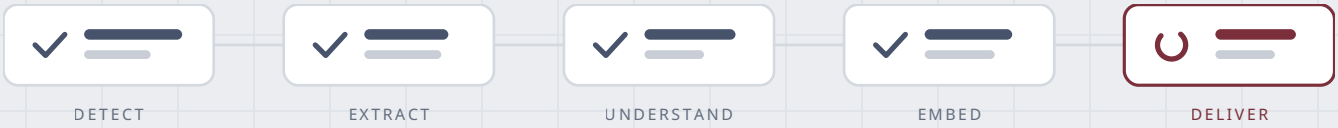
— // DOCUMENT INTELLIGENCE · SUSTAIND

Turning *hostile* documents into structured knowledge.

ARCHITECTURE & IMPLEMENTATION OF A PRODUCTION DOCUMENT-INGESTION WORKER · PYTHON / CLAUDE / AWS BEDROCK

sustaind, a compliance platform covering everything from sustainability law to product compliance and safety data sheets, answers regulatory questions from its customers' own documents. Those documents are scanned PDFs, legacy Word files, ten-thousand-row spreadsheets and screenshots — material no off-the-shelf parser reads reliably. I architected and built the worker that ingests all of them: a fault-tolerant pipeline that turns arbitrary uploads into semantically grouped chunks — searchable by vector, and traceable back to the exact pixels they came from.



01

One Worker, Every Format

PDF · DOC/DOCX · XLSX · PNG/JPEG

02

Three Extraction Engines

Layout, OCR, LLM vision — merged per page

03

Semantic Chunking

LLM-grouped, hierarchical, pixel-traceable

04

Built to Fail Safely

Circuit breakers, retries, dead-letter queue

01 CONTEXT

Compliance answers live inside customer paperwork.

sustaind helps companies manage regulatory compliance end to end — working out which laws apply, keeping products compliant, handling safety data sheets. The evidence behind every answer sits in the customer's own uploads: audit reports, policies, certificates, supplier questionnaires, safety data sheets. Before the platform's retrieval and LLM layers can reason over any of it, something has to convert those files into clean, structured, searchable data.

My engagement was to architect and implement that something: **Ingestor**, the Python service that sits between file upload and the platform's vector store, and owns everything in between — format detection, extraction, OCR, visual understanding, semantic chunking and embedding generation.

---

02 THE PROBLEM

Real-world documents fight back.

A naive "extract the text, split into chunks" approach fails on almost everything an enterprise actually uploads:

/01 **Scans without a text layer**
Certificates and signed audit reports arrive as photographed or scanned pages. There is no text to extract — only pixels.

/02 **Meaning that lives in visuals**
Charts, stamps, org diagrams and embedded figures carry information that plain text extraction silently drops.

/03 **Spreadsheets with structure only humans see**
A security questionnaire may hold five logical sections inside one 70-row table — separated by nothing but a bold row.

/04 **Legacy formats**
Binary .doc files from 2009 still show up in compliance workflows and still need to be read correctly.

/05 **Context destroyed by naive chunking**
Fixed-size text splitting cuts tables in half and separates answers from their questions — retrieval quality collapses.

/06 **Flaky, rate-limited dependencies**
The pipeline leans on S3 and LLM APIs. Throttling and outages must never lose a customer's document.

> **The hard constraint: traceability.** A compliance answer nobody can verify is worthless. Every chunk of extracted knowledge had to remain traceable to the exact region of the original document it came from — down to the pixel.

"Before a platform can *reason* over a document, something has to *read* it."

03 ARCHITECTURE

A queue-driven worker with pluggable pipelines.

The worker consumes ingestion jobs from a Redis queue, sniffs and validates the real MIME type of each file, and dispatches it to a format-specific pipeline resolved from a registry. Four production pipelines — PDF, Word, spreadsheet, image — share one execution model: extract, understand, chunk, embed, persist. Every intermediate artifact is written to S3 under a deterministic per-document layout, so any run can be inspected, replayed or reprocessed.

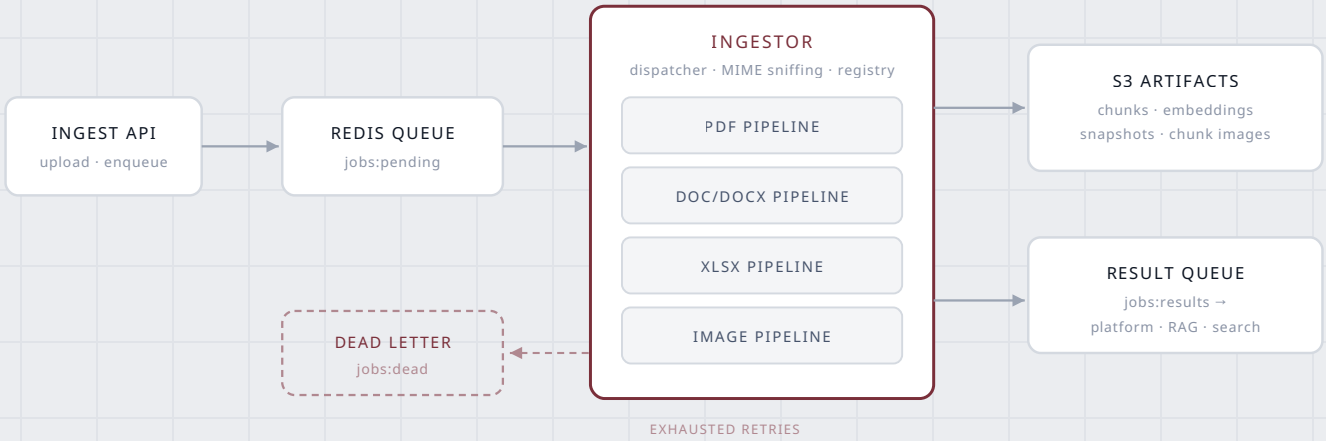


FIG. 01 — INGESTION TOPOLOGY. PIPELINES REGISTER AGAINST MIME TYPES; ADDING A FORMAT TOUCHES NO EXISTING CODE.

04 EXTRACTION

No single engine can read everything — so three run together.

Each PDF page passes through three complementary extractors. **PyMuPDF** pulls native text blocks with exact bounding boxes and renders a page snapshot. **Tesseract OCR** steps in for image-heavy or scanned pages where no text layer exists. **Claude vision** (via AWS Bedrock) reads the rendered page like a human — identifying tables, charts, logos, stamps and figures, and describing what they mean.

The three results merge into a single per-page model: machine-precise geometry from the parser, text recovered from pixels, and semantic understanding from the LLM. The same engine trio backs the image pipeline; Word files add a LibreOffice-headless conversion step for legacy .doc uploads.

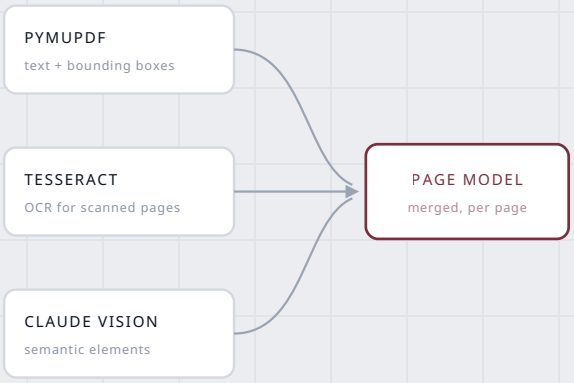


FIG. 02 — THREE ENGINES, ONE MERGED PAGE MODEL.

Chunks a retrieval system can trust — and a human can verify.

Instead of splitting text at arbitrary character counts, an **LLM-driven chunker** reads the merged page models across the whole document and groups related elements into semantic chunks — flat, or hierarchical with parent summaries above child detail. A table stays a table; a question stays attached to its answer.

Crucially, every element's bounding box travels with its chunk. After chunking, the worker merges each chunk's boxes per page, crops that region from the page snapshot and uploads it to S3.

Any answer the platform gives can show the exact patch of the original document it came from.

FIG. 03 — HIERARCHICAL CHUNKS; EACH CARRIES ITS SOURCE COORDINATES.

Chunk text is then embedded with Amazon Titan Text Embeddings V2 — 1024-dimension, normalized vectors, generated in concurrent batches — and shipped alongside the chunks for the platform's vector store.

Spreadsheets: finding the structure only humans see.

Spreadsheets got their own intelligence layer. Sheets are analyzed by Claude in **sliding windows of 250 rows with 50 rows of overlap**, so tables spanning window boundaries are detected whole. The model is fed multimodally — raw cell data plus any embedded images — and returns typed regions: tables, text blocks, headings, lists, illustrations.

Within tables it identifies **semantic groups** — the logical sections a human sees in a bold divider row. A 70-row security questionnaire becomes parents like `"Identity & Access Management"` with each requirement row as a retrievable child chunk. Overlapping detections from adjacent windows are merged deterministically before chunking.

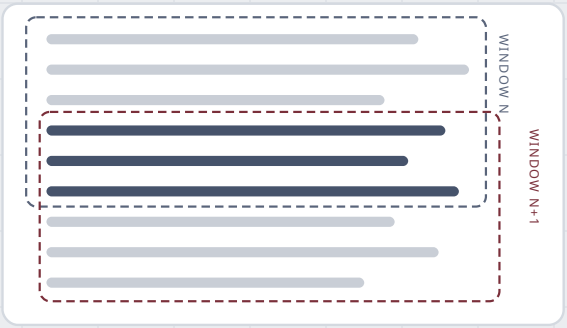


FIG. 04 — SLIDING-WINDOW REGION DETECTION OVER LARGE SHEETS.

06 PRODUCTION HARDENING

Failures happen. Documents are never lost.

- **Circuit breakers** guard S3 and Bedrock: five consecutive failures open the circuit and fail fast; after a cool-down (60 s / 120 s) a half-open probe lets two successes close it again.
- **Retries with exponential backoff and jitter** — up to five attempts per job, with throttle-aware delays for LLM rate limits.
- **Dead-letter queue**: jobs that exhaust retries land in `jobs:dead` with full context — no silent failures, ever.
- **Cost discipline**: concurrency caps on every LLM and embedding call, a provider abstraction that swaps Bedrock for a local llama.cpp server in development (zero cloud spend), a fallback model on malformed LLM output, and per-job cost accounting in each result.
- **Observability**: structured logging throughout, Sentry error tracking and performance tracing, graceful shutdown that drains in-flight jobs.

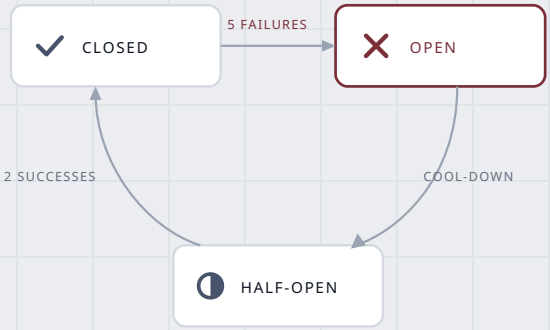


FIG. 05 — PER-DEPENDENCY CIRCUIT BREAKER (S3, BEDROCK).

07 OUTCOME

One worker, every document, every answer verifiable.

04	03	0	100%
PRODUCTION PIPELINES — PDF, WORD, EXCEL, IMAGES	EXTRACTION ENGINES MERGED PER PAGE	SILENT FAILURES — RETRIES, BREAKERS, DEAD-LETTER QUEUE	OF CHUNKS TRACEABLE TO SOURCE PIXELS

Ingestor now handles every format customers upload and delivers semantically grouped chunks with embeddings to the platform — each one traceable to its source. New formats slot in as registered pipeline modules without touching existing code, and every processing step stays inspectable in S3 after the fact.

STACK — Python 3.11 · Redis · AWS S3 · AWS Bedrock (Claude) · Amazon Titan Embeddings V2 · llama.cpp · PyMuPDF · Tesseract · LibreOffice · python-docx · openpyxl · Pillow · structlog · Sentry · Kubernetes / Tilt

Have documents your software can't read?

I design and build document-intelligence and LLM pipelines end to end — from queue to vector store, architecture to production hardening.

START A PROJECT ↗

hello@bastian-schoentag.de