Comprehensive Report: Data Lakes for Side-by-Side PDF Viewer with RAG Integration

Executive Summary

This report provides a detailed guide on leveraging data lakes for managing PDFs, metadata, and embeddings in a side-by-side PDF viewer integrated with Retrieval-Augmented Generation (RAG). It covers architecture, data flow, storage strategies, retrieval optimization, tools, implementation steps, and best practices.

Introduction

Modern AI-driven applications require efficient storage and retrieval of large volumes of documents and associated metadata. A data lake offers a scalable, cost-effective solution for storing raw PDFs, metadata, and embeddings, enabling advanced features like semantic search and synchronized scrolling in PDF viewers.

Why Use Data Lakes

Data lakes provide schema-on-read flexibility, support for structured and unstructured data, and seamless integration with big data and AI pipelines. They allow centralized storage of raw files and metadata, making them ideal for RAG-powered applications.

Architecture Overview

The architecture consists of three layers: Storage (Data Lake for PDFs and metadata), Processing (ETL pipelines for extraction and embedding generation), and Retrieval (Vector DB for semantic search, SQL DB for metadata indexing). The PDF viewer interacts with these layers via APIs to fetch relevant chunks and highlight them in the UI.

Data Flow Steps

1. 1. PDF Upload → Store in Data Lake.
2. 2. Extract text and metadata (page, bbox, chunk_text).
3. 3. Generate embeddings for each chunk using AI models.
4. 4. Store metadata in Data Lake and index in SQL DB.
5. 5. Store embeddings in a vector database for semantic search.
6. 6. On user query, retrieve relevant chunks and metadata.
7. 7. PDF viewer scrolls and highlights using bbox coordinates.

Storage Strategy

- - Raw PDFs: Store in cloud object storage (AWS S3, Azure Data Lake, GCS).
- - Metadata: Save as JSON or Parquet in the data lake; optionally index in SQL DB for fast lookups.
- - Embeddings: Store in a vector database (Pinecone, Weaviate, Milvus) for semantic search.

Retrieval Optimization Techniques

- - Partition metadata files by doc_id and date for faster queries.
- - Use hybrid architecture: data lake for storage, SQL DB for quick lookups, vector DB for embeddings.
- - Precompute bounding boxes and cache frequently accessed chunks in Redis.
- - Implement lazy loading for PDFs in the viewer.

Tools & Tech Stack

- - Storage: AWS S3, Azure Data Lake, Google Cloud Storage.
- - Processing: Apache Spark, Databricks, AWS Glue.
- - Vector Search: Pinecone, Weaviate, Milvus, FAISS.
- - Orchestration: Airflow, Prefect.
- - Embedding Models: OpenAI text-embedding-3-large, HuggingFace models.

Implementation Plan

8. Step 1: Set up cloud storage bucket for PDFs and metadata.
9. Step 2: Build ETL pipeline to extract text and metadata using PyMuPDF or pdf.js.

10. Step 3: Generate embeddings using OpenAI or HuggingFace models.
11. Step 4: Store embeddings in a vector database and metadata in SQL DB.
12. Step 5: Implement API endpoints for retrieval and UI integration.
13. Step 6: Integrate scrollToChunk(chunkMeta) in the PDF viewer for synchronized highlighting.

Best Practices

- - Always include normalized bounding boxes and original PDF coordinates.
- - Standardize coordinate origins for consistency.
- - Use semantic metadata for smarter scroll synchronization.
- - Cache frequently accessed chunks for performance.

Conclusion

Data lakes provide a robust foundation for managing large-scale document workflows in AI-powered applications. By combining data lakes with vector databases and optimized retrieval strategies, developers can deliver seamless, interactive PDF viewing experiences integrated with RAG.