Miscs

- Real-Time Support Ticket Intelligence Pipeline

Real-Time Support Ticket Intelligence Pipeline

Scenario

You're joining "TicketPulse," a SaaS company handling ~50,000 support tickets/day across email, chat, and social. Support agents currently triage everything manually. Your job: build the backbone of a system that ingests tickets, uses an LLM to classify and enrich them, and makes them searchable and filterable for agents and analysts in near real time.

Required stack

- **FastAPI** : ingestion endpoint + query API
- **Kafka** : event backbone between ingestion and processing
- **Celery** : async enrichment workers
- **An LLM** : (any provider, or a mocked/local stand-in that mimics real latency/failure behavior) classification and reply drafting
- **Elasticsearch** : indexed store for search, filtering, and basic analytics

Core requirements

1. **Ingestion API** : `POST /tickets` accepts a raw ticket (customer id, channel, subject, body, timestamp) and publishes it onto a Kafka topic. Should return immediately, not block on processing.
2. **Enrichment pipeline** : consumes from Kafka, and for each ticket:
   - Deduplicates (e.g., same customer + similar content within a short window)
   - Calls the LLM to classify: category, urgency (low/med/high), sentiment
   - Generates a short suggested-reply draft
   - Handles LLM failures without losing the ticket (retry, then fall back to a "needs manual triage" state, never silently drop)
3. **Indexing** : enriched ticket lands in Elasticsearch with a mapping that supports both full-text search on the body and structured filtering (category, urgency, sentiment, date

range).
4. **Query API** : `GET /tickets/search` supporting full-text search + filters; bonus if it supports "similar past tickets" for a given ticket id.
5. **Idempotency** : replaying the same Kafka message twice must not double-index or double-notify.

Explicitly out of scope

- No frontend/UI, API responses are enough
- No auth/multi-tenancy
- No actual load testing at 50k/day, reason about it, don't prove it
- No fine-tuned model, prompting an off-the-shelf LLM (or a stub) is fine

Sample data

Use a public dataset like Kaggle's "Customer Support on Twitter" for realistic ticket text, or generate synthetic tickets with an LLM. Either is fine, just note which one and why in your README.

Deliverables

1. **Working repo** : `docker-compose up` should bring up Kafka, Elasticsearch, Celery worker(s), broker, and the FastAPI app.
2. **README** : how to run it, and the key design decisions you made (not a commit-by-commit history).
3. **ARCHITECTURE.md** : short written answers to:
   - How does this scale to 10x ticket volume? What's the first thing that breaks?
   - How do you control LLM cost at this volume?
   - What's your dead-letter / poison-message strategy when a ticket keeps failing enrichment?
   - What would you monitor in production, and what would page someone at 2am vs. just show up on a dashboard?
   - What did you deliberately cut given the time box, and what would you build next?

Optional stretch

- Dead-letter queue with a visible retry/inspection path

- Cost-aware LLM routing (cheap model for obvious cases, escalate only when uncertain)
- Semantic "similar tickets" via embeddings instead of ES `more_like_this`
- Basic metrics/tracing

Submission

Send a link to the repo (or a zip) with the README and ARCHITECTURE.md included. Reach out if any requirement is ambiguous, how you handle ambiguity is also part of what we're looking at.