HOWTO: Set up and Use the Advanced Data Platform Files

This guide provides step-by-step instructions on how to set up, run, and interact with the advanced data platform using the provided files.

1. Project Structure

First, create the following directory structure in your project root. This structure is critical for Docker Compose to find all the necessary files and mount volumes correctly.

```
.
├── airflow_dags/
├── data/
│   ├── airflow_logs/
│   ├── grafana/
│   ├── openmetadata_elasticsearch/
│   ├── openmetadata_mysql/
│   ├── spark-events/
│   ├── starter-minio/
│   ├── starter-mongodb/
│   ├── starter-postgres/
│   └── spline_jars/  <-- You will place Spline JARs here
├── fastapi_app/
│   └── app/
├── observability/
├── openmetadata_ingestion_scripts/
├── pyspark_jobs/
├── webhook_listener_app/
│   ├── Dockerfile
│   └── requirements.txt
├── .env  <-- Optional: for environment variables (not used in current setup but good practice)
└── docker-compose.yml
└── simulate_data.py
```

2. Populate Files

Copy the content from the immersives I provided into their respective files within this structure.

- docker-compose.yml (from docker-compose-updated-final)
- simulate_data.py (from simulate-data-script-updated)
- fastapi_app/requirements.txt (from fastapi-app-updated)

- fastapi_app/app/main.py (from fastapi-app-updated)
- pyspark_jobs/streaming_consumer.py (from pyspark-streaming-consumer-updated)
- pyspark_jobs/batch_transformations.py (from pyspark-batch-transformations-updated)
- pyspark_jobs/delta_merge_cdc.py (from pyspark-delta-merge-cdc-updated)
- pyspark_jobs/ml_model_inference.py (from pyspark-ml-model-inference-updated)
- pyspark_jobs/mongo_processor.py (from pyspark-mongo-processor-updated)
- observability/alloy-config.river (from grafana-alloy-config-updated)
- airflow_dags/openmetadata_ingestion_dag.py (from airflow-dag-openmetadata-ingestion-updated)
- airflow_dags/full_pipeline_with_governance_dag.py (from airflow-dag-full-pipeline-updated)
- openmetadata_ingestion_scripts/kafka_connector_config.yaml (from openmetadata-kafka-config-updated)
- openmetadata_ingestion_scripts/s3_delta_connector_config.yaml (from openmetadata-s3-delta-config-updated)
- openmetadata_ingestion_scripts/postgres_connector_config.yaml (from openmetadata-postgres-config-updated)
- openmetadata_ingestion_scripts/spline_connector_config.yaml (from openmetadata-spline-config-updated)
- openmetadata_ingestion_scripts/fastapi_connector_config.yaml (from openmetadata-fastapi-config-updated)
- openmetadata_ingestion_scripts/mongodb_connector_config.yaml (from openmetadata-mongodb-config-updated)
- openmetadata_ingestion_scripts/spark_lineage_and_s3_delta_connector_config.yaml (from openmetadata-spark-lineage-s3-delta-config-updated)
- webhook_listener_app/app.py (from minio-webhook-listener-updated)
- webhook_listener_app/Dockerfile (from minio-webhook-listener-dockerfile-updated)
- webhook_listener_app/requirements.txt (from minio-webhook-listener-requirements-updated)

3. Download External Dependencies

Spline JARs:
The Spark containers require Spline agent JARs for lineage tracking.

1. Go to the Spline GitHub releases page (e.g., https://github.com/AbsaOSS/spline/releases).
2. Download the **spline-spark-agent-bundle_2.12-0.7.1.jar** and **spline-agent-bundle-0.7.1.jar** files (ensure the version matches 0.7.1 as used in docker-compose.yml).
3. Place these two JAR files inside the data/spline_jars/ directory you created.

4. Build Docker Images

Navigate to your project root directory (where docker-compose.yml is located) in your terminal and run:

docker compose build

This command will build the Docker images for fastapi_ingestor and webhook_listener services based on their respective Dockerfiles and requirements.txt.

5. Start the Data Platform Services

Once the images are built, start all the services defined in docker-compose.yml:
docker compose up --build -d

- --build: Ensures your local images are rebuilt if there are changes.
- -d: Runs the containers in detached mode (in the background).

It might take several minutes for all services to become healthy, especially OpenMetadata and Airflow due to their database initialization and migrations. You can monitor their status using docker compose ps and docker compose logs <service_name>.

6. Initial Data Setup (MongoDB)

For the mongo_processor.py Spark job to work, you need to insert some initial data into MongoDB.

1. Wait for the mongodb service to be healthy (docker compose ps should show healthy).
2. Connect to the MongoDB shell inside the container:
   docker exec -it starter-mongodb mongosh --authenticationDatabase admin -u root -p password

3. Inside the mongosh prompt, insert some sample data for financial_events and/or claims as described in "Highlighting MongoDB: Flexible NoSQL Document Database" (Basic Use Case and Advanced Use Case 1).
   Example for financial_events:

```
use my_data_platform_db
db.financial_events.insertOne({
   "event_id": "EVT-001",
   "type": "login_attempt",
   "user_id": "USR-XYZ",
   "timestamp": ISODate("2024-06-14T10:30:00Z"),
   "ip_address": "192.168.1.100",
   "status": "success"
})
db.financial_events.insertOne({
   "event_id": "EVT-002",
   "type": "failed_login",
   "user_id": "USR-ABC",
   "timestamp": ISODate("2024-06-14T10:31:00Z"),
   "ip_address": "192.168.1.101",
   "reason": "incorrect_password",
```

```
    "attempts": 3
})
db.financial_events.find().pretty()
```

Exit mongosh by typing exit.

7. Configure MinIO Event Notifications (For Webhook Listener)

This step must be performed *after* the minio service is up and healthy.

1. Access the minio container's bash shell:
   docker exec -it starter-minio bash

2. Inside the MinIO container, run these mc commands:
   mc alias set local http://localhost:9000 minioadmin minioadmin
   mc mb local/my-event-bucket --ignore-existing # Create a bucket for events
   mc event add local/my-event-bucket arn:minio:sqs::1:webhook --suffix .parquet --event put # Add put object event
   mc admin config set notify_webhook:webhook_target endpoint='http://webhook_listener:8081/minio-event' queue_limit=100 --console-address ":9001"
   exit

3. Restart the MinIO service for event configurations to take effect:
   docker compose restart minio

8. Access User Interfaces

Once all services are up, you can access the various UIs:

- **FastAPI Swagger UI:** http://localhost:8000/docs
- **MinIO Console:** http://localhost:9001 (Login: minioadmin/minioadmin)
- **Kafka (Conceptual):** No direct UI, but accessible via applications.
- **Spark Master UI:** http://localhost:8088
- **Spark History Server:** http://localhost:18080
- **Airflow Webserver UI:** http://localhost:8081 (Login: admin/admin)
- **Grafana UI:** http://localhost:3000 (Login: admin/admin)
- **cAdvisor UI/Metrics:** http://localhost:8082 (provides raw metrics, Grafana will visualize them)
- **Spline UI:** http://localhost:9090
- **OpenMetadata UI:** http://localhost:8585 (Initial login: admin/admin if ENABLE_AUTH is false in docker-compose.yml)

9. Run Data Pipeline and Observe

Follow these steps to generate data and observe the platform's behavior:

1. Generate Data Traffic:
   Open a new terminal window in your project root and run the simulate_data.py script:
   python3 simulate_data.py

   This script will continuously send financial transactions and insurance claims to your FastAPI ingestor. You should see logs in your fastapi_ingestor container indicating successful ingestion and Kafka messages being sent.
2. Run Spark Streaming Consumer:
   Open another terminal window. This will start the Spark Structured Streaming job that consumes from Kafka and writes to MinIO.
   docker exec -it advanced-spark-master spark-submit \
     --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.0,io.delta:delta-core_2