Data Ops Technical Challenge – ETL & Serving Pipeline

Objective

Design **and** (optionally) prototype a small end-to-end data platform that:

1. **Ingests** data from a handful of predefined public URLs.
2. **Processes & cleans** that data via an ETL workflow.
3. **Stores** the curated results in **PostgreSQL**.
4. **(Stretch) Serves** the curated data through HTTPS endpoints.

> **Hard requirement:** orchestrate all ingestion & processing with **Apache Airflow (v2.9 +)**.
>
> **Time-box:** the entire exercise is intended to take **≈ 8 hours**. Focus first on architectural clarity; code beyond a minimal PoC is optional.

Requirements

Functional

ID	Description	Level
F-1	*Extract Raw Data* – fetch the contents of each URL and stage them as raw files or tables.	Core
F-2	*Transform & Clean* – apply at least one meaningful transformation (type-cast, filter, aggregation, etc.).	Core
F-3	*Load to Postgres* – upsert or append curated data into well-designed tables.	Core
F-4	*Expose REST Endpoints* – HTTP routes to list records, fetch by ID, query by date range.	**Stretch**
F-5	*Schedule* – pipeline runs daily (cron) **or** on-demand with retries on failure.	Core

Technical

ID	Requirement	Level
T-1	**Airflow DAGs** must be idempotent and parameterized.	Core
T-2	Target database: **PostgreSQL 15 +**.	Core

T-3	Containerized local stack via **Docker Compose** (Airflow + Postgres + optional API).	**Stretch**
T-4	Implementation language: **Python 3.11 +** (you may use `requests`, `pandas`, `FastAPI`, etc.).	Core
T-5	Provide a **README** that boots the solution in < 5 min.	Core
T-6	Use **Git** with logical, incremental commits.	Core

Deliverables

Level	What to Turn In
Mandatory	**Design Document** (Google Doc or Word). Link it in your repo's README.
Bonus	**Minimal PoC** – Docker Compose stack, single Airflow DAG, *optional* API endpoint returning data.

Design Document Template

Core sections (1–4) are required. Sections 5–9 are *additional* – include them if time allows.

1. **Executive Summary** *(Core)* – one paragraph describing the solution's value.
2. **Data Source Overview** *(Core)* – URLs, formats, expected frequency/volume.
3. **High-Level Architecture Diagram** *(Core)* – major components & data flow (Mermaid or draw.io).
4. **Airflow DAG Design** *(Core)* – task graph, scheduling, idempotency, retries/alerts.
5. **Data Modeling** *(Additional)* – DDL, keys, indexes, partitioning.
6. **Serving Layer** *(Additional)* – API design, endpoints, versioning.
7. **Observability & Alerting** *(Additional)* – logging, metrics, SLAs, health-checks.
8. **Infrastructure & CI/CD** *(Additional)* – dev environment, env vars, deployment outline.
9. **Security & Compliance** *(Additional)* – TLS, secrets storage, least-privilege DB access.

Evaluation Criteria

Weight	Area	What We Look For
40 %	**Architecture Clarity**	Clean diagram & narrative, clear boundaries, articulated trade-offs, failure recovery plan.
25 %	**Airflow DAG Quality**	Idempotent, parameterized, retries, alerting, readable code.
15 %	**Data Model**	Well-normalized where sensible, appropriate keys & indexes.
10 %	**Serving Layer**	Logical resource naming, pagination, versioning, (basic auth if implemented).
10 %	**Document Quality**	Concise, structured, actionable next steps.
+5 %	**Prototype Bonus**	Docker stack spins up, DAG run succeeds, endpoint returns data.

Remember: Architectural reasoning & trade-offs outweigh code volume. Deliver the clearest solution you can within the 8-hour window.