# Unstructured Data Indexing & AI-Query Application — Software Specification (v2)

\*\*Document status:\*\* Draft for review • \*\*Date:\*\* Aug 7, 2025 • \*\*Owners:\*\* Engineering & Product

## 1. Executive Summary

This system indexes large, unstructured enterprise repositories (Box, SharePoint, OneDrive), applies metadata extraction and optional sensitive-data classification, and exposes natural‑language Q&A via a Microsoft Teams bot and an Admin Web UI. It supports incremental scanning, durable progress tracking, cost visibility with forecast, and answer rendering including tables and charts.

## 2. Goals & Scope

\*\*Goals\*\*

* Connect to Box, SharePoint, and OneDrive at enterprise scale via Model Context Protocol (MCP) servers.
* Traverse deep directory trees efficiently; resume and re‑scan only when items change.
* Create a master index (MongoDB) and a vector index (Azure AI Search) for retrieval‑augmented generation (RAG).
* Provide end‑user Q&A in Teams; provide admin controls (policies, cost tracking, directory selection, processing controls) in a Web UI.
* Ensure strong security (least privilege, tenant isolation, secrets in Key Vault) and governance (audit, PII/PHI/ITAR controls).

\*\*Out of scope\*\*

* End‑user document editing; DLP beyond the defined policy enforcement; non‑Microsoft chat surfaces other than Teams.

## 3. Definitions & Acronyms

* \*\*MCP\*\* — Model Context Protocol: standardized way for tools/servers to expose capabilities to models/clients.
* \*\*MCP Python Interpreter\*\* — MCP tool for controlled execution of Python utilities.
* \*\*Box MCP Server\*\* — MCP server providing Box file system tools.
* \*\*Microsoft files‑mcp‑server\*\* — MCP server exposing SharePoint/OneDrive file system tools.
* \*\*RAG\*\* — Retrieval Augmented Generation.
* \*\*PII/PHI/PCI/ITAR\*\* — Sensitive data categories.

## 4. System Architecture

### 4.1 Components

* \*\*Admin Web UI\*\* (Next.js or equivalent): Admin policy editor, directory picker, run controls, spend dashboard, answer audit.
* \*\*Teams Bot Service\*\* (Python/Node): Conversational entry point; Adaptive Cards; OAuth via Entra ID.
* \*\*RAG Orchestrator\*\* (Python): Query parsing, retrieval, prompt assembly, post‑processing (tables/graphs) and citations.
* \*\*Ingestion & Monitor Service\*\* (Python): Schedules full/incremental scans; receives change events; orchestrates per‑file pipelines.
* \*\*MCP Connectors\*\* (sidecars): MCP Python Interpreter, Box MCP Server, Microsoft files‑mcp‑server.
* \*\*Pre‑filter & Metadata Extractor\*\* (Python): File type detection, text extraction, metadata normalization.
* \*\*Summarizer & Embedder\*\* (Azure OpenAI): Per‑file summary JSON and embeddings.
* \*\*Vector Index\*\* (Azure AI Search, vector + keyword): Stores embeddings and metadata for retrieval.
* \*\*Master Database\*\* (MongoDB): File catalog, processing state, policies, costs, Q&A logs.
* \*\*Object Store\*\* (Azure Blob): Cache for large artifacts, chart images, CSV exports, backups (immutable/WORM buckets).
* \*\*Secrets & Config\*\* (Azure Key Vault); \*\*Observability\*\* (Azure Monitor/App Insights); \*\*CI/CD\*\* (GitHub Actions/Azure DevOps).

### 4.2 Data Flow (high level)

1. Admin connects sources (Box/SharePoint/OneDrive). 2) Ingestion traverse discovers files and emits work items. 3) Pre‑filter extracts text/metadata. 4) Summarizer creates structured summary JSON; Embedder produces embeddings. 5) Index update in MongoDB + AI Search. 6) Teams Q&A retrieves, composes answer, emits tables/graphs as needed. 7) Costs metered and forecasted; policies enforced in retrieval/answer.

### 4.3 Tenancy & Isolation

Single‑control plane, per‑tenant credentials and policy records. Data partitioning at DB/Index level via tenant\_id and at storage via per‑tenant containers.

## 5. Detailed Component Specifications

### 5.1 Ingestion & Monitor Service

\*\*Responsibilities\*\*: discovery, change detection, enqueueing, orchestration of file pipeline, poisoning/redo handling, and metrics.

#### 5.1.1 Directory Traversal Controller (Python)

Directory traversal is \*\*triggered and controlled by an associated set of Python programs\*\* that operate in three modes:

* \*\*CLI\*\* — `python -m ingestion.traverse --tenant TENANT --source {box|sharepoint|onedrive} --mode {full|incremental} [--paths "/Finance,/HR"]`
* \*\*Scheduler\*\* — CronJob (AKS) for periodic incremental scans (default hourly, tenant‑configurable).
* \*\*Event‑driven\*\* — Webhook/MCP event ingester maps provider change notifications to targeted re‑scan queue items.

The controller uses MCP tools for listing and content fetch (Box MCP Server, files‑mcp‑server). It persists traversal cursors (folder IDs, delta tokens), last‑seen ETags/versions, and content hashes in MongoDB.

#### 5.1.2 Change Detection & Idempotency

* A file is re‑processed only if any of: \*\*modified\_time\*\*, \*\*provider version/ETag\*\*, or \*\*content hash\*\* has changed.
* A per‑file \*\*processing\_state\*\* document records last successful stage, retries, and error snapshots.

#### 5.1.3 Throughput & Backpressure

* Work queues sized per tenant; HPA scales workers by queue depth/latency. Max parallelism per provider to respect API limits.

### 5.2 Pre‑filter & Metadata Extraction

* Normalizes provider metadata (file\_id, path, size, mime, author, modified\_time, permissions) and extracts text (Office/PDF).
* \*\*Progress recording\*\*: For every discovered file, update \*\*processing\_state\*\* in MongoDB with version/hash. On subsequent scans, the file is \*\*not scanned again unless these values change\*\*.
* Emits a canonical document for downstream summarization and embedding.

### 5.3 Sensitive Data Classification & Policies

#### 5.3.1 Admin‑Configurable “Sensitive Data” Definition (UI‑Driven)

Admin UI provides a \*\*policy editor\*\* to define what “sensitive data” means per tenant:

* \*\*Categories\*\*: PII, PCI, PHI, ITAR/EAR, Trade Secrets, Legal Privilege, HR.
* \*\*Detectors\*\*: Enable/disable built‑in regex/patterns or NLP classifiers; define \*\*custom regexes\*\*.
* \*\*Examples/Exclusions\*\*: Provide examples (e.g., SSN formats) and \*\*explicit exclusions\*\* (e.g., employee IDs like `E‑12345`).
* \*\*Enforcement\*\*: Choose behaviors on retrieval/answer: mask, block, alert, or allow with banner.

Policies are versioned and stored in MongoDB; a policy hash is stamped into each file’s metadata during classification.

#### 5.3.2 Optional Directory Selection

This feature is \*\*optional\*\*. When enabled, Admins see a \*\*browsable directory tree\*\* (via the MCP providers) and can select one or more folders to:

* Prioritize classification/summarization.
* Restrict the Q&A scope (query filter).
* Trigger ad‑hoc re‑processing for the selected subtrees.

If unused, the system processes according to global defaults.

### 5.4 Summarization & Embeddings

* \*\*Summaries\*\*: Azure OpenAI (GPT‑4‑class) produces a compact JSON summary with fields: title, purpose, entities, dates, table of contents (if any), and key facts.
* \*\*Embeddings\*\*: Azure OpenAI text‑embedding model. Chunking by semantic boundaries (target 1–2k tokens).
* \*\*Index Update\*\*: Upsert to Azure AI Search (vector + keyword) with file\_id, tenant\_id, path, sensitivity flags, and summary facets.

### 5.5 RAG Orchestrator & Teams Bot

\*\*Bot build & interfaces\*\*

* \*\*Registration\*\*: Azure Bot Service + Teams App manifest; Entra ID OAuth (Auth code w/ PKCE), scopes: profile, basic Teams APIs; Microsoft Graph minimal permissions for channel install and user attribution.
* \*\*Endpoints\*\*: `/api/messages` (Bot Framework), `/api/notify` (service‑to‑bot), `/api/healthz`.
* \*\*Secrets\*\*: Stored in Key Vault; injected via CSI Driver.

\*\*Query Flow\*\*

1. Parse user prompt; apply tenant policy (filters for sensitivity, directory scope if configured). 2) Retrieve top‑k from AI Search (vector + hybrid keyword). 3) Compose prompt (system + retrieval snippets) and call Azure OpenAI. 4) Post‑process.

\*\*Answer Rendering\*\*

* \*\*Text\*\*: Natural‑language answer with \*\*citations\*\* (file names/IDs, links to provider locations when available).
* \*\*Tables\*\*: When aggregations or lists are present, the model is instructed to emit a \*\*machine JSON payload\*\*. Orchestrator converts JSON to a Markdown table (Teams Adaptive Card) \*\*and\*\* stores the JSON alongside the answer log. A \*\*CSV\*\* is generated and attached for download.
* \*\*Graphs\*\*: For time‑series or categorical distributions, the orchestrator generates a \*\*server‑side chart (PNG)\*\* (e.g., line/bar/pie) using the same JSON payload and attaches it to the Teams card. The underlying data is preserved in the answer log.
* \*\*Guardrails\*\*: Sensitive fields masked per policy; disallowed content blocked with an explanatory banner.

### 5.6 Cost Metering & Forecast Service

* \*\*Meters\*\*: Token counts (prompt/completion/embeddings), AI Search operations, OCR/vision calls, container runtime hours.
* \*\*Storage\*\*: Cost events written to MongoDB (collection `cost\_usage`), aggregated hourly.
* \*\*Forecast\*\*: 7‑day moving average projected to period end with min/max bands; supports per‑feature drill‑down.

### 5.7 Admin Web UI

* \*\*Spend Dashboard\*\*: Configurable \*\*to‑date\*\* window (e.g., since project start), \*\*month‑to‑date\*\*, and \*\*projected month‑end\*\* spend; per‑feature breakdown; budgets with soft limits and alert thresholds; throttle batch jobs when thresholds hit.
* \*\*Policy Editor\*\*: Manage sensitive data definitions, examples, and exclusions (5.3.1).
* \*\*Directory Picker\*\*: Optional tree view to select/prioritize folders (5.3.2).
* \*\*Run Control\*\*: Start/stop ingestion, force incremental/full scans, re‑process failures.
* \*\*Answer Audit\*\*: Searchable Q&A history; view tables/graphs and download CSV payloads.

### 5.8 Data Model (MongoDB)

Collections & key fields:

* `files`: tenant\_id, file\_id (provider + id), path, mime, size, author, modified\_time, version/etag, content\_hash, sensitivity\_flags, summary\_ref.
* `processing\_state`: file\_id, stage (prefilter/summarize/embed/index), last\_success\_ts, retry\_count, error\_snapshot, cursors.
* `policies`: tenant\_id, version, definitions {categories, regexes, exclusions, behaviors}, created\_by, created\_at.
* `cost\_usage`: ts, tenant\_id, feature (ingest/summarize/embed/query), units, unit\_cost, amount.
* `qa\_logs`: question, answer, citations[], table\_json\_ref, chart\_ref, csv\_ref, cost\_snapshot.
* `embeddings`: file\_id, chunk\_id, vector\_ref/index\_id, metadata.

Indexes: compound on (tenant\_id, file\_id), (tenant\_id, path), and TTL on transient error logs.

## 6. External Systems & Connectors

* \*\*MCP Python Interpreter\*\*: Executes vetted traversal/util scripts in a sandbox. Tool whitelist only.
* \*\*Box MCP Server\*\*: Folder listing, file metadata/content fetch, delta streams.
* \*\*Microsoft files‑mcp‑server\*\*: SharePoint/OneDrive folder tree, file metadata/content, delta queries.
* \*\*Azure OpenAI\*\*: Chat/completions for summarization and QA; embeddings for retrieval.
* \*\*Azure AI Search\*\*: Vector + keyword hybrid retrieval.
* \*\*Azure Key Vault\*\*: Secrets, connection strings, certificates.
* \*\*Azure Blob Storage\*\*: Artifacts, chart images, CSVs, and immutable backups.

## 7. Security, Privacy & Compliance

* \*\*AuthN/Z\*\*: Entra ID; Admin UI RBAC (Owner, Operator, Auditor). Bot uses user identity; per‑tenant authorization enforced in orchestrator filters.
* \*\*Data Protection\*\*: AES‑256 at rest; TLS 1.2+ in transit; KMS‑managed keys.
* \*\*Sensitive Data Enforcement\*\*: Policies applied at retrieval time and answer rendering; masking and blocking rules logged.
* \*\*Auditability\*\*: Immutable Q&A logs with input/output, citations, and policy decisions.
* \*\*Data Residency\*\*: Region selection per tenant; co‑locate AI Search/OpenAI.

## 8. DevOps, IaC, and Repository Layout

### 8.1 Repo Structure

/docker/

bot/Dockerfile

orchestrator/Dockerfile

ingestion/Dockerfile

prefilter/Dockerfile

ai-pipeline/Dockerfile

admin-ui/Dockerfile

mcp-box-server/Dockerfile

mcp-files-server/Dockerfile

/infra/

azureai/provision\_azure\_openai.bicep

azureai/provision\_ai\_search.bicep

azureai/provision\_key\_vault.bicep

k8s/\*.yaml # Deployments, Services, HPA, CronJobs

/scripts/

teams/register\_bot.py

teams/configure\_channels.py

azure/set\_env\_keyvault.sh

azure/provision\_ai.sh

cost/export\_usage\_report.py

cost/recalculate\_forecast.py

ingestion/run\_traverse.py

ingestion/enqueue\_webhook\_event.py

backup/run\_backup.sh

restore/run\_restore.sh

### 8.2 Pipelines

* Build & push Docker images; run unit/integration tests; deploy to AKS (dev→stg→prod).
* Secrets from Key Vault via federated identity; no secrets in pipelines.

### 8.3 Environments

* \*\*Local\*\*: `docker-compose.yaml` spins up all services + local Mongo + Azurite (for dev).
* \*\*AKS\*\*: Namespaces per stage; HPA by CPU and queue depth; CronJobs for backups and index reconciliation.

## 9. Deployment & Configuration

* \*\*Teams Bot\*\*: Scripted registration (`/scripts/teams/register\_bot.py`); outputs App ID, password, and messaging endpoint; manifest packaged and published; optional `/scripts/teams/configure\_channels.py` to install to Teams/Channels.
* \*\*Azure AI Services\*\*: `provision\_azure\_openai.bicep` & `provision\_ai\_search.bicep`; `/scripts/azure/provision\_ai.sh` wires resources, alerts, and stores endpoints/keys in Key Vault.
* \*\*MCP Servers\*\*: Deployed as sidecars with health checks; provider credentials injected at start.
* \*\*Configuration\*\*: Per‑tenant JSON (policies, directory preferences, budgets) stored in DB; environment configuration via Key Vault references.

## 10. Observability & SRE

* \*\*Metrics\*\*: Queue depth, throughput, failure rates, scan latency, token usage, cost per feature.
* \*\*Logs\*\*: Structured JSON (OpenTelemetry); correlation IDs across components.
* \*\*Tracing\*\*: Ingestion pipeline spans; RAG request spans including retrieval and LLM calls.
* \*\*Dashboards & Alerts\*\*: Spend thresholds, ingestion stalls, elevated 5xx, policy violations.

## 11. Performance & Scalability Targets

* Ingestion: ≥ 1M files/day per region with p95 stage latency < 2s/file for text‑extractable formats.
* Q&A: p95 time‑to‑first‑token < 3s; p95 total answer < 10s for top‑k=8.
* Cost: Maintain cost per ingested file below configured threshold; automatic throttling when budget near limit.

## 12. Testing & Quality

* \*\*Unit\*\*: Policy engine, changelog logic, summarization prompts (golden tests), cost aggregation.
* \*\*Integration\*\*: MCP connector flows; AI Search upserts; Teams bot roundtrips.
* \*\*E2E\*\*: Seeded repositories; verify incremental scan correctness, policy enforcement, and Q&A outputs (tables/graphs included).
* \*\*Security\*\*: Secret‑scan, SAST, container image scan, dependency audit.

## 13. Risks & Mitigations

* \*\*Provider API limits\*\* → Adaptive rate limiting; retry with jitter; backoff and reschedule.
* \*\*LLM hallucinations\*\* → Strict citation requirement; retrieval‑only facts; confidence banners; guardrails.
* \*\*Cost overrun\*\* → Budgets, alerts, throttle batch jobs; forecast visible in UI.
* \*\*Data leakage\*\* → Policy enforcement before render; masking; deny‑lists; RBAC.

## 14. Roadmap (post‑v2)

* Additional connectors (Google Drive, S3 via MCP equivalents).
* Fine‑tuned classifiers for domain‑specific sensitivity.
* Report builder for recurring analytics across repositories.

## Appendix A — Teams App Manifest (template)

* Name, ID, bot endpoints, permissions, valid domains, OAuth settings.

## Appendix B — API Endpoints (selected)

* `POST /ingestion/start|stop` — admin‑auth only.
* `POST /ingestion/rescan` — {paths[], mode}.
* `GET /cost/summary` — to‑date, MTD, forecast.
* `POST /policy` — create/update policy.
* `POST /bot/notify` — admin broadcast.

## Appendix C — Answer Table/Graph JSON Schema

* `columns[]`, `rows[]`; `chart` { type, x, y, series }.

## Appendix D — Cost Calculation

* Token usage × model unit costs; AI Search Ops × tier rate; container‑hours × node price; OCR per page.

## Appendix E — CLI Reference

* `run\_traverse.py` (options), `enqueue\_webhook\_event.py` (payload), `export\_usage\_report.py` (filters).