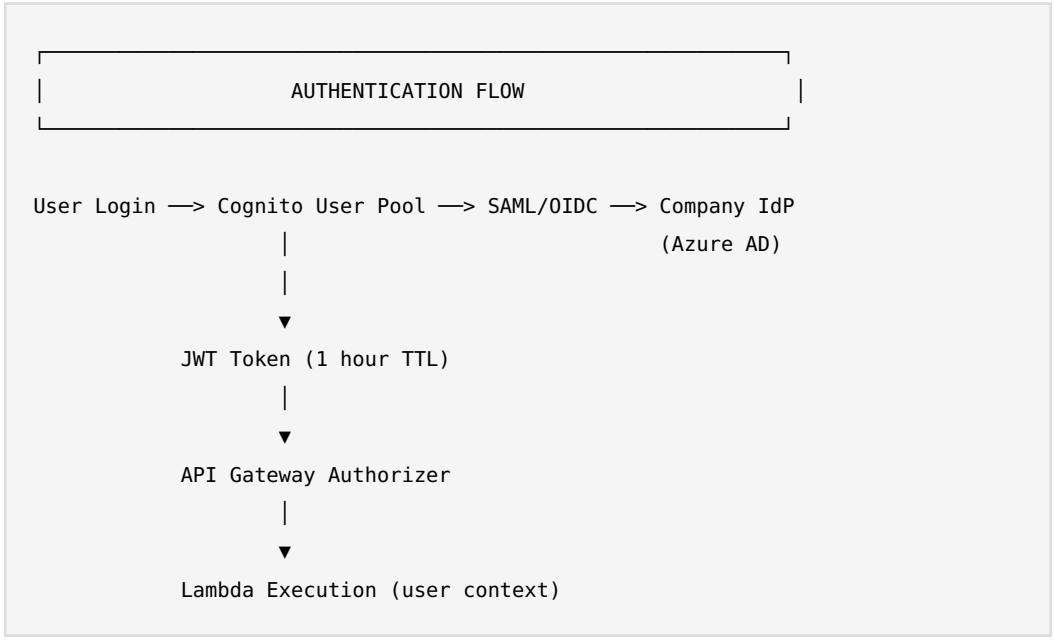
- **SQS Queue**: For document ingestion jobs
- **Step Functions**: Orchestrate multi-stage document processing

- **EventBridge**: Schedule daily sync jobs

6. Security Architecture

Authentication & Authorization

Employee Authentication: AWS Cognito



Configuration: - **Integration**: SAML 2.0 with Azure AD/Okta (company IdP) - **MFA**: Required for all users - **Token Expiration**: 1 hour access tokens, 30-day refresh tokens - **Groups**: Synced from company directory (Engineering, Sales, HR, etc.)

Document-Level Authorization

Access Control Model:

```
        Document Metadata:
        {
            "document_id": "doc_12345",
            "access_level": "department",  # public | department | confidential
```

```
    "allowed_departments": ["Engineering", "Product"],
    "allowed_users": ["user_789"],  # For confidential docs
    "created_by": "user_456"
}

User Context:
{
    "user_id": "user_123",
    "department": "Engineering",
    "groups": ["Engineering", "Managers"],
    "clearance_level": "standard"  # standard | sensitive
}
```

Filter Application: - Vector search includes metadata filters (OpenSearch query filters) - Double-check: Post-retrieval validation before passing to LLM - Audit: Log when access is denied

Encryption Strategy

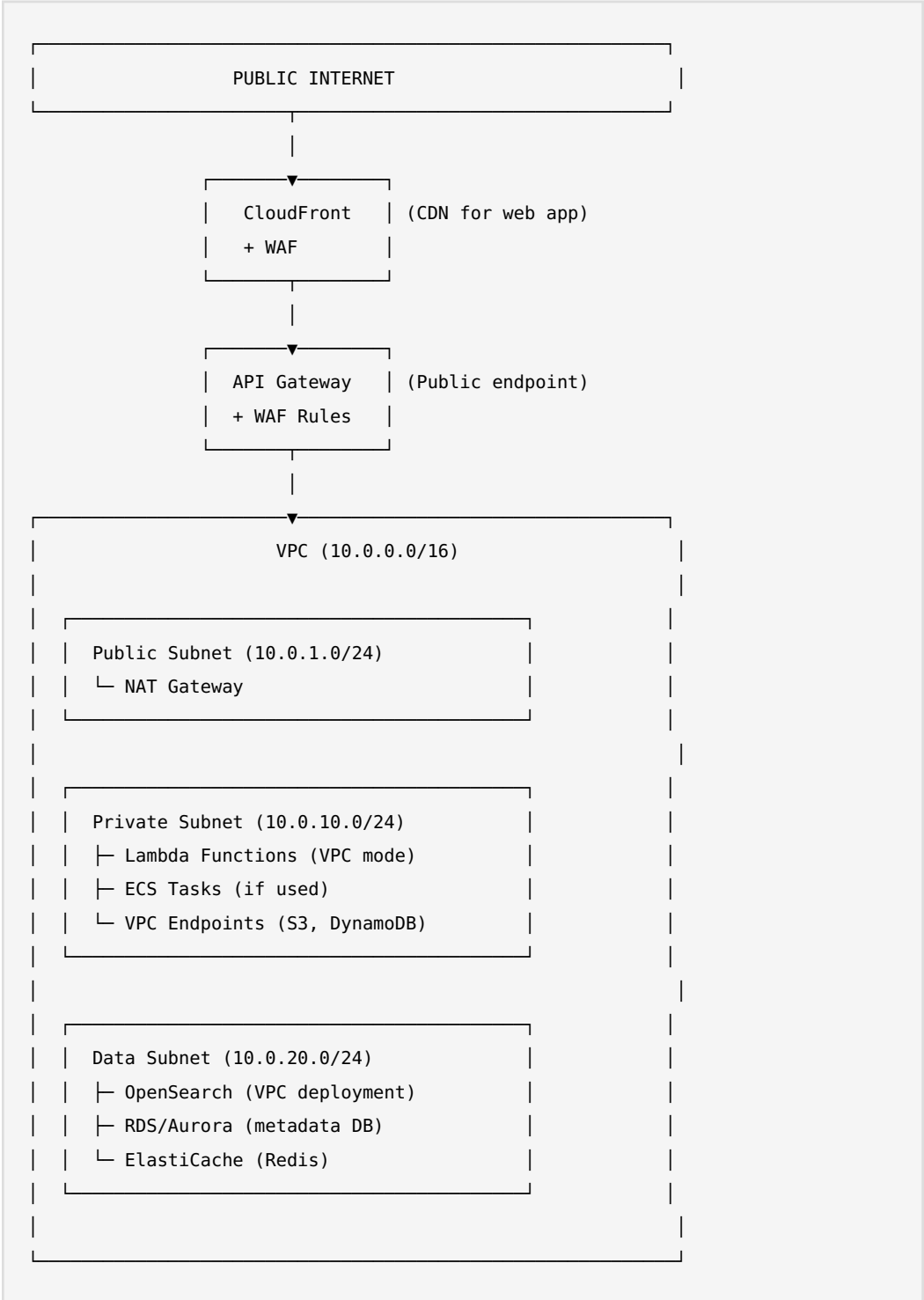
Data at Rest: - **S3**: SSE-S3 (AES-256) for raw documents - **OpenSearch**: Encryption at rest enabled (AWS-managed keys) - **RDS/DynamoDB**: Default encryption with KMS - **KMS Key**: Customer-managed CMK for sensitive data rotation

Data in Transit: - **TLS 1.3**: All API communications - **VPC PrivateLink**: Internal AWS service communication - **Certificate Management**: AWS Certificate Manager (ACM)

Secrets Management: - **AWS Secrets Manager**: API keys, database credentials - **IAM Roles**: Preferred over long-lived credentials - **Rotation**: Automatic 90-day rotation for DB passwords

Network Security

Network Architecture



Segmentation: - Application layer cannot directly access data layer (routed through private subnets) - No public IPs for backend services - VPC endpoints for AWS services (no internet egress)

WAF Rules (AWS WAF)

- **Rate Limiting**: 2000 requests per 5 minutes per IP
- **Geo-Blocking**: US-only access (data residency requirement)
- **OWASP Top 10**: Managed rule groups for SQLi, XSS, etc.
- **IP Reputation**: AWS managed IP reputation lists

Audit Logging & Monitoring

Audit Trail

CloudWatch Logs + **S3 Archive**: - **Query Logs**: User ID, timestamp, query text, results returned, access denials - **Document Access**: Who accessed which documents, when - **Authentication Events**: Login attempts, MFA challenges, session expirations - **Data Changes**: Document uploads, deletions, permission changes

Retention: - CloudWatch: 90 days (hot access for investigations) - S3 Archive: 7 years (compliance requirement)

Security Monitoring

AWS GuardDuty: Threat detection for unusual API calls or access patterns

AWS Config: Configuration compliance tracking - Rule: Ensure all S3 buckets have encryption enabled - Rule: Ensure all Lambda functions are in VPC (if required)

CloudWatch Alarms: - Spike in 403 errors (potential unauthorized access attempts) - Unusual query patterns (potential data exfiltration) - Failed authentication rate >10/ minute

Compliance

Automated Evidence Collection: - **AWS Audit Manager**: Pre-configured SOC 2 frameworks - **Data Residency**: Tag all resources with region, monitor with Config - **Access Reviews**: Quarterly Cognito user group audits

7. Scaling Strategy

Document Volume Growth

Current State (Year 1)

- **Documents**: 100,000
- **Vectors**: ~500,000 (5 chunks/document avg)
- **Storage**: 500 GB raw + 50 GB vectors

Projected Growth (Year 5)

- **Documents**: 150,000 (50% growth)
- **Vectors**: 750,000
- **Storage**: 750 GB raw + 75 GB vectors

Scaling Approach

Vector Store Scaling: - **OpenSearch Serverless**: Auto-scales data nodes based on storage/query load - **Pinecone**: Automatically handles scaling to millions of vectors - **Sharding Strategy**: If self-managed, shard by department (natural partitioning)

Embedding Pipeline: - **SageMaker Endpoint**: Auto-scaling from 1-5 instances based on CloudWatch metrics - **Batch Processing**: Use SageMaker Batch Transform for bulk re-indexing (cost-effective)

Storage: - **S3**: Unlimited scaling, no action needed - **Metadata DB**: - Start: RDS PostgreSQL (db.t3.medium, 100 GB) - Scale: Upgrade to db.r5.large or migrate to DynamoDB (serverless)

User Concurrency Scaling

Current Requirement

- **Peak Users**: 500 concurrent
- **Queries**: ~5000/day (~350/hour during peaks)

Scaling Components

API Gateway: - Default limit: 10,000 requests/second (more than sufficient) - No scaling action needed

Lambda Functions: - **Reserved Concurrency**: 500 (matches peak users) - **Provisioned Concurrency**: 50 (for cold start elimination) - **Cost**: Provisioned concurrency adds ~$500/month, reduces latency from 2s → 200ms

SageMaker Embedding Endpoint: - Auto-scaling policy: `yaml ScalingPolicy: TargetValue: 70% (InvocationsPerInstance) MinCapacity: 2 instances MaxCapacity: 10 instances ScaleUpCooldown: 60 seconds ScaleDownCooldown: 300 seconds`

Bedrock LLM: - Fully managed, no scaling configuration - Pay-per-token pricing absorbs spikes automatically

ElastiCache (Redis): - Start: cache.t3.medium (1 node) - Scale: Add read replicas (2-3 nodes) for read-heavy workloads - Cluster mode: Enable if cache size exceeds 100 GB

Performance Under Load

Latency Targets

Component	Target	Scaling Strategy
Query Embedding	<100ms	SageMaker auto-scaling
Vector Search	<200ms	OpenSearch compute scaling
Re-ranking	<300ms	Async processing if >500ms
LLM Generation	<3s	Use Bedrock on-demand (no scaling needed)
Total E2E	**<5s**	Cache frequent queries for <500ms

Load Testing Plan

Tools: Locust or Artillery for API load testing

Test Scenarios: 1. **Baseline**: 100 concurrent users, 10 queries/min each 2. **Peak**: 500 concurrent users, 20 queries/min each 3. **Spike**: 1000 users for 5 minutes (2x peak)

Success Criteria: - 95th percentile latency <5 seconds - Error rate <0.1% - Cache hit rate >25%

Database Scaling

Metadata DB (RDS PostgreSQL): - **Read Replicas**: 1-2 replicas for query history, document metadata reads - **Connection Pooling**: RDS Proxy (500 max connections) - **Upgrade Path**: db.t3.medium → db.r5.large → Aurora Serverless v2

Alternative: DynamoDB: - Better for unpredictable scaling - Cost-effective with on-demand pricing - Trade-off: Less flexible for complex queries

High Availability

Multi-AZ Deployment

- **RDS**: Multi-AZ for automatic failover
- **OpenSearch**: 2 replicas across 3 AZs
- **ElastiCache**: Multi-AZ with automatic failover
- **Lambda**: Inherently multi-AZ

Disaster Recovery

- **RTO (Recovery Time Objective)**: 1 hour
- **RPO (Recovery Point Objective)**: 15 minutes
- **Backup Strategy**:
  - S3: Versioning + cross-region replication (to US-WEST for DR)
  - RDS: Automated backups (daily snapshots, 7-day retention)
  - OpenSearch: Automated snapshots to S3 (daily)

Uptime Target: 99.5%

- **Allowed Downtime**: ~3.6 hours/month
- **Monitoring**: CloudWatch synthetic canaries (1-minute checks)
- **Alerting**: PagerDuty integration for on-call team

8. Cost Strategy

Monthly Cost Breakdown (Estimated)

Compute & Processing

Service	Configuration	Monthly Cost
Lambda (Query handling)	350,000 invocations/ month, 3GB RAM, 5s avg	$250
Lambda (Document processing)	10,000 docs/month, 10GB RAM, 5min avg	$200
SageMaker Endpoint (Embeddings)	2x ml.m5.large instances, 50% utilization	$500
Bedrock (Claude 3.5 Sonnet)	5M input tokens, 1M output tokens/month	$900
Subtotal		**$1,850**

Storage

Service	Configuration	Monthly Cost
S3 (Raw documents)	500 GB Standard, 10 GB uploads/ month	$15
S3 (Processed data)	100 GB	$3
OpenSearch Serverless	50 GB indexed data, ~500K vectors	$800
RDS PostgreSQL	db.t3.medium, 100 GB storage	$120
Subtotal		**$938**

Networking & Caching

Service	Configuration	Monthly Cost
CloudFront	100 GB data transfer, 1M requests	$20
API Gateway	350,000 requests/month	$1.25
ElastiCache (Redis)	cache.t3.medium, 1 node	$60
VPC Endpoints	3 endpoints (S3, DynamoDB, Secrets Manager)	$22
Data Transfer	50 GB out to internet	$5
Subtotal		**$108**

Security & Monitoring

Service	Configuration	Monthly Cost
Cognito	500 MAU (monthly active users)	$14
WAF	2 rules, 350K requests	$10
CloudWatch Logs	20 GB ingestion, 3-month retention	$30
GuardDuty	500 GB CloudTrail, VPC Flow Logs	$40
Secrets Manager	5 secrets	$2
Subtotal		**$96**

Additional Services

Service	Configuration	Monthly Cost
Step Functions	10,000 state transitions/month	$2.50
SQS	1M requests	$0.40
Textract (OCR)	1,000 pages/month	$15
EventBridge	10,000 events	$0.10
Subtotal		**$18**

TOTAL MONTHLY COST: ~$3,010

Buffer for unexpected usage: ~$500 **Target Budget**: $8,000/month **Remaining Budget**: ~$4,500/month (56% under budget)

Cost Optimization Strategies

Reserved Capacity (20-40% savings)

1. **RDS Reserved Instance**: 1-year commitment saves ~$40/month
2. **SageMaker Savings Plan**: 1-year commitment saves ~$150/month
3. **ElastiCache Reserved Nodes**: 1-year commitment saves ~$20/month
4. **Total Savings**: ~$210/month = $2,520/year

Auto-Scaling & Right-Sizing

1. **SageMaker Endpoint**: Scale to 1 instance during off-hours (6 PM - 8 AM)
   - Savings: ~$150/month
2. **Lambda Provisioned Concurrency**: Only during business hours (9 AM - 6 PM)
   - Savings: ~$300/month (if using provisioned concurrency)
3. **RDS Instance Scheduler**: Stop dev/test instances on weekends
   - Not applicable for production, but consider for staging environments

Managed vs. Self-Hosted Tradeoffs

Option 1: Replace OpenSearch Serverless with Pinecone Serverless - Pinecone: ~$70/month for 500K vectors (starter tier) - Savings: ~$730/month - Trade-off: Vendor lock-in, data egress costs

Option 2: Replace SageMaker with Self-Hosted Embeddings (ECS) - ECS Fargate: 2x small tasks (~$100/month) - Savings: ~$400/month - Trade-off: Maintenance overhead, slower cold starts

Option 3: Use Bedrock Claude 3 Haiku (instead of Sonnet) - Haiku: 80% cheaper for simpler queries - Hybrid approach: Route simple queries to Haiku, complex to Sonnet - Savings: ~$500/month (if 60% queries can use Haiku)

Caching Strategy (Cost Reduction)

- **ElastiCache Hit Rate**: 30% → saves 30% of LLM API calls
  - LLM cost reduction: ~$270/month
  - Cache cost: $60/month
  - **Net Savings**: $210/month

Data Lifecycle Management

1. **S3 Lifecycle Policies**:
   - Move raw documents to Glacier after 1 year: saves ~$10/month (minimal)
   - Delete temporary processing files after 7 days
2. **CloudWatch Logs**:
   - Reduce retention from 90 days to 30 days for non-audit logs: saves ~$10/month

3. **OpenSearch Index Lifecycle**:
    - Archive older document versions (if versioning enabled)

Alternative: Serverless-First Architecture

Replace RDS with DynamoDB: - DynamoDB On-Demand: ~$50-100/month (vs $120 RDS) - Savings: ~$20-70/month - Trade-off: NoSQL schema design, less flexible querying

Cost Monitoring & Alerts

AWS Cost Explorer: - Daily cost tracking by service - Budget alerts at 80%, 100%, 120% of $3,500/month

Anomaly Detection: - AWS Cost Anomaly Detection for unexpected spikes - Alert on >20% day-over-day increases

Cost Allocation Tags: - Tag all resources: `Project:KnowledgeRAG`, `Environment:Production`, `Department:Engineering` - Track costs by department for chargeback

9. Risks, Tradeoffs, and Alternatives

Design Optimizations

Primary Focus

1. **Security & Compliance**: Document-level access control, audit logging, US data residency
2. **Cost-Effectiveness**: Under budget by 56%, managed services for lower maintenance
3. **Developer Velocity**: Familiar stack (AWS, Python, React), managed services reduce ops burden

Intentionally Deprioritized

1. **Multi-Cloud Strategy**
   - **Why**: Adds complexity, increases costs, not required by business constraints
   - **Risk**: Vendor lock-in to AWS
   - **Mitigation**: Use portable abstractions (LangChain), avoid AWS-specific APIs where possible
2. **Advanced RAG Techniques**
   - **Not Implemented**: Multi-query generation, query decomposition, agentic workflows
   - **Why**: Simple RAG meets 80% of use cases, added complexity not justified yet
   - **Future**: Iterate based on user feedback and failure analysis

3. **Real-Time Collaboration Features**
    - **Not Implemented**: Multi-user chat sessions, shared workspaces
    - **Why**: Not in requirements, adds architectural complexity
    - **Future**: Phase 2 feature if demand exists
4. **Advanced Analytics**
    - **Not Implemented**: Query trend analysis, document popularity tracking
    - **Why**: Focus on core functionality first
    - **Future**: Add once baseline is stable

Key Risks & Mitigations

Technical Risks

Risk	Impact	Probability	Mitigation
Poor retrieval quality	High	Medium	- Extensive testing with real queries - A/B test chunking strategies - Implement user feedback loop - Add re-ranking layer
LLM hallucination	High	Medium	- Low temperature (0.1) - Strict prompt engineering - Citation validation - User feedback: “Was this helpful?”
High latency (>10s)	Medium	Low	- Aggressive caching - Async query processing

Risk	Impact	Probability	Mitigation
			- Optimize vector search parameters - Provision concurrency for Lambda
OCR accuracy for scanned docs	Medium	Medium	- Use Textract (high accuracy) - Flag low-confidence extractions - Manual review queue for critical docs
Access control bypass	High	Low	- Defense in depth: metadata filters + post-retrieval checks - Penetration testing - Regular access audits

Business Risks

Risk	Impact	Probability	Mitigation
User adoption resistance	High	Medium	- Extensive user training - Gradual rollout (pilot → full) - Champions program - Show value early (quick wins)
Document quality issues	Medium	High	- Data quality assessment during migration

Risk	Impact	Probability	Mitigation
			- Metadata enrichment workflows - User reporting for bad results
Cost overruns	Medium	Low	- Conservative estimates with 50% buffer - Cost alerts at 80% - Auto-scaling limits to prevent runaway costs
Compliance violations	High	Low	- Legal review of architecture - Regular compliance audits - Automated evidence collection

Operational Risks

Risk	Impact	Probability	Mitigation
On-call burden	Low	Medium	- Comprehensive monitoring - Runbook documentation - Auto-remediation for common issues - Managed services reduce ops load
Vendor API limits	Medium	Low	- Monitor Bedrock quotas - Request limit increases proactively - Implement graceful degradation

Risk	Impact	Probability	Mitigation
Data loss	High	Low	- Multi-AZ deployments - Cross-region S3 replication - Regular backup testing - Immutable audit logs

Alternative Architectures Considered

1. Self-Hosted LLM (e.g., Llama 3, Mistral)

Pros: - No per-token costs, predictable pricing - Full control over model behavior - Data never leaves VPC

Cons: - High upfront GPU costs (~$2,000+/month for p3.2xlarge) - Ops burden: model updates, fine-tuning, monitoring - Lower quality vs. Claude 3.5 Sonnet - Harder to scale for bursty traffic

Verdict: Not cost-effective at this scale, revisit if query volume exceeds 50K/day

2. Hybrid Search (Vector + Traditional Search)

Pros: - Better recall for specific terms (project IDs, contract numbers) - Complements semantic search

Cons: - Adds complexity (2 indexes to maintain) - Minimal improvement for narrative queries

Verdict: Consider as Phase 2 optimization after measuring retrieval quality

3. Fine-Tuned Embedding Model

Pros: - Domain-specific embeddings for consulting terminology - Potentially better retrieval accuracy

Cons: - Requires labeled data (1000+ query-document pairs) - Training and hosting costs - Maintenance overhead

Verdict: Not justified for v1, revisit if retrieval quality is poor

4. Multi-Modal RAG (Images, Charts, Tables)

Pros: - Extract information from visuals in PowerPoints, PDFs - More comprehensive knowledge coverage

Cons: - Requires vision-capable LLM (Claude 3.5 supports this) - Textract table extraction adds cost - Increased complexity

Verdict: Phase 2 feature, implement if users request it

5. GraphRAG (Knowledge Graphs)

Pros: - Better for relationship queries (“Who worked on X project?”) - Explainable retrieval paths

Cons: - Requires entity extraction and graph construction - Complex to implement and maintain - Minimal benefit for document-centric queries

Verdict: Not suitable for this use case, better for knowledge graphs with structured data

Future Improvements

Phase 2 (Months 6-12)

1. **Multi-Turn Conversations**: Maintain chat history, context-aware follow-ups
2. **Query Suggestions**: Auto-complete based on popular queries
3. **Advanced Filters**: Filter by date range, document type, department
4. **User Feedback Loop**: “Helpful/Not Helpful” buttons → improve retrieval
5. **Analytics Dashboard**: Popular queries, low-quality responses, usage trends

Phase 3 (Year 2+)

1. **Proactive Insights**: Weekly digests of relevant new documents
2. **Document Summarization**: Auto-generate summaries for long documents
3. **Hybrid Search**: Combine vector + keyword search for specific queries
4. **Multi-Lingual Support**: If expanding to international offices
5. **Fine-Tuned Models**: Custom embeddings and LLMs for domain expertise

Success Metrics

Technical KPIs

- **Retrieval Accuracy**: >85% of queries return relevant documents (measured via user feedback)

- **Latency**: 95th percentile <5 seconds
- **Uptime**: >99.5% (within SLA)
- **Cache Hit Rate**: >25%

Business KPIs

- **User Adoption**: >60% of employees use system monthly within 6 months
- **Query Volume**: 5,000+ queries/month by month 3
- **Time Savings**: 30% reduction in time spent searching for documents (user survey)
- **Cost Efficiency**: Remain under $4,000/month in Year 1

Quality KPIs

- **Citation Accuracy**: >95% of citations point to correct sources
- **User Satisfaction**: >4.0/5.0 average rating
- **False Positive Rate**: <10% of responses marked “Not Helpful”

Conclusion

This RAG system architecture balances security, scalability, and cost-effectiveness for a mid-size consulting firm. By leveraging AWS managed services (Bedrock, OpenSearch Serverless, Cognito), we minimize operational overhead while maintaining enterprise-grade security controls.

The design prioritizes: 1. **Document-level access control** to protect confidential information 2. **Cost efficiency** (56% under budget with room to scale) 3. **Developer productivity** (familiar stack, managed services) 4. **Iterative improvement** (phased rollout, feedback loops)

With comprehensive monitoring, audit trails, and a clear scaling path, the system is production-ready and positioned for long-term success.