Software Specification: Unstructured Data Indexing and AI-Query Application

1 Introduction and Scope

Modern enterprises accumulate vast amounts of unstructured data – images, audio, video, text documents, log files and other formats – across services such as **Box** and **SharePoint**. Unstructured data lacks a predefined schema and can include multimedia files, text, web logs and many other forms. Traditional storage systems struggle to scale and to search such repositories; companies therefore need tools that can impose *artificial structure* on the data and make it searchable with AI models.

This specification defines an end-to-end system that ingests unstructured data from cloud repositories on a per-customer basis, extracts and stores descriptive metadata in MongoDB, performs AI-powered summarization and classification on selected files, builds an index suitable for retrieval-augmented generation (RAG) and provides a conversational interface through Microsoft Teams. The goal is to enable users to ask questions against their data lake and obtain meaningful answers grounded in the relevant files.

2 High-Level Objectives

1. **Traverse and monitor cloud file systems.** The system must traverse deep directory trees in cloud repositories (e.g., Box, SharePoint) and discover new, modified or deleted files. Processed files must be tracked to avoid duplication and enable incremental updates.
2. **Pre-filter and metadata extraction.** A Python pre-filter analyses file metadata and content type using the cloud provider's API. Certain file types (audio, video, images, CAD drawings, etc.) are indexed but not passed to AI for expensive processing.
3. **Dynamic metadata database.** A MongoDB database stores a document per file. The schema is dynamic: new metadata fields can be added at run time as the AI pipeline discovers new features. Each field is annotated to describe its purpose (e.g., sensitive_data_pii: yes/no) and supports full-text and vector search.
4. **AI processing pipeline.** Text-based files flagged by the pre-filter are summarized and classified using LLMs. The AI prompt is dynamically tuned with user input (representative directories, target data types such as legal or financial documents, cost constraints) and through continuous improvement.
5. **Master index and RAG.** A vector-based index is built from the summarized content. Retrieval-Augmented Generation (RAG) combines an information retrieval system with LLMs to provide grounded answers. The index supports queries against the entire data lake.

6. **Teams-based query interface.** A Teams bot receives questions, optimizes the queries, retrieves relevant files via the RAG index and returns AI-generated answers. All user queries are recorded for auditing and iterative improvement.
7. **Backup and recovery.** The MongoDB database is backed up to Azure Blob Storage with immutable (WORM) policies, protecting backups from modification or deletion. Backups occur every 4 business hours when activity is detected.
8. **Deployment and scalability.** The system is containerized. Each major component (pre-filter, AI pipeline, database, indexer, Teams bot) runs as its own service, allowing horizontal scaling across customers. Azure AI and other cloud services supply compute for AI workloads.

3 System Architecture

3.1 Component Overview

1. **Ingestion & Monitor Service** (Python). Communicates with Box/SharePoint via their APIs. Performs initial crawl of directory trees; stores file and directory metadata; schedules incremental scans; handles webhook notifications (if available) for near real-time change detection. Maintains a *State Store* recording the last processed version (hash, modified timestamp or ETag) of each file.
2. **Pre-filter Service** (Python). For each discovered file:
    - Retrieves metadata (file path, unique ID, file name, author, last modified time, size, mime type) from the cloud provider.
    - Classifies the file type. Files such as .mp3, .mp4, images (e.g., .jpg, .png), CAD or design files are considered **non-textual** and are **metadata-only**; they receive basic indexing but are *not* forwarded to the AI pipeline. Text-based files (.pdf, .docx, .txt, .pptx, .msg, etc.) are flagged for AI processing.
    - Populates an initial MongoDB document for the file with metadata fields and classification flags.
3. **Metadata & Feature Store (MongoDB).** Each file corresponds to a MongoDB document. Required fields include:

Field	Purpose	Example
file_id	Cloud provider's unique identifier for the file.	box_file_id

Field	Purpose	Example
file_path	Full path in the repository.	/CustomerA/contracts/2024/abc.docx
file_name	Human-readable name.	abc.docx
modified_time	Last modified date/time.	2025-07-06T12:34:56Z
author	File author/owner where available.	jane.doe@example.com
mime_type	MIME type or file extension.	application/pdf
size_bytes	File size for cost estimation.	145032
ai_processed	Flag: has AI processing been completed?	false
non_textual	Flag: file is metadata-only.	true/false
tags	List of classification tags (e.g., “contract”, “financial”).	["contract"]
sensitive_flags	Map of sensitivity types (PII, PCI, PHI, etc.) with yes/no values.	{ "PII": "yes" }
embedding	Vector representation of summarized content.	[float32,...]
summary	Short summary extracted via AI.	“Contract between ...”
extracted_entities	Extracted names, dates, amounts, etc.	{"Company": "Acme"}

4. The document model is flexible. When a new feature is discovered (e.g., detection of a certain clause or risk indicator), the system adds a new field to documents that require it. A separate **Field Metadata Collection** stores definitions of each field

(e.g., field_name, description, data_type, sensitivity), enabling dynamic annotation and easy search on new columns.

5. **AI Processing Pipeline.** Runs asynchronously on flagged files. For each file:

a. **Prompt Engineering:** The pipeline builds a prompt that describes the context and instructs the AI model to summarize the document, extract key entities, detect sensitive data (PII, PCI, etc.) and classify the document by type (e.g., legal contract, financial statement, proposal). It incorporates user-provided directives such as desired categories, cost budget, and representative directories. The prompt uses existing metadata to reduce redundancy and may ask the user to refine or confirm categories.

b. **Model Invocation:** Calls Azure OpenAI (GPT-4 or GPT-4-Turbo) or another LLM. For large documents, the file is chunked into sections (e.g., 2k tokens each). Summaries and extracted entities are aggregated. Sensitive data detectors may be integrated via additional models or rule-based algorithms.

c. **Dynamic Improvement:** The prompt and summarization strategy are logged with performance metrics (e.g., summarization accuracy, cost). A continuous improvement loop analyses user feedback on query answers, adjusts prompt parameters (chunk size, summarization depth) and updates classification rules. Because RAG relies on high-quality retrieval, the pipeline attempts to produce concise yet comprehensive summaries.

d. **Database Update:** The MongoDB document for the file is updated with the summary, extracted entities, tags and embedding vectors. ai_processed is set to true and processing_time is recorded.

6. **Master Index & Vector Store.** After AI processing, the summarized text and extracted features are used to create vector embeddings (e.g., using Azure OpenAI Embeddings or Hugging Face models). A **Vector Index** is built for each customer, enabling similarity search. The system may use Azure AI Search or a managed vector database (e.g., Pinecone, Qdrant) for storing embeddings. Azure AI Search is a proven information retrieval system for RAG and supports indexing strategies that refresh at scale and return relevant results within token limits. The index is updated periodically or via streaming to incorporate new or modified embeddings.

7. **RAG Query Orchestrator.** Handles user questions and coordinates retrieval and answer generation:

1. **Receive question via Teams.** The Teams bot captures the user's query and persists it in a **Query Log** (MongoDB). The bot uses Azure Bot Framework for message routing.

2. **Query Optimization:** A small LLM or heuristics rephrase the question for clarity and injects context (e.g., specifying the customer and date range). The orchestrator may ask follow-up questions if the intent is unclear.

3. **Retrieve relevant files.** The orchestrator uses the vector index to perform similarity search; filters results by metadata (date, author, tags) and queries MongoDB for additional filters (e.g., sensitive_flags.PII == 'yes'). This yields a ranked set of documents and their summaries.

4. **Generate answer.** A final prompt is created that includes the refined question and the retrieved summaries as "grounding data". Using an LLM, the system generates an answer and returns it through Teams. The prompt emphasizes citing the source files and warns against fabricating information.

5. **Post-processing:** The system logs the answer, metrics (latency, cost, feedback), and updates the continuous improvement loop. If no relevant files exist, the bot politely reports that the knowledge base contains no information on the topic.

8. **Backup & Recovery Service.** A scheduled task (cron job) runs every four business hours (e.g., 9am, 1pm, 5pm, 9pm local time) if database writes have occurred since the last backup. It dumps MongoDB data to compressed archives and uploads them to an Azure Blob Storage container with **Immutable Storage** configured. Immutable storage stores data in a **Write Once, Read Many (WORM)** state; files cannot be modified or deleted for a user-specified interval, ensuring backups are tamper-proof. The system uses time-based retention policies so that backups can be deleted only after the retention period expires. Audit logs record retention policy changes.

9. **Administration & Monitoring.** A web-based admin UI allows operators to view ingestion status, AI processing queue, index statistics and query logs. Logging and metrics are centralized (e.g., via Azure Monitor or Elastic Stack). Alerts notify administrators of failures, high error rates or unusual activity. Access is role-based; administrators can grant or revoke customer access and adjust scanning frequency.

3.2 Data Flow Diagram

(1) **Initial Ingestion:** The ingestion service scans the repository and stores metadata into MongoDB. (2) **Pre-filter** decides whether to forward the file to AI. (3) **AI processing** summarizes and annotates files. (4) **Vector embeddings** are generated and stored in the vector index. (5) **RAG query orchestrator** handles user queries by retrieving relevant embeddings and generating answers. (6) **Backups** copy MongoDB data to immutable Azure storage.

4 Detailed Component Specifications

4.1 Ingestion & Monitor Service

1. **Directory Traversal:** Uses the cloud provider's API (e.g., Box Folders API, SharePoint REST or Graph API) to list directories and files. Traversal is done breadth-first to avoid deep recursion stacks and can run in parallel to improve throughput. The service respects API rate limits by batching requests and using exponential backoff.
2. **State Tracking:** Maintains a collection (processing_state) in MongoDB that stores the last processed modified_time and a content hash or version ID for each file. On each scan, the service compares the current file metadata against this state; only new or changed files are passed to the pre-filter. Deleted files result in deletion of their documents (or marking them as deleted for historical traceability).
3. **Scheduling & Notifications:** Provides two modes:
    - **Scheduled Scans.** A configurable scheduler triggers full or incremental scans at set intervals (e.g., every hour). Suitable for providers without event notifications.
    - **Event-Driven Scans.** Subscribes to webhooks from Box or SharePoint (where supported) for near-real-time change events. The service throttles bursts of events to avoid overloading the AI pipeline.
4. **Scalability:** The service runs as a container replicated per customer or as a multi-tenant service with concurrency controls. Each instance uses asynchronous I/O to handle numerous API requests. Authentication tokens for each customer are stored securely (e.g., Azure Key Vault).

4.2 Pre-filter & Metadata Extraction

1. **Metadata Retrieval:** For each discovered file, the service requests file metadata from the cloud API. Additional attributes (file owner, creation date, version history) are captured where available.
2. **File Type Classification:**
    - Determine MIME type via the API or local inspection (file extension, magic numbers).
    - Maintain an allowlist of text-processable types (PDF, Word, Excel, PowerPoint, plain text, HTML, JSON, markdown, emails) and a blocklist of

non-textual types (audio, video, images, CAD, archives). Blocklisted files are flagged as non_textual and not forwarded for AI analysis.

3. **Sensitive Data Pre-Detection:** Use heuristic or open-source tools (e.g., regex patterns for SSNs, credit card numbers) to flag files likely containing PII/PCI. These flags help prioritize AI processing and cost estimation.
4. **Insertion into MongoDB:** Create or update a document in the collection files. Basic fields are inserted (see Section 3.1). Indexes are created on file_id, file_path, modified_time, tags, and any frequently queried fields to optimise lookups.

4.3 AI Processing Pipeline

1. **Prompt Design:** A templated prompt instructs the LLM to summarise the document, identify the document type, extract structured data (dates, amounts, parties), detect sensitive information and produce tags. The template includes:
    - The file's metadata (name, path, last modified date) to give context.
    - The user-provided goals (e.g., "focus on legal clauses" or "extract financial figures").
    - A cost budget: the pipeline estimates the number of tokens based on file size and chooses summary depth accordingly.
    - Guidelines to avoid hallucination and to respond in JSON so that the output can be parsed programmatically.
2. **User-Driven Customization:** At system setup or periodically, the user is prompted via Teams to select representative directories and to specify categories of documents (legal, financial, proposals, contracts, HR, etc.) and sensitive data types of interest. This information informs the prompt templates and classification models.
3. **Self-Improvement Loop:** The AI pipeline tracks the performance of prompts by comparing extracted summaries with ground truth when available or by soliciting user feedback. It adjusts summarization length, classification thresholds and entity extraction patterns to improve accuracy and reduce cost over time.
4. **Model Selection:** The default model is Azure OpenAI's GPT-4-Turbo or GPT-4 depending on cost/quality trade-offs. For large file types, summarization may be done in stages using smaller models (e.g., GPT-3.5-Turbo) before final refinement with GPT-4.

5. **Post-Processing:** Parsed JSON is validated. If extraction fails or the content is too large, the file is retried with adjusted parameters. Successfully processed data is stored in MongoDB and the vector index is updated.

4.4 Master Index & Vector Store

1. **Embedding Generation:** Use an embedding model (e.g., Azure OpenAI's text-embedding-ada-002) to generate a vector for each summarized chunk. Store the embedding along with document ID and segment ID. For non-textual files, embeddings may be generated from available captions or left blank.
2. **Index Storage:**
    - **Azure AI Search**: The index service supports both keyword and vector search and offers indexing strategies that refresh at scalelearn.microsoft.com. Each index field maps to a metadata field (e.g., summary, tags, sensitive_flags). Vector search is enabled on the embedding field, and text search on summary.
    - **Alternative vector databases:** For multi-cloud flexibility, Pinecone, Qdrant or Milvus can be used. The system provides an abstraction to switch between vector backends.
3. **Index Refresh:** When files are added or modified, the ingestion service triggers asynchronous updates to the vector index. A batch refresh job runs daily to reconcile the index with MongoDB and remove stale entries.
4. **Retrieval Strategy:** Use hybrid search: first perform a metadata filter on date, tags and sensitive flags via MongoDB; then perform vector search on the filtered set; finally rank the results using a scoring function combining vector similarity, recency and user-specified weights.

4.5 RAG Query Orchestrator & Teams Bot

1. **Bot Registration:** Register a bot in Azure Bot Service configured for Microsoft Teams. Use OAuth to authenticate users. Implement conversation logic using Bot Framework SDK.
2. **Natural Language Understanding:** A lightweight model (could be GPT-3.5-Turbo) cleans and reformulates user questions, extracts key parameters (customer, time range, subject). For ambiguous queries, ask clarifying questions through Teams.

3. **File Retrieval:** Query the vector index using the processed query as embedding. Limit results to a reasonable number (e.g., top 10 summaries) to stay within model token limits. Use additional filters from metadata (e.g., tags or sensitive_flags).
4. **Answer Generation:** Construct a prompt that includes:
    - The refined question.
    - Summaries and key facts from each selected document.
    - Instructions to base the answer strictly on the provided summaries, cite file names and avoid hallucination.
    - Optionally, the conversation history for context.

Send this prompt to the LLM (via Azure OpenAI). Return the answer to Teams and log the conversation ID, question, answer and source document IDs.

5. **Post-Answer Feedback:** Allow the user to rate the answer or indicate if important information was missing. Feedback loops into the AI pipeline's self-improvement process.

4.6 Backup & Disaster Recovery

1. **Backup Frequency:** A schedule runs every four business hours (e.g., 9am, 1pm, 5pm, 9pm America/New_York) when writes have occurred since the last backup.
2. **Backup Process:** Use mongodump to create compressed archives of all collections, optionally with separate dumps per customer. Upload to Azure Blob Storage using client libraries with SAS tokens.
3. **Immutable Storage:** Configure the destination container with **time-based retention policies** so that backups are stored in a **WORM** state where they cannot be modified or deleted until the retention period expireslearn.microsoft.com. This ensures secure backups for regulatory compliancelearn.microsoft.com. Retention periods (e.g., 90 days) can be configured per policy. Audit logs record policy changes and backup events.
4. **Restore Procedure:** Provide scripts to download and restore backups from blob storage. In case of partial corruption, restore specific collections. The backup service also supports point-in-time restore using oplog if configured.

4.7 Security & Compliance

1. **Customer Isolation:** Each customer's data is stored in a separate MongoDB database or via tenant-specific collections with strict access controls. API credentials for Box/SharePoint are stored in Azure Key Vault and scoped per tenant.
2. **Authentication & Authorization:** Users authenticate via Microsoft Entra ID (Azure AD). Roles determine access to resources (administrative console, query interface). API calls to cloud providers use OAuth tokens with least privilege.
3. **Data Encryption:** Enable encryption at rest for MongoDB and vector store. Use TLS for all network communication. Secrets (e.g., API keys, database passwords) are injected at runtime via environment variables or Key Vault references.
4. **Sensitive Data Handling:** The AI pipeline flags PII, PCI, PHI and other sensitive data. These flags can be used to mask sensitive content in responses or to restrict access. The system logs access to sensitive documents for auditing.
5. **Compliance:** Using immutable blob storage ensures compliance with WORM regulations, preventing deletion or modification of backupslearn.microsoft.com. The system should support data residency requirements and configurable retention policies per customer.

4.8 Containerization & Deployment

1. **Service Separation:** Each major component (ingestion, pre-filter, AI processing, indexer, query orchestrator, backup, admin UI) runs in its own Docker container. Containers communicate via REST or message queues (e.g., RabbitMQ or Azure Service Bus).
2. **Orchestration:** Use Kubernetes (AKS) or Docker Compose to manage services. Horizontal Pod Autoscaling can increase instances of the AI processing component during peaks.
3. **Environment Configuration:** Use environment variables for configuration (e.g., database URIs, API endpoints, concurrency limits). Secrets are retrieved from Key Vault at startup.
4. **Monitoring:** Deploy sidecar containers for collecting metrics (Prometheus exporters) and logs (Fluent Bit). Alert rules notify on high error rates, failing backups or long processing times.
5. **CI/CD:** Implement pipelines (e.g., GitHub Actions, Azure Pipelines) to build, test and deploy images. Use blue/green deployments for minimal downtime.

4.9 Scalability & Performance

1. **Parallel Processing:** The ingestion service processes directories concurrently. The AI pipeline uses asynchronous tasks and job queues to handle many files; concurrency is limited based on available AI tokens and cost budgets.
2. **Sharding:** For very large datasets, shard MongoDB collections by customer or by file path hash. Use index partitioning in the vector store.
3. **Caching:** Cache AI summarization results and vector search results to avoid repeated processing. Use Redis or Azure Cache for quick retrieval of recent queries.
4. **Incremental Updates:** By tracking file version IDs, the system avoids re-processing unchanged files, reducing compute cost.
5. **Cost Control:** The system monitors the number of tokens processed and the cost per query. Users can set budgets; the AI pipeline will adjust summarization granularity and the number of retrieved documents accordingly.

5 Additional Recommended Features

1. **OCR & Image Text Extraction:** Integrate OCR (e.g., Azure Computer Vision) to extract text from scanned PDFs or images. This makes image-based documents searchable.
2. **Table Extraction:** Use AI models to extract tables from PDFs/Excel and store them as structured data. This enables queries like “list payment amounts from invoices”.
3. **Duplication Detection:** Calculate content hashes to detect duplicate or near-duplicate files across directories. Only store one representation and maintain cross-reference pointers.
4. **User-Defined Tags & Annotations:** Allow users to add manual tags, notes or ratings to files. These annotations are stored in MongoDB and considered during retrieval.
5. **Dashboards & Analytics:** Provide dashboards showing file counts by type, growth over time, sensitive data distribution and AI processing status. Insights help prioritize scanning and training budgets.
6. **API Access:** Expose REST and GraphQL APIs for integration with other enterprise systems, allowing programmatic querying and ingestion.
7. **Support for Additional Data Sources:** Extend connectors to other repositories (Google Drive, OneDrive, AWS S3, email archives, Git repositories). The architecture allows plug-in connectors.

8. **Workflow Automation:** Trigger workflows based on file classification. For example, when a contract is detected, notify legal teams; when sensitive data is found, create a compliance ticket.
9. **Advanced Privacy Controls:** Support differential privacy for aggregated analytics and allow automatic redaction of sensitive information in responses.
10. **AI Model Governance:** Track versions of models and prompts used for each file. Provide mechanisms to re-process content when models are updated.

6 Conclusion

This specification outlines a comprehensive application for structuring and querying unstructured data from cloud repositories. By combining an ingestion pipeline, dynamic metadata storage, a sophisticated AI processing layer, retrieval-augmented generation and a conversational Teams interface, the system transforms messy files into actionable knowledge. The use of MongoDB's flexible schema supports unstructured data storagemongodb.com, and Azure AI Search provides robust retrieval capabilitieslearn.microsoft.com. Immutable backups in Azure Blob Storage protect data against tamperinglearn.microsoft.com, while containerized deployment ensures scalability and operational resilience. The modular design allows new connectors, models and features to be added as requirements evolve, giving users powerful tools to derive insight from their unstructured data lakes.