Below is an extended, end‑to‑end example that not only implements the core pipeline stages (extraction, loading, transformation, metadata ingestion, dashboarding) but also adds extra stages for quality validation, enrichment/aggregation, and lineage/logging activities. This enhanced pipeline uses additional open‑source libraries and practices from modern data engineering to help ensure reliability, observability, and data usability.

In this example you’ll see two new stages added as Airflow tasks:

1. **Data Quality Validation:**  
   We use [Great Expectations](https://docs.greatexpectations.io/) to run a set of automated quality checks on the raw and/or normalized data. This ensures that key business columns are not null, values lie within acceptable ranges, and the extracted data meets our expectations.
2. **Data Enrichment and KPI Aggregation:**  
   After the DBT transformation, a new enrichment task computes aggregate KPIs (for example, sales per category, review sentiment scores, or customized metrics) and writes these results into a dedicated table. This stage makes the data immediately useful for the dashboard and for operational reporting.

Below is the revised **directory structure** and code for each pipeline stage.

**Directory Structure**

project-root/

│

├── docker-compose.yml

├── dags/

│ ├── data\_pipeline\_dag.py

│ ├── extract\_to\_minio.py

│ ├── load\_to\_postgres.py

│ ├── data\_quality\_check.py

│ ├── data\_enrichment.py

│ └── dbt\_project/

│ ├── dbt\_project.yml

│ ├── models/

│ │ └── raw\_to\_normalized.sql

│ └── schema.yml

├── data\_source/

│ └── sample\_data.csv

├── datahub\_ingestion.yml

└── README.md

**1. Docker‑Compose Environment**

Below is the updated docker-compose.yml file that spins up our entire stack. (This example uses PostgreSQL as the data warehouse, MinIO for object storage, Airflow for orchestration, DBT for transformations, DataHub for metadata management, and Superset for visualization.)

# docker-compose.yml

# version: 3.8 # version is obsolete future docker development

networks:

  default:

    name: datahub\_network

services:

# Data Warehouse (for DBT transformations, etc.)

  postgres\_dw:

    image: postgres:15

    container\_name: postgres\_dw

    environment:

      POSTGRES\_USER: dwh\_user

      POSTGRES\_PASSWORD: dwh\_password

      POSTGRES\_DB: datamart

    ports:

      - "5432:5432"

    volumes:

      - postgres\_dw\_data:/var/lib/postgresql/data

# Object Storage (MinIO)

  minio:

    image: minio/minio

    container\_name: minio\_oss

    environment:

      MINIO\_ROOT\_USER: minioadmin

      MINIO\_ROOT\_PASSWORD: minioadmin

    command: server /data --console-address ":9001"

    ports:

      - "9000:9000"

      - "9001:9001"

    volumes:

      - minio\_data:/data

# Airflow Metadata DB (separate from postgres\_dw)

  postgres\_airflow:

    image: postgres:13

    container\_name: postgres\_airflow

    environment:

      POSTGRES\_USER: airflow

      POSTGRES\_PASSWORD: airflow

      POSTGRES\_DB: airflow

    healthcheck:

      test: ["CMD", "pg\_isready", "-U", "airflow"]

      interval: 10s

      retries: 5

      start\_period: 5s

    restart: always

    ports:

      - "5439:5432"

    volumes:

      - postgres\_airflow\_data:/var/lib/postgresql/data

  redis:

    # Redis is limited to 7.2-bookworm due to licencing change

    # https://redis.io/blog/redis-adopts-dual-source-available-licensing/

    image: redis:7.2-bookworm

    expose:

      - 6379

    healthcheck:

      test: ["CMD", "redis-cli", "ping"]

      interval: 10s

      timeout: 30s

      retries: 50

      start\_period: 30s

    restart: always

  airflow:

    image: apache/airflow:2.3.4

    container\_name: airflow

    restart: always

    environment:

      AIRFLOW\_\_CORE\_\_EXECUTOR: LocalExecutor

      AIRFLOW\_\_DATABASE\_\_SQL\_ALCHEMY\_CONN: postgresql+psycopg2://airflow:airflow@postgres\_airflow/airflow

      AIRFLOW\_\_CORE\_\_LOAD\_EXAMPLES: "false"

    depends\_on:

      - postgres\_airflow

    ports:

      - "8082:8082"

    volumes:

      - ./dags:/opt/airflow/dags

    command: webserver

  broker:

    depends\_on:

      zookeeper:

        condition: service\_healthy

    environment:

      - KAFKA\_BROKER\_ID=1

      - KAFKA\_ZOOKEEPER\_CONNECT=zookeeper:2181

      - KAFKA\_LISTENER\_SECURITY\_PROTOCOL\_MAP=PLAINTEXT:PLAINTEXT,PLAINTEXT\_HOST:PLAINTEXT

      - KAFKA\_ADVERTISED\_LISTENERS=PLAINTEXT://broker:29092,PLAINTEXT\_HOST://localhost:9092

      - KAFKA\_OFFSETS\_TOPIC\_REPLICATION\_FACTOR=1

      - KAFKA\_GROUP\_INITIAL\_REBALANCE\_DELAY\_MS=0

      - KAFKA\_HEAP\_OPTS=-Xms256m -Xmx256m

      - KAFKA\_CONFLUENT\_SUPPORT\_METRICS\_ENABLE=false

      - KAFKA\_MESSAGE\_MAX\_BYTES=5242880

      - KAFKA\_MAX\_MESSAGE\_BYTES=5242880

    healthcheck:

      interval: 1s

      retries: 5

      start\_period: 60s

      test: nc -z broker $${DATAHUB\_KAFKA\_BROKER\_PORT:-9092}

      timeout: 5s

    hostname: broker

    image: ${DATAHUB\_CONFLUENT\_KAFKA\_IMAGE:-confluentinc/cp-kafka}:${DATAHUB\_CONFLUENT\_VERSION:-7.4.0}

    ports:

    - ${DATAHUB\_MAPPED\_KAFKA\_BROKER\_PORT:-9092}:9092

    volumes:

    - broker:/var/lib/kafka/data/

  datahub-actions:

    depends\_on:

      datahub-gms:

        condition: service\_healthy

    environment:

    - ACTIONS\_CONFIG=${ACTIONS\_CONFIG:-}

    - ACTIONS\_EXTRA\_PACKAGES=${ACTIONS\_EXTRA\_PACKAGES:-}

    - DATAHUB\_GMS\_HOST=datahub-gms

    - DATAHUB\_GMS\_PORT=8080

    - DATAHUB\_GMS\_PROTOCOL=http

    - DATAHUB\_SYSTEM\_CLIENT\_ID=\_\_datahub\_system

    - DATAHUB\_SYSTEM\_CLIENT\_SECRET=JohnSnowKnowsNothing

    - KAFKA\_BOOTSTRAP\_SERVER=broker:29092

    - KAFKA\_PROPERTIES\_SECURITY\_PROTOCOL=PLAINTEXT

    - METADATA\_AUDIT\_EVENT\_NAME=MetadataAuditEvent\_v4

    - METADATA\_CHANGE\_LOG\_VERSIONED\_TOPIC\_NAME=MetadataChangeLog\_Versioned\_v1

    - SCHEMA\_REGISTRY\_URL=http://schema-registry:8081

    hostname: actions

    image: ${DATAHUB\_ACTIONS\_IMAGE:-acryldata/datahub-actions}:${ACTIONS\_VERSION:-head}

  datahub-frontend-react:

    depends\_on:

      datahub-gms:

        condition: service\_healthy

    environment:

    - DATAHUB\_GMS\_HOST=datahub-gms

    - DATAHUB\_GMS\_PORT=8080

    - DATAHUB\_SECRET=YouKnowNothing

    - DATAHUB\_APP\_VERSION=1.0

    - DATAHUB\_PLAY\_MEM\_BUFFER\_SIZE=10MB

    - JAVA\_OPTS=-Xms512m -Xmx512m -Dhttp.port=9002 -Dconfig.file=datahub-frontend/conf/application.conf -Djava.security.auth.login.config=datahub-frontend/conf/jaas.conf -Dlogback.configurationFile=datahub-frontend/conf/logback.xml -Dlogback.debug=false -Dpidfile.path=/dev/null

    - KAFKA\_BOOTSTRAP\_SERVER=broker:29092

    - DATAHUB\_TRACKING\_TOPIC=DataHubUsageEvent\_v1

    - ELASTIC\_CLIENT\_HOST=elasticsearch

    - ELASTIC\_CLIENT\_PORT=9200

    hostname: datahub-frontend-react

    image: ${DATAHUB\_FRONTEND\_IMAGE:-acryldata/datahub-frontend-react}:${DATAHUB\_VERSION:-head}

    ports:

    - ${DATAHUB\_MAPPED\_FRONTEND\_PORT:-9002}:9002

    volumes:

    - ${HOME}/.datahub/plugins:/etc/datahub/plugins

  datahub-gms:

    depends\_on:

      datahub-upgrade:

        condition: service\_completed\_successfully

    environment:

    - DATAHUB\_SERVER\_TYPE=${DATAHUB\_SERVER\_TYPE:-quickstart}

    - DATAHUB\_TELEMETRY\_ENABLED=${DATAHUB\_TELEMETRY\_ENABLED:-true}

    - DATAHUB\_UPGRADE\_HISTORY\_KAFKA\_CONSUMER\_GROUP\_ID=generic-duhe-consumer-job-client-gms

    - EBEAN\_DATASOURCE\_DRIVER=com.mysql.jdbc.Driver

    - EBEAN\_DATASOURCE\_HOST=mysql:3306

    - EBEAN\_DATASOURCE\_PASSWORD=datahub

    - EBEAN\_DATASOURCE\_URL=jdbc:mysql://mysql:3306/datahub?verifyServerCertificate=false&useSSL=true&useUnicode=yes&characterEncoding=UTF-8&enabledTLSProtocols=TLSv1.2

    - EBEAN\_DATASOURCE\_USERNAME=datahub

    - ELASTICSEARCH\_HOST=elasticsearch

    - ELASTICSEARCH\_INDEX\_BUILDER\_MAPPINGS\_REINDEX=true

    - ELASTICSEARCH\_INDEX\_BUILDER\_SETTINGS\_REINDEX=true

    - ELASTICSEARCH\_PORT=9200

    - ENTITY\_REGISTRY\_CONFIG\_PATH=/datahub/datahub-gms/resources/entity-registry.yml

    - ENTITY\_SERVICE\_ENABLE\_RETENTION=true

    - ES\_BULK\_REFRESH\_POLICY=WAIT\_UNTIL

    - GRAPH\_SERVICE\_DIFF\_MODE\_ENABLED=true

    - GRAPH\_SERVICE\_IMPL=${GRAPH\_SERVICE\_IMPL:-elasticsearch}

    - JAVA\_OPTS=-Xms1g -Xmx1g

    - KAFKA\_BOOTSTRAP\_SERVER=broker:29092

    - KAFKA\_CONSUMER\_STOP\_ON\_DESERIALIZATION\_ERROR=${KAFKA\_CONSUMER\_STOP\_ON\_DESERIALIZATION\_ERROR:-true}

    - KAFKA\_SCHEMAREGISTRY\_URL=http://schema-registry:8081

    - MAE\_CONSUMER\_ENABLED=true

    - MCE\_CONSUMER\_ENABLED=true

    - METADATA\_SERVICE\_AUTH\_ENABLED=${METADATA\_SERVICE\_AUTH\_ENABLED:-false}

    - NEO4J\_HOST=http://neo4j:7474

    - NEO4J\_PASSWORD=datahub

    - NEO4J\_URI=bolt://neo4j

    - NEO4J\_USERNAME=neo4j

    - PE\_CONSUMER\_ENABLED=true

    - THEME\_V2\_DEFAULT=true

    - UI\_INGESTION\_ENABLED=true

    healthcheck:

      interval: 1s

      retries: 3

      start\_period: 90s

      test: curl -sS --fail http://datahub-gms:${DATAHUB\_GMS\_PORT:-8080}/health

      timeout: 5s

    hostname: datahub-gms

    image: ${DATAHUB\_GMS\_IMAGE:-acryldata/datahub-gms}:${DATAHUB\_VERSION:-head}

    ports:

    - ${DATAHUB\_MAPPED\_GMS\_PORT:-8080}:8080

    volumes:

    - ${HOME}/.datahub/plugins:/etc/datahub/plugins

  datahub-upgrade:

    command:

    - -u

    - SystemUpdate

    depends\_on:

      elasticsearch-setup:

        condition: service\_completed\_successfully

      kafka-setup:

        condition: service\_completed\_successfully

      mysql-setup:

        condition: service\_completed\_successfully

      neo4j:

        condition: service\_healthy

    environment:

    - BACKFILL\_BROWSE\_PATHS\_V2=true

    - DATAHUB\_GMS\_HOST=datahub-gms

    - DATAHUB\_GMS\_PORT=8080

    - EBEAN\_DATASOURCE\_DRIVER=com.mysql.jdbc.Driver

    - EBEAN\_DATASOURCE\_HOST=mysql:3306

    - EBEAN\_DATASOURCE\_PASSWORD=datahub

    - EBEAN\_DATASOURCE\_URL=jdbc:mysql://mysql:3306/datahub?verifyServerCertificate=false&useSSL=true&useUnicode=yes&characterEncoding=UTF-8

    - EBEAN\_DATASOURCE\_USERNAME=datahub

    - ELASTICSEARCH\_BUILD\_INDICES\_CLONE\_INDICES=false

    - ELASTICSEARCH\_HOST=elasticsearch

    - ELASTICSEARCH\_INDEX\_BUILDER\_MAPPINGS\_REINDEX=true

    - ELASTICSEARCH\_INDEX\_BUILDER\_SETTINGS\_REINDEX=true

    - ELASTICSEARCH\_PORT=9200

    - ENTITY\_REGISTRY\_CONFIG\_PATH=/datahub/datahub-gms/resources/entity-registry.yml

    - GRAPH\_SERVICE\_IMPL=${GRAPH\_SERVICE\_IMPL:-elasticsearch}

    - KAFKA\_BOOTSTRAP\_SERVER=broker:29092

    - KAFKA\_SCHEMAREGISTRY\_URL=http://schema-registry:8081

    - REPROCESS\_DEFAULT\_BROWSE\_PATHS\_V2=false

    hostname: datahub-upgrade

    image: ${DATAHUB\_UPGRADE\_IMAGE:-acryldata/datahub-upgrade}:${DATAHUB\_VERSION:-head}

    labels:

      datahub\_setup\_job: true

  elasticsearch:

    deploy:

      resources:

        limits:

          memory: 1G

    environment:

    - discovery.type=single-node

    - ${XPACK\_SECURITY\_ENABLED:-xpack.security.enabled=false}

    - ES\_JAVA\_OPTS=-Xms256m -Xmx512m -Dlog4j2.formatMsgNoLookups=true

    - OPENSEARCH\_JAVA\_OPTS=-Xms512m -Xmx512m -Dlog4j2.formatMsgNoLookups=true

    healthcheck:

      interval: 1s

      retries: 3

      start\_period: 20s

      test: curl -sS --fail http://elasticsearch:$${DATAHUB\_ELASTIC\_PORT:-9200}/\_cluster/health?wait\_for\_status=yellow&timeout=0s

      timeout: 5s

    hostname: elasticsearch

    image: ${DATAHUB\_SEARCH\_IMAGE:-elasticsearch}:${DATAHUB\_SEARCH\_TAG:-7.10.1}

    ports:

    - ${DATAHUB\_MAPPED\_ELASTIC\_PORT:-9200}:9200

    volumes:

    - esdata:/usr/share/elasticsearch/data

  elasticsearch-setup:

    depends\_on:

      elasticsearch:

        condition: service\_healthy

    environment:

    - ELASTICSEARCH\_USE\_SSL=${ELASTICSEARCH\_USE\_SSL:-false}

    - USE\_AWS\_ELASTICSEARCH=${USE\_AWS\_ELASTICSEARCH:-false}

    - ELASTICSEARCH\_HOST=elasticsearch

    - ELASTICSEARCH\_PORT=9200

    - ELASTICSEARCH\_PROTOCOL=http

    hostname: elasticsearch-setup

    image: ${DATAHUB\_ELASTIC\_SETUP\_IMAGE:-acryldata/datahub-elasticsearch-setup}:${DATAHUB\_VERSION:-head}

    labels:

      datahub\_setup\_job: true

  kafka-setup:

    depends\_on:

      broker:

        condition: service\_healthy

      schema-registry:

        condition: service\_healthy

    environment:

    - DATAHUB\_PRECREATE\_TOPICS=${DATAHUB\_PRECREATE\_TOPICS:-false}

    - KAFKA\_BOOTSTRAP\_SERVER=broker:29092

    - KAFKA\_ZOOKEEPER\_CONNECT=zookeeper:2181

    - USE\_CONFLUENT\_SCHEMA\_REGISTRY=TRUE

    hostname: kafka-setup

    image: ${DATAHUB\_KAFKA\_SETUP\_IMAGE:-acryldata/datahub-kafka-setup}:${DATAHUB\_VERSION:-head}

    labels:

      datahub\_setup\_job: true

  mysql:

    command: --character-set-server=utf8mb4 --collation-server=utf8mb4\_bin --default-authentication-plugin=mysql\_native\_password

    environment:

    - MYSQL\_DATABASE=datahub

    - MYSQL\_USER=datahub

    - MYSQL\_PASSWORD=datahub

    - MYSQL\_ROOT\_PASSWORD=datahub

    healthcheck:

      interval: 1s

      retries: 3

      start\_period: 10s

      test: mysqladmin ping -h mysql -u $$MYSQL\_USER --password=$$MYSQL\_PASSWORD

      timeout: 5s

    hostname: mysql

    image: mysql:8.0

    ports:

      - "3306:3306"

    restart: on-failure

    volumes:

    - mysqldata:/var/lib/mysql

  mysql-setup:

    depends\_on:

      mysql:

        condition: service\_healthy

    environment:

    - MYSQL\_HOST=mysql

    - MYSQL\_PORT=3306

    - MYSQL\_USERNAME=datahub

    - MYSQL\_PASSWORD=datahub

    - DATAHUB\_DB\_NAME=datahub

    hostname: mysql-setup

    image: ${DATAHUB\_MYSQL\_SETUP\_IMAGE:-acryldata/datahub-mysql-setup}:${DATAHUB\_VERSION:-head}

    labels:

      datahub\_setup\_job: true

  neo4j:

    environment:

    - NEO4J\_AUTH=neo4j/datahub

    - NEO4J\_dbms\_default\_\_database=graph.db

    - NEO4J\_dbms\_allow\_\_upgrade=true

    - NEO4JLABS\_PLUGINS=["apoc"]

    healthcheck:

      interval: 1s

      retries: 5

      start\_period: 5s

      test: wget http://neo4j:$${DATAHUB\_NEO4J\_HTTP\_PORT:-7474}

      timeout: 5s

    hostname: neo4j

    image: neo4j:4.4.9-community

    ports:

    - ${DATAHUB\_MAPPED\_NEO4J\_HTTP\_PORT:-7474}:7474

    - ${DATAHUB\_MAPPED\_NEO4J\_BOLT\_PORT:-7687}:7687

    volumes:

    - neo4jdata:/data

  schema-registry:

    depends\_on:

      broker:

        condition: service\_healthy

    environment:

    - SCHEMA\_REGISTRY\_HOST\_NAME=schemaregistry

    - SCHEMA\_REGISTRY\_KAFKASTORE\_SECURITY\_PROTOCOL=PLAINTEXT

    - SCHEMA\_REGISTRY\_KAFKASTORE\_BOOTSTRAP\_SERVERS=broker:29092

    healthcheck:

      interval: 1s

      retries: 3

      start\_period: 60s

      test: nc -z schema-registry ${DATAHUB\_SCHEMA\_REGISTRY\_PORT:-8081}

      timeout: 5s

    hostname: schema-registry

    image: ${DATAHUB\_CONFLUENT\_SCHEMA\_REGISTRY\_IMAGE:-confluentinc/cp-schema-registry}:${DATAHUB\_CONFLUENT\_VERSION:-7.4.0}

    ports:

    - ${DATAHUB\_MAPPED\_SCHEMA\_REGISTRY\_PORT:-8081}:8081

  zookeeper:

    environment:

    - ZOOKEEPER\_CLIENT\_PORT=2181

    - ZOOKEEPER\_TICK\_TIME=2000

    healthcheck:

      interval: 5s

      retries: 3

      start\_period: 10s

      test: echo srvr | nc zookeeper $${DATAHUB\_ZK\_PORT:-2181}

      timeout: 5s

    hostname: zookeeper

    image: ${DATAHUB\_CONFLUENT\_ZOOKEEPER\_IMAGE:-confluentinc/cp-zookeeper}:${DATAHUB\_CONFLUENT\_VERSION:-7.4.0}

    ports:

    - ${DATAHUB\_MAPPED\_ZK\_PORT:-2181}:2181

    volumes:

    - zkdata:/var/lib/zookeeper/data

    - zklogs:/var/lib/zookeeper/log

  x-airflow-common: &airflow-common

    # Note: You can update the Airflow version as needed.

    image: apache/airflow:2.3.4

    environment:

      - AIRFLOW\_\_CORE\_\_EXECUTOR=LocalExecutor

      # Point to our dedicated Airflow metadata DB

      - AIRFLOW\_\_CORE\_\_SQL\_ALCHEMY\_CONN=postgresql+psycopg2://airflow:airflow@postgres\_airflow/airflow

      - AIRFLOW\_\_CELERY\_\_RESULT\_BACKEND=postgresql+psycopg2://airflow:airflow@postgres\_airflow/airflow

      - AIRFLOW\_\_CORE\_\_FERNET\_KEY=FB0o\_zt4e3Ziq3LdUUO7F2Z95cvFFx16hU8jTeR1ASM=

      - AIRFLOW\_\_CORE\_\_LOAD\_EXAMPLES=True

      - AIRFLOW\_\_CORE\_\_DAGS\_ARE\_PAUSED\_CREATION=True

      - AIRFLOW\_\_CORE\_\_LOGGING\_LEVEL=INFO

      - AIRFLOW\_\_SCHEDULER\_\_ENABLE\_HEALTH\_CHECK=True

    volumes:

      - ./dags:/opt/airflow/dags

      - ./airflow-data/logs:/opt/airflow/logs

      - ./airflow-data/plugins:/opt/airflow/plugins

      - ./airflow-data/airflow.cfg:/opt/airflow/airflow.cfg

    depends\_on:

      - postgres\_airflow

  airflow-init:

    <<: \*airflow-common

    container\_name: airflow\_init

    entrypoint: /bin/bash

    command:

      - -c

      - >

        airflow users list || ( airflow db init &&

        airflow users create --role Admin --username airflow --password airflow

        --email airflow@airflow.com --firstname airflow --lastname airflow )

    restart: on-failure

  airflow-webserver:

    <<: \*airflow-common

    container\_name: airflow\_webserver

    command: airflow webserver

    ports:

      - "8085:8085"

  airflow-scheduler:

    <<: \*airflow-common

    container\_name: airflow\_scheduler

    command: airflow scheduler

    restart: always

volumes:

  broker: null

  esdata: null

  mysqldata: null

  neo4jdata: null

  zkdata: null

  zklogs: null

  minio\_data: null

  postgres\_airflow\_data: null

  postgres\_dw\_data: null

After saving the file, launch the stack with:

docker-compose up -d

**2. Extraction & Loading**

**Extraction to OSS (MinIO)**

In dags/extract\_to\_minio.py we verify the file hash and upload the CSV only if it has changed.

#!/usr/bin/env python3

import boto3, hashlib

from botocore.exceptions import ClientError

MINIO\_ENDPOINT = "localhost:9000"

ACCESS\_KEY = "minioadmin"

SECRET\_KEY = "minioadmin"

BUCKET\_NAME = "raw-data"

FILE\_PATH = "data\_source/sample\_data.csv"

OBJECT\_NAME = "sample\_data.csv"

def get\_file\_md5(file\_path):

hash\_md5 = hashlib.md5()

with open(file\_path, "rb") as f:

for chunk in iter(lambda: f.read(4096), b""):

hash\_md5.update(chunk)

return hash\_md5.hexdigest()

def main():

s3 = boto3.client('s3',

endpoint\_url=f"http://{MINIO\_ENDPOINT}",

aws\_access\_key\_id=ACCESS\_KEY,

aws\_secret\_access\_key=SECRET\_KEY,

region\_name='us-east-1')

# Create bucket if it doesn't exist

try:

s3.head\_bucket(Bucket=BUCKET\_NAME)

except ClientError:

s3.create\_bucket(Bucket=BUCKET\_NAME)

file\_md5 = get\_file\_md5(FILE\_PATH)

try:

response = s3.head\_object(Bucket=BUCKET\_NAME, Key=OBJECT\_NAME)

etag = response['ETag'].strip('"')

except ClientError:

etag = None

if etag != file\_md5:

print("File new or changed. Uploading new version...")

s3.upload\_file(FILE\_PATH, BUCKET\_NAME, OBJECT\_NAME)

else:

print("No changes detected. Skipping upload.")

if \_\_name\_\_ == '\_\_main\_\_':

main()

**Loading from OSS into PostgreSQL**

In dags/load\_to\_postgres.py, we download the file from MinIO and load it into a raw table in PostgreSQL using Pandas:

#!/usr/bin/env python3

import boto3

import pandas as pd

from io import BytesIO

from sqlalchemy import create\_engine

MINIO\_ENDPOINT = "localhost:9000"

ACCESS\_KEY = "minioadmin"

SECRET\_KEY = "minioadmin"

BUCKET\_NAME = "raw-data"

OBJECT\_NAME = "sample\_data.csv"

POSTGRES\_CONN = "postgresql+psycopg2://dwh\_user:dwh\_password@localhost:5432/datamart"

def download\_file\_from\_minio():

s3 = boto3.client('s3',

endpoint\_url=f"http://{MINIO\_ENDPOINT}",

aws\_access\_key\_id=ACCESS\_KEY,

aws\_secret\_access\_key=SECRET\_KEY,

region\_name='us-east-1')

response = s3.get\_object(Bucket=BUCKET\_NAME, Key=OBJECT\_NAME)

return response['Body'].read()

def load\_to\_postgres(csv\_data):

df = pd.read\_csv(BytesIO(csv\_data))

engine = create\_engine(POSTGRES\_CONN)

df.to\_sql("raw\_data", engine, if\_exists="replace", index=False)

print("Data loaded successfully into PostgreSQL.")

def main():

csv\_data = download\_file\_from\_minio()

load\_to\_postgres(csv\_data)

if \_\_name\_\_ == '\_\_main\_\_':

main()

**3. Transformation with DBT**

Under the dags/dbt\_project directory:

**dbt\_project.yml**

name: 'data\_pipeline'

version: '1.0'

config-version: 2

profile: 'data\_pipeline'

source-paths: ["models"]

target-path: "target"

**profiles.yml (Place in ~/.dbt/profiles.yml or mount into DBT container)**

data\_pipeline:

target: dev

outputs:

dev:

type: postgres

host: postgres\_dw

user: dwh\_user

password: dwh\_password

port: 5432

dbname: datamart

schema: public

**Model: models/raw\_to\_normalized.sql**

Transform the raw data into a normalized format. Adjust column names and logic according to your CSV.

with source\_data as (

select \*

from {{ source('raw', 'raw\_data') }}

)

select

column1,

column2,

upper(column3) as column3\_upper

from source\_data

**Schema & Tests: models/schema.yml**

Include documentation and tests for quality:

version: 2

models:

- name: raw\_to\_normalized

description: "Normalized version of raw uploaded data."

columns:

- name: column1

description: "First example column"

tests:

- not\_null

- name: column2

description: "Second example column"

tests:

- unique

- name: column3\_upper

description: "Uppercase version of column3"

tests:

- accepted\_values:

values: ["VALUE1", "VALUE2"]

Run the following commands in the DBT container (or via your CI/CD process):

dbt run

dbt test

dbt docs generate

**4. Added Stage: Data Quality Validation using Great Expectations**

Create a new file dags/data\_quality\_check.py to perform quality checks on the raw data (or even on the transformed output):

#!/usr/bin/env python3

import sys

import pandas as pd

from sqlalchemy import create\_engine

import great\_expectations as ge

# Connect to Postgres where raw data was loaded

DATABASE\_URI = "postgresql+psycopg2://dwh\_user:dwh\_password@localhost:5432/datamart"

engine = create\_engine(DATABASE\_URI)

def run\_data\_quality\_checks():

# Read raw data table

df = pd.read\_sql("SELECT \* FROM raw\_data", engine)

ge\_df = ge.from\_pandas(df)

# Define expectations, e.g., column1 should not be null and column2 should have unique values.

expectations = [

ge\_df.expect\_column\_values\_to\_not\_be\_null("column1"),

ge\_df.expect\_column\_values\_to\_be\_unique("column2")

]

# Check overall success

for expectation in expectations:

if not expectation.get("success", False):

print("Data quality check failed:", expectation)

sys.exit(1)

print("Data quality checks passed.")

sys.exit(0)

if \_\_name\_\_ == '\_\_main\_\_':

run\_data\_quality\_checks()

This script reads the raw data, applies a few sample expectations, and exits with a non‑zero status if any check fails. In a real scenario, you’d add more detailed and domain‐specific expectations.

**5. Added Stage: Data Enrichment & KPI Aggregation**

Create a new file dags/data\_enrichment.py that reads the normalized data and computes aggregated KPIs. For example, suppose you want to compute the sum of a “sales” column per “category.” (Adjust the SQL and aggregation logic as needed.)

#!/usr/bin/env python3

import pandas as pd

from sqlalchemy import create\_engine

DATABASE\_URI = "postgresql+psycopg2://dwh\_user:dwh\_password@localhost:5432/datamart"

def enrich\_data():

engine = create\_engine(DATABASE\_URI)

# Read the transformed (normalized) data from DBT output; adjust table name as needed.

df = pd.read\_sql("SELECT \* FROM raw\_to\_normalized", engine)

# Example enrichment: aggregate a metric by category.

if "category" in df.columns and "sales" in df.columns:

kpi = df.groupby("category").agg(total\_sales=("sales", "sum")).reset\_index()

else:

# Fallback: just count the number of rows per unique column1 value.

kpi = df.groupby("column1").size().reset\_index(name="row\_count")

# Write the aggregated KPIs to a new table in the data warehouse

kpi.to\_sql("kpi\_metrics", engine, if\_exists="replace", index=False)

print("Data enrichment complete. KPI metrics stored in 'kpi\_metrics'.")

if \_\_name\_\_ == '\_\_main\_\_':

enrich\_data()

In practice, you may perform more sophisticated enrichment (for example, joining multiple datasets, doing time-series aggregations, or even leveraging machine learning models to derive insights).

**6. Updated Airflow DAG**

Finally, update your Airflow DAG (dags/data\_pipeline\_dag.py) to chain all stages together. The DAG now includes tasks for extraction, loading, DBT transformation, quality validation, enrichment, and (optionally) notification. For notification, we include a Slack notification example:

from airflow import DAG

from airflow.operators.bash import BashOperator

from airflow.providers.slack.operators.slack\_webhook import SlackWebhookOperator

from airflow.utils.dates import days\_ago

from datetime import timedelta

default\_args = {

'owner': 'airflow',

'start\_date': days\_ago(1),

'retry\_delay': timedelta(minutes=5),

}

with DAG('data\_pipeline',

default\_args=default\_args,

schedule\_interval='@daily',

catchup=False) as dag:

extract\_task = BashOperator(

task\_id='extract\_data',

bash\_command='python /opt/airflow/dags/extract\_to\_minio.py'

)

load\_task = BashOperator(

task\_id='load\_data',

bash\_command='python /opt/airflow/dags/load\_to\_postgres.py'

)

dbt\_run\_task = BashOperator(

task\_id='dbt\_run',

bash\_command='dbt run --project-dir /opt/airflow/dags/dbt\_project'

)

quality\_check\_task = BashOperator(

task\_id='quality\_check',

bash\_command='python /opt/airflow/dags/data\_quality\_check.py'

)

enrichment\_task = BashOperator(

task\_id='enrich\_data',

bash\_command='python /opt/airflow/dags/data\_enrichment.py'

)

notify\_task = SlackWebhookOperator(

task\_id='notify\_success',

http\_conn\_id='slack\_connection', # Configure this connection in the Airflow UI if needed

message="Data pipeline run completed successfully and enriched KPIs are available!",

channel="#data-team"

)

extract\_task >> load\_task >> dbt\_run\_task >> quality\_check\_task >> enrichment\_task >> notify\_task

I have placed the supporting Python scripts as well (extract\_to\_minio.py, load\_to\_postgres.py, data\_quality\_check.py, and data\_enrichment.py) in my dags folder.

**7. Metadata and Dashboarding**

You can continue to run your DBT metadata ingestion (using the datahub\_ingestion.yml file described earlier) and then access your DataHub UI at <http://localhost:9002> to view lineage and model details. Superset will be available at <http://localhost:8088> for dashboard creation over both the normalized data and the aggregated 'kpi\_metrics' table.

**Final Notes**

* **Data Quality & Monitoring:**  
  The quality check stage can be extended with more specialized expectations and integrated into a monitoring dashboard (for example, using Airflow’s logging or a dedicated monitoring tool).
* **Data Enrichment:**  
  Additional enrichment could include real‑time streaming aggregations (using Kafka with a Spark Structured Streaming job), auto‑detection of schema changes, or even ML‑powered anomaly detection.
* **Advanced Orchestration:**  
  Consider modularizing the pipeline into separate DAGs (e.g., one for ingestion, one for transformation and validation, and one for enrichment and notification) for larger projects.
* **Documentation & Reporting:**  
  Automated documentation (using dbt docs) and metadata ingestion to DataHub help maintain a single source of truth for your data lineage, which is critical in real‑world data engineering.

This enriched solution provides a robust, modern pipeline with multiple stages that you can customize and expand further as you build out your final project. Happy coding, and may fortune favor your innovation in data engineering!