Take-Home Assignment: Document Ingestion Service

Background

At Docket, our AI agents answer complex product and technical questions by drawing from a company's knowledge base — sales decks, product docs, support articles, technical specs, and more.

To power this, we need a robust **document ingestion system** that processes uploaded files and makes their content searchable using vector similarity.

Your task is to design and build a working prototype of this system.

Problem Statement

Build a service that allows users to:

1. **Upload documents** (PDF and plain text files) via an API
2. **Process the content** — extract text, chunk it appropriately, generate embeddings, and store in a vector database
3. **Search the content** — given a query, return the most semantically relevant portions of ingested documents

---

Technical Constraints

- **Vector Store:** Use **Qdrant** as your vector database
- **Embeddings:** Use any embedding model/API of your choice (OpenAI, Cohere, open-source, or mock embeddings for the prototype)
- **Multi-tenancy:** The system must support multiple tenants (customers), with strict data isolation between them
- **Containerization:** The implementation must be Dockerized

---

Requirements

Functional

- Support PDF (.pdf) and plain text (.txt) file uploads
- Handle documents ranging from 1 page to 100+ pages
- Support both **single file uploads** and **bulk uploads** (e.g., multiple files at once, or via integrations like Google Drive)
- Search should return relevant *excerpts/chunks*, not entire documents
- Each result should reference its source document and tenant
- Tenants must only see/search their own documents
- **Authentication:** The service should be authenticated so other internal services can securely call its APIs

Scale Considerations

Design with the following targets in mind:

Metric	Target
Total documents (all tenants)	1,000,000+
Documents per tenant	Up to 50,000
Ingestion throughput	50,000 documents per tenant in < 24 hours
Search latency	Responsive at scale

You don't need to prove these numbers in your prototype, but your design should explain **how** it would achieve them.

Fairness & Resource Management

- The system should ensure **fairness across tenants** — a large bulk upload from one customer should not starve or significantly delay processing for other customers
- Consider how you would prioritize or rate-limit work across tenants

Non-Functional

- Handle concurrent uploads without blocking
- Failures during processing should not result in data loss or inconsistent state
- Consider observability — how would you know if something is wrong?

Deliverables

Required

1. **Working code** — a runnable service using **Docker Compose** (should include Qdrant and all dependencies)
2. **API documentation** — how to authenticate, upload (single & bulk), check processing status, and search
3. **High-level architecture diagram** — showing components, data flow, and how they interact
4. **ER diagram** — showing your data model (database tables, Qdrant collections, relationships)
5. **Design document (README or separate doc)** covering:
    - How you approached multi-tenancy in Qdrant and why
    - Your chunking and embedding strategy
    - How you ensure fairness across tenants during bulk ingestion
    - How the system would scale to the targets above
    - How you'd handle failures and retries
    - Authentication approach
    - What you'd change or add for production readiness
    - Trade-offs you made

Bonus (Optional)

- **Terraform scripts** for deploying the infrastructure (AWS preferred)

Evaluation Criteria

Area	What we're looking for
System Design	Multi-tenancy approach, data modeling in Qdrant, scalability thinking, fairness mechanisms
Problem Solving	Chunking strategy, handling large docs, bulk uploads, failure modes
Code Quality	Readability, structure, error handling, tests
Scalability	Realistic path to handling 1M+ docs, ingestion throughput, tenant fairness
Documentation	Clear diagrams, articulation of decisions, trade-offs, and production considerations
Pragmatism	A working prototype with a clear roadmap > overengineered incomplete solution

---

Time Expectation

We expect this to take **2 days of effort**. Focus on demonstrating your thinking — it's fine to simplify parts (e.g., mock embeddings, basic auth) and document what you'd do differently in production.

Submission

- GitHub repo (private is fine — add collaborators we specify) or zip file
- Include instructions to run locally with docker-compose up
- Ensure all diagrams are included (PNG/PDF or embedded in docs)

Good luck! We're excited to see your approach.