Project Setup

This project consists of multiple services running via Docker and Docker Compose. Follow the steps below to start the containers. The aim is to use Kafka to ingest data from different sensors and then use airflow to process the data and store it in a database. The backend service generates reports and analytics based on data processed through the Airflow pipelines.

Prerequisites

- Ensure **Docker** and **Docker Compose** are installed.
- Tested on **Windows 11 with WSL2, Ubuntu 24.04, Docker Desktop and using Git Bash**.

Starting the Containers

1. Infrastructure Services

Navigate to the `infra` directory and start the required services:

```
cd infra
docker compose up -d
```

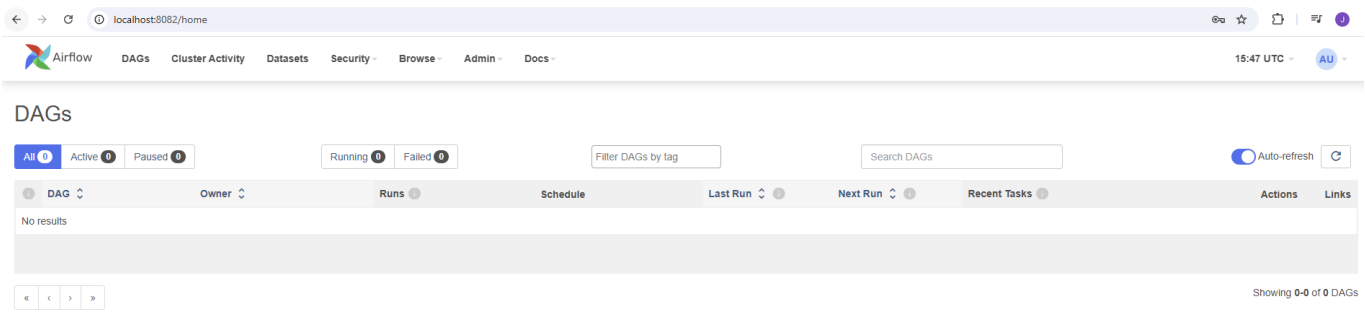
This will start **MySQL, Kafka, Zookeeper, Airflow**, and their respective UIs.

Create an Airflow user:

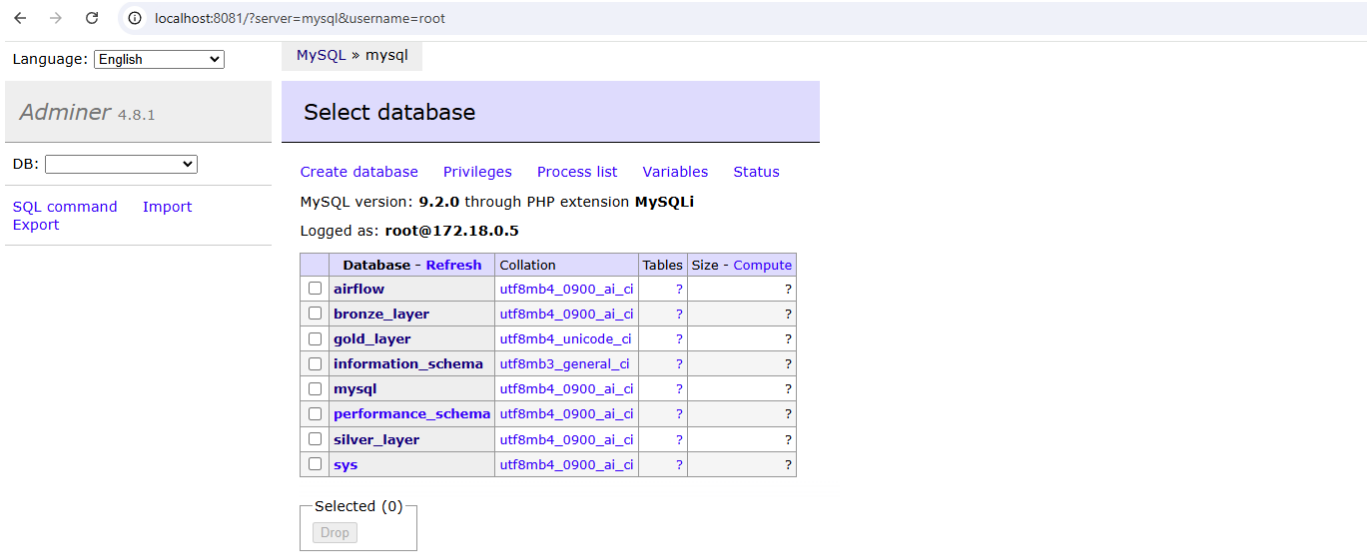
'docker exec -it airflow airflow users create -e admin@gmail.com -f Admin -l User -p secret -r Admin -u root'

Credentials & UI Access:

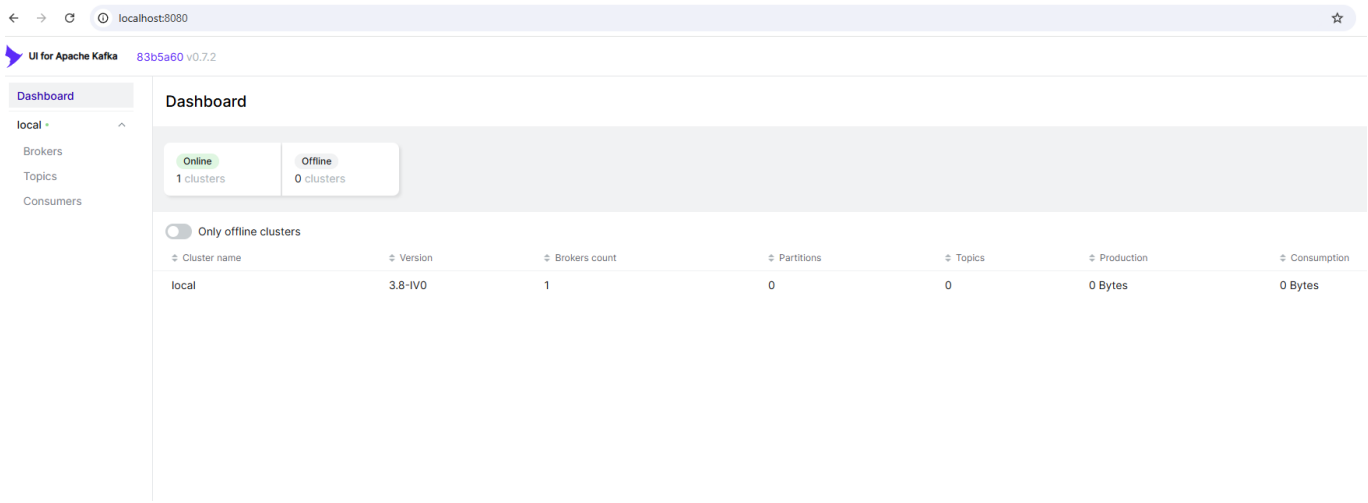
- **Airflow**: http://localhost:8082
  - User: `root` | Password: `secret`



- **MySQL**: http://localhost:8081
  - User: `root` | Password: `secret`



- **Kafka UI**: http://localhost:8080

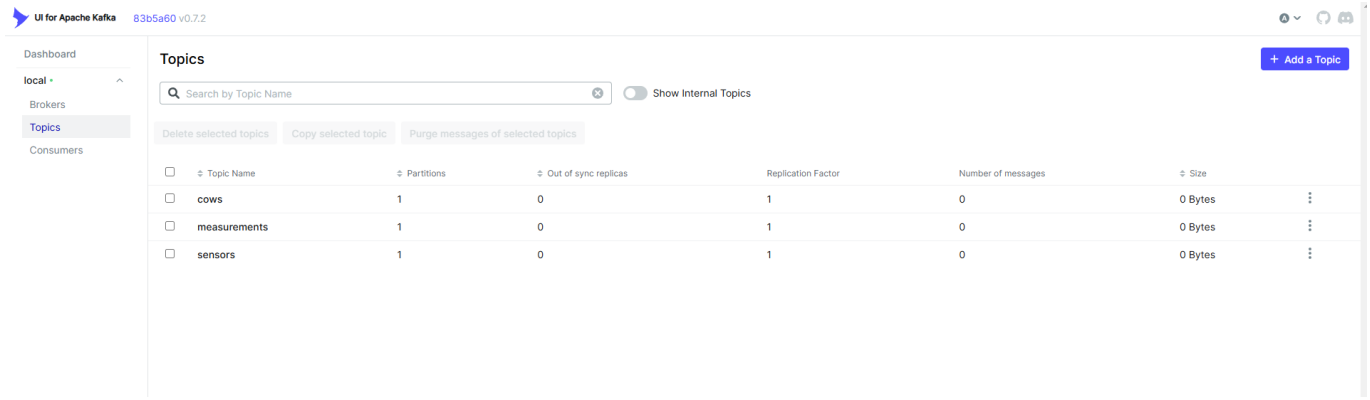


2. Producer Service

Once the infrastructure is running, start the **Producer Service**:

```
cd ../producer
docker compose up -d
```

This service initializes Kafka topics and allows data ingestion.

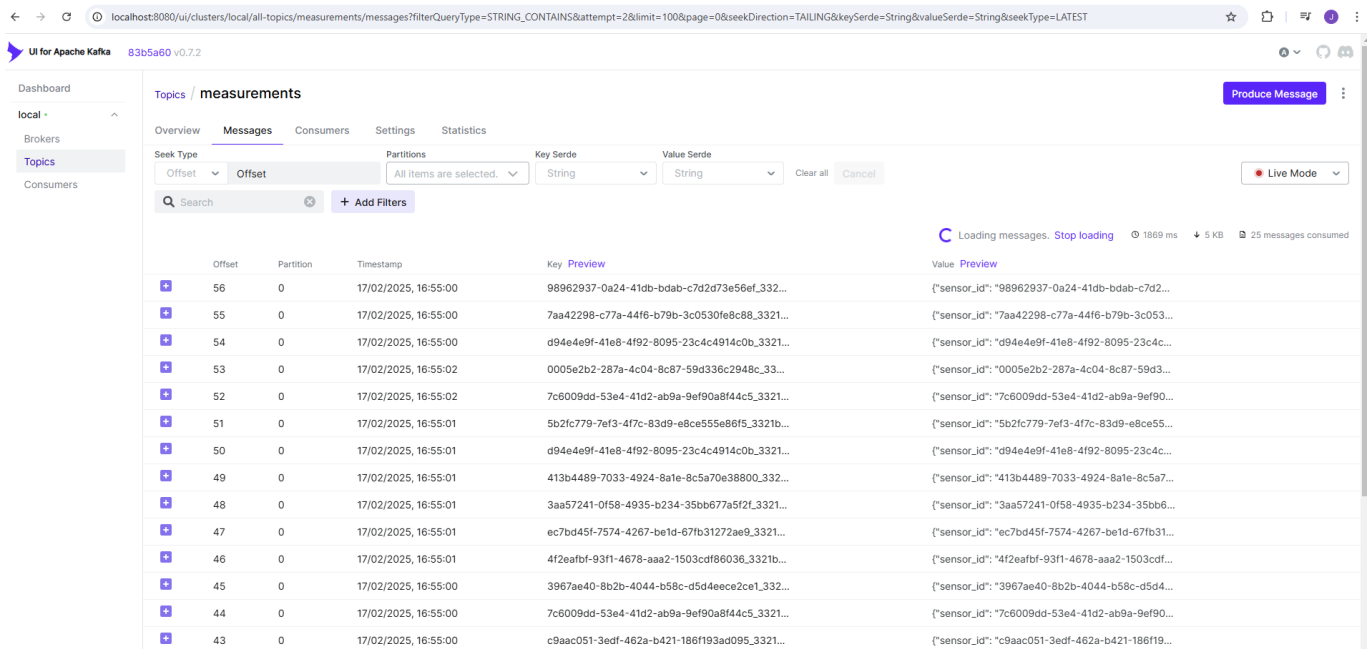


Populate Kafka Topics:

```
./producer_seeders.sh
```

This script reads **parquet files** from the `data` directory and sends them to Kafka topics:

- `data/cows`
- `data/measurements`
- `data/sensors`



*__Notice:__ Measurements take a while to become available in Kafka topics. Nevertheless, the remaining steps can be executed.

Endpoints for ingestion:

- `POST http://localhost:5001/cow`
- `POST http://localhost:5001/ingrid/measurement`
- `POST http://localhost:5001/ingrid/sensor`

Check API documentation at http://localhost:5001/docs.

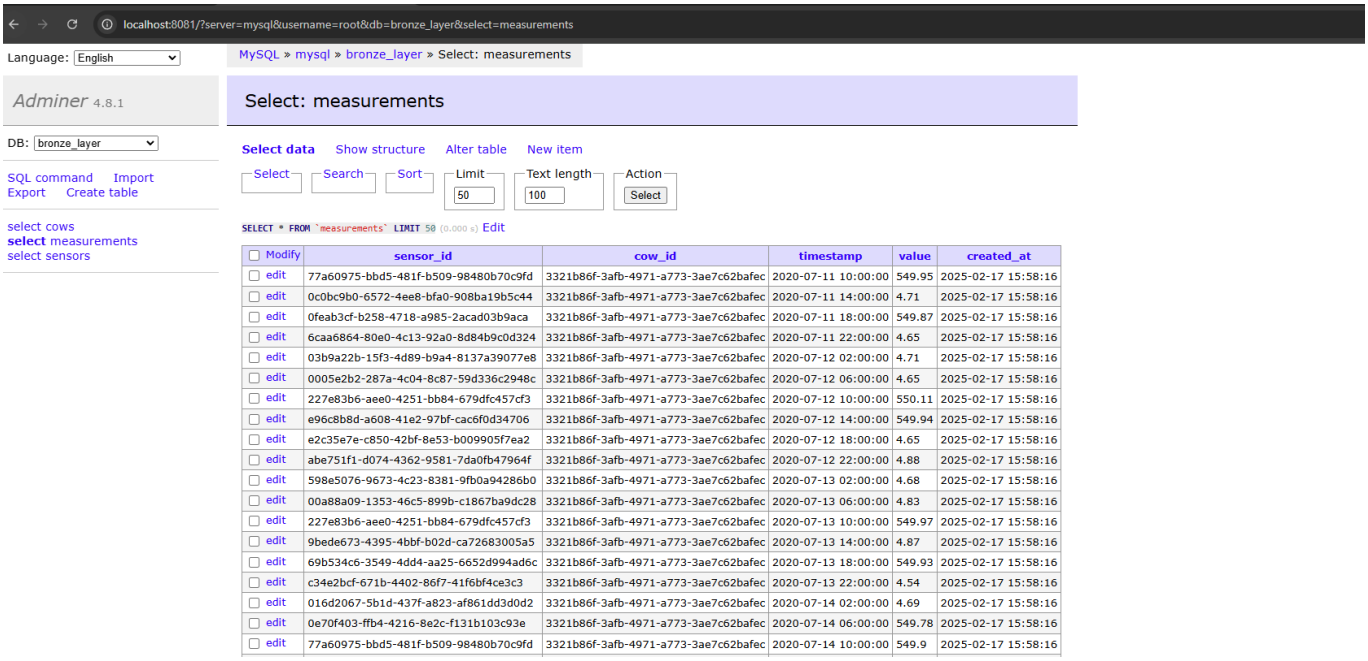
3. Pipeline Service

The **Pipeline Service** processes Kafka data and organizes it into layers:

- **Kafka Consumer**: Reads Kafka data and stores it in the raw layer.
- **Airflow DAGs**: Process data into silver and gold layers.

Load Data into Raw Layer:

```
./consumer_seeder.sh
```

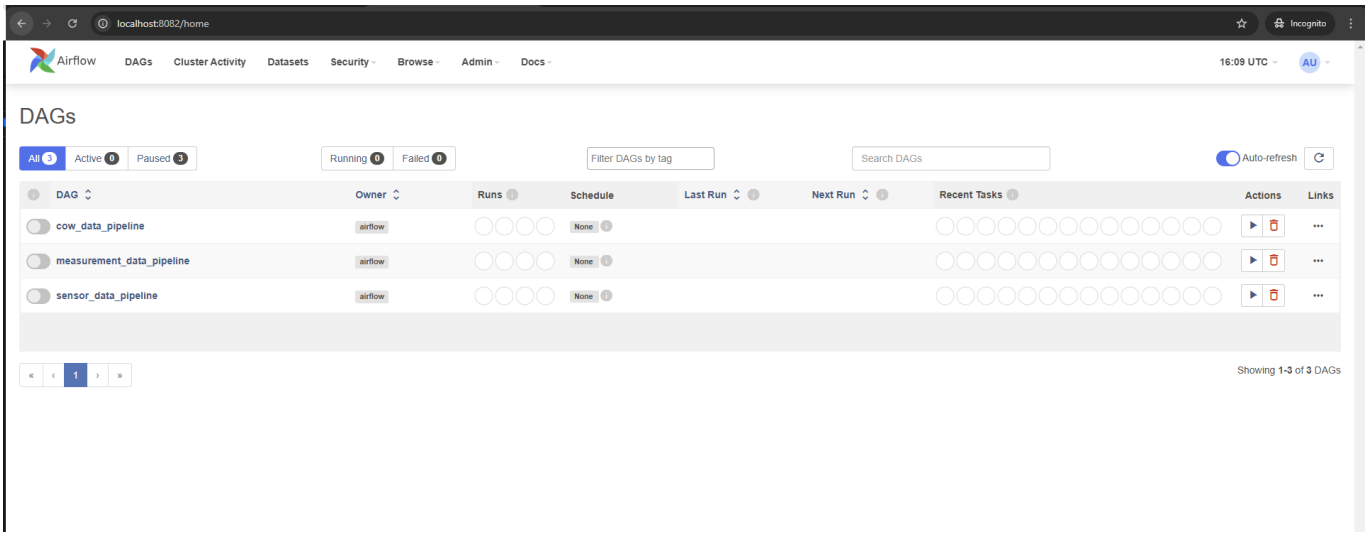


Modify	sensor_id	cow_id	timestamp	value	created_at
edit	77a60975-bbd5-481f-b509-98480b70c9fd	3321b86f-3afb-4971-a773-3ae7c62bafec	2020-07-11 10:00:00	549.95	2025-02-17 15:58:16
edit	0c0bc9b0-6572-4ee8-bfa0-908ba19b5c44	3321b86f-3afb-4971-a773-3ae7c62bafec	2020-07-11 14:00:00	4.71	2025-02-17 15:58:16
edit	0feab3cf-b258-4718-a985-2acad03b9aca	3321b86f-3afb-4971-a773-3ae7c62bafec	2020-07-11 18:00:00	549.87	2025-02-17 15:58:16
edit	6caa6864-80e0-4c13-92a0-8d84b9c0d324	3321b86f-3afb-4971-a773-3ae7c62bafec	2020-07-11 22:00:00	4.65	2025-02-17 15:58:16
edit	03b9a22b-15f3-4d89-b9a4-8137a39077e8	3321b86f-3afb-4971-a773-3ae7c62bafec	2020-07-12 02:00:00	4.71	2025-02-17 15:58:16
edit	0005e2b2-287a-4c04-8c87-59d336c2948c	3321b86f-3afb-4971-a773-3ae7c62bafec	2020-07-12 06:00:00	4.65	2025-02-17 15:58:16
edit	227e83b6-aee0-4251-bb84-679dfc457cf3	3321b86f-3afb-4971-a773-3ae7c62bafec	2020-07-12 10:00:00	550.11	2025-02-17 15:58:16
edit	e96c8b8d-a608-41e2-97bf-cac6f0d34706	3321b86f-3afb-4971-a773-3ae7c62bafec	2020-07-12 14:00:00	549.94	2025-02-17 15:58:16
edit	e2c35e7e-c850-42bf-8e53-b009905f7ea2	3321b86f-3afb-4971-a773-3ae7c62bafec	2020-07-12 18:00:00	4.65	2025-02-17 15:58:16
edit	abe751f1-d074-4362-9581-7da0fb47964f	3321b86f-3afb-4971-a773-3ae7c62bafec	2020-07-12 22:00:00	4.88	2025-02-17 15:58:16
edit	598e5076-9673-4c23-8381-9fb0a94286b0	3321b86f-3afb-4971-a773-3ae7c62bafec	2020-07-13 02:00:00	4.68	2025-02-17 15:58:16
edit	00a88a09-1353-46c5-899b-c1867ba9dc28	3321b86f-3afb-4971-a773-3ae7c62bafec	2020-07-13 06:00:00	4.83	2025-02-17 15:58:16
edit	227e83b6-aee0-4251-bb84-679dfc457cf3	3321b86f-3afb-4971-a773-3ae7c62bafec	2020-07-13 10:00:00	549.97	2025-02-17 15:58:16
edit	9bede673-4395-4bbf-b02d-ca72683005a5	3321b86f-3afb-4971-a773-3ae7c62bafec	2020-07-13 14:00:00	4.87	2025-02-17 15:58:16
edit	69b534c6-3549-4dd4-aa25-6652d994ad6c	3321b86f-3afb-4971-a773-3ae7c62bafec	2020-07-13 18:00:00	549.93	2025-02-17 15:58:16
edit	c34e2bcf-671b-4402-86f7-41f6bf4ce3c3	3321b86f-3afb-4971-a773-3ae7c62bafec	2020-07-13 22:00:00	4.54	2025-02-17 15:58:16
edit	016d2067-5b1d-437f-a823-af861dd3d0d2	3321b86f-3afb-4971-a773-3ae7c62bafec	2020-07-14 02:00:00	4.69	2025-02-17 15:58:16
edit	0e70f403-ffb4-4216-8e2c-f131b103c93e	3321b86f-3afb-4971-a773-3ae7c62bafec	2020-07-14 06:00:00	549.78	2025-02-17 15:58:16
edit	77a60975-bbd5-481f-b509-98480b70c9fd	3321b86f-3afb-4971-a773-3ae7c62bafec	2020-07-14 10:00:00	549.9	2025-02-17 15:58:16

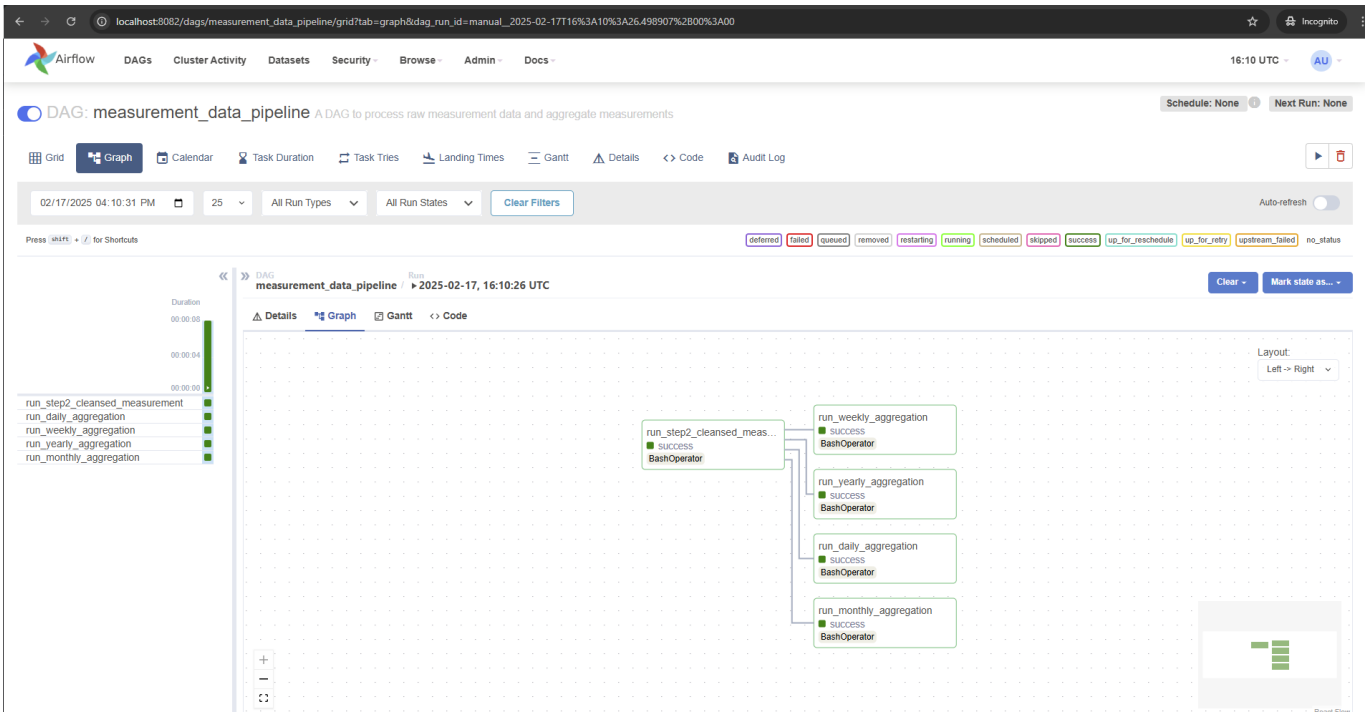
Deploy Pipeline Scripts to Airflow:

```
./update_dags.sh
```

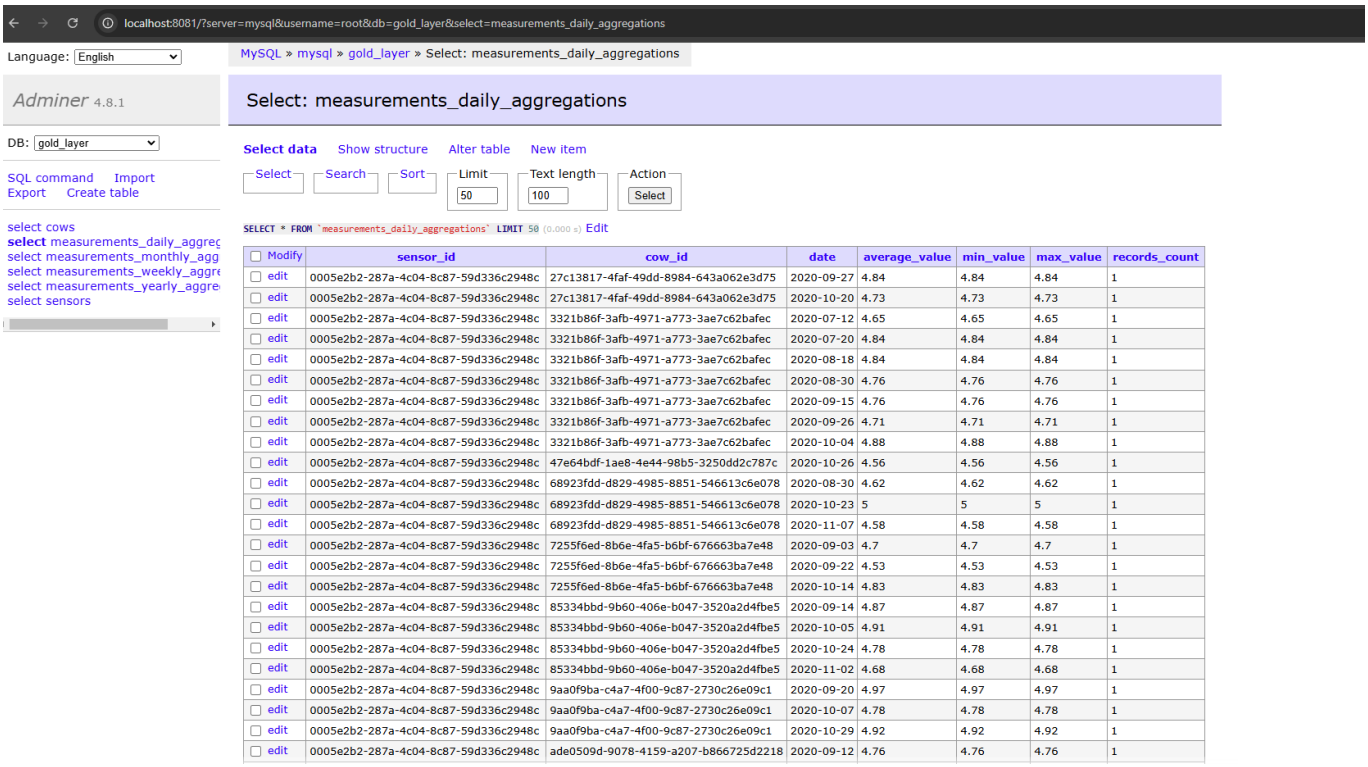
Notice: The new DAGs take a while to be available in the Airflow UI.



Run DAGs via **Airflow UI**: http://localhost:8082



After running the 3 DAGs, the data will be available in **MySQL** in the **gold** layer. The gold layer is used by the **Backend Service** to generate the reports.



Modify	sensor_id	cow_id	date	average_value	min_value	max_value	records_count
edit	0005e2b2-287a-4c04-8c87-59d336c2948c	27c13817-4faf-49dd-8984-643a062e3d75	2020-09-27	4.84	4.84	4.84	1
edit	0005e2b2-287a-4c04-8c87-59d336c2948c	27c13817-4faf-49dd-8984-643a062e3d75	2020-10-20	4.73	4.73	4.73	1
edit	0005e2b2-287a-4c04-8c87-59d336c2948c	3321b86f-3afb-4971-a773-3ae7c62bafec	2020-07-12	4.65	4.65	4.65	1
edit	0005e2b2-287a-4c04-8c87-59d336c2948c	3321b86f-3afb-4971-a773-3ae7c62bafec	2020-07-20	4.84	4.84	4.84	1
edit	0005e2b2-287a-4c04-8c87-59d336c2948c	3321b86f-3afb-4971-a773-3ae7c62bafec	2020-08-18	4.84	4.84	4.84	1
edit	0005e2b2-287a-4c04-8c87-59d336c2948c	3321b86f-3afb-4971-a773-3ae7c62bafec	2020-08-30	4.76	4.76	4.76	1
edit	0005e2b2-287a-4c04-8c87-59d336c2948c	3321b86f-3afb-4971-a773-3ae7c62bafec	2020-09-15	4.76	4.76	4.76	1
edit	0005e2b2-287a-4c04-8c87-59d336c2948c	3321b86f-3afb-4971-a773-3ae7c62bafec	2020-09-26	4.71	4.71	4.71	1
edit	0005e2b2-287a-4c04-8c87-59d336c2948c	3321b86f-3afb-4971-a773-3ae7c62bafec	2020-10-04	4.88	4.88	4.88	1
edit	0005e2b2-287a-4c04-8c87-59d336c2948c	47e64bdf-1ae8-4e44-98b5-3250dd2c787c	2020-10-26	4.56	4.56	4.56	1
edit	0005e2b2-287a-4c04-8c87-59d336c2948c	68923fdd-d829-4985-8851-546613c6e078	2020-08-30	4.62	4.62	4.62	1
edit	0005e2b2-287a-4c04-8c87-59d336c2948c	68923fdd-d829-4985-8851-546613c6e078	2020-10-23	5	5	5	1
edit	0005e2b2-287a-4c04-8c87-59d336c2948c	68923fdd-d829-4985-8851-546613c6e078	2020-11-07	4.58	4.58	4.58	1
edit	0005e2b2-287a-4c04-8c87-59d336c2948c	7255f6ed-8b6e-4fa5-b6bf-676663ba7e48	2020-09-03	4.7	4.7	4.7	1
edit	0005e2b2-287a-4c04-8c87-59d336c2948c	7255f6ed-8b6e-4fa5-b6bf-676663ba7e48	2020-09-22	4.53	4.53	4.53	1
edit	0005e2b2-287a-4c04-8c87-59d336c2948c	7255f6ed-8b6e-4fa5-b6bf-676663ba7e48	2020-10-14	4.83	4.83	4.83	1
edit	0005e2b2-287a-4c04-8c87-59d336c2948c	85334bbd-9b60-406e-b047-3520a2d4fbe5	2020-09-14	4.87	4.87	4.87	1
edit	0005e2b2-287a-4c04-8c87-59d336c2948c	85334bbd-9b60-406e-b047-3520a2d4fbe5	2020-10-05	4.91	4.91	4.91	1
edit	0005e2b2-287a-4c04-8c87-59d336c2948c	85334bbd-9b60-406e-b047-3520a2d4fbe5	2020-10-24	4.78	4.78	4.78	1
edit	0005e2b2-287a-4c04-8c87-59d336c2948c	85334bbd-9b60-406e-b047-3520a2d4fbe5	2020-11-02	4.68	4.68	4.68	1
edit	0005e2b2-287a-4c04-8c87-59d336c2948c	9aa0f9ba-c4a7-4f00-9c87-2730c26e09c1	2020-09-20	4.97	4.97	4.97	1
edit	0005e2b2-287a-4c04-8c87-59d336c2948c	9aa0f9ba-c4a7-4f00-9c87-2730c26e09c1	2020-10-07	4.78	4.78	4.78	1
edit	0005e2b2-287a-4c04-8c87-59d336c2948c	9aa0f9ba-c4a7-4f00-9c87-2730c26e09c1	2020-10-29	4.92	4.92	4.92	1
edit	0005e2b2-287a-4c04-8c87-59d336c2948c	ade0509d-9078-4159-a207-b866725d2218	2020-09-12	4.76	4.76	4.76	1

4. Backend Service

Finally, start the **Backend Service**:

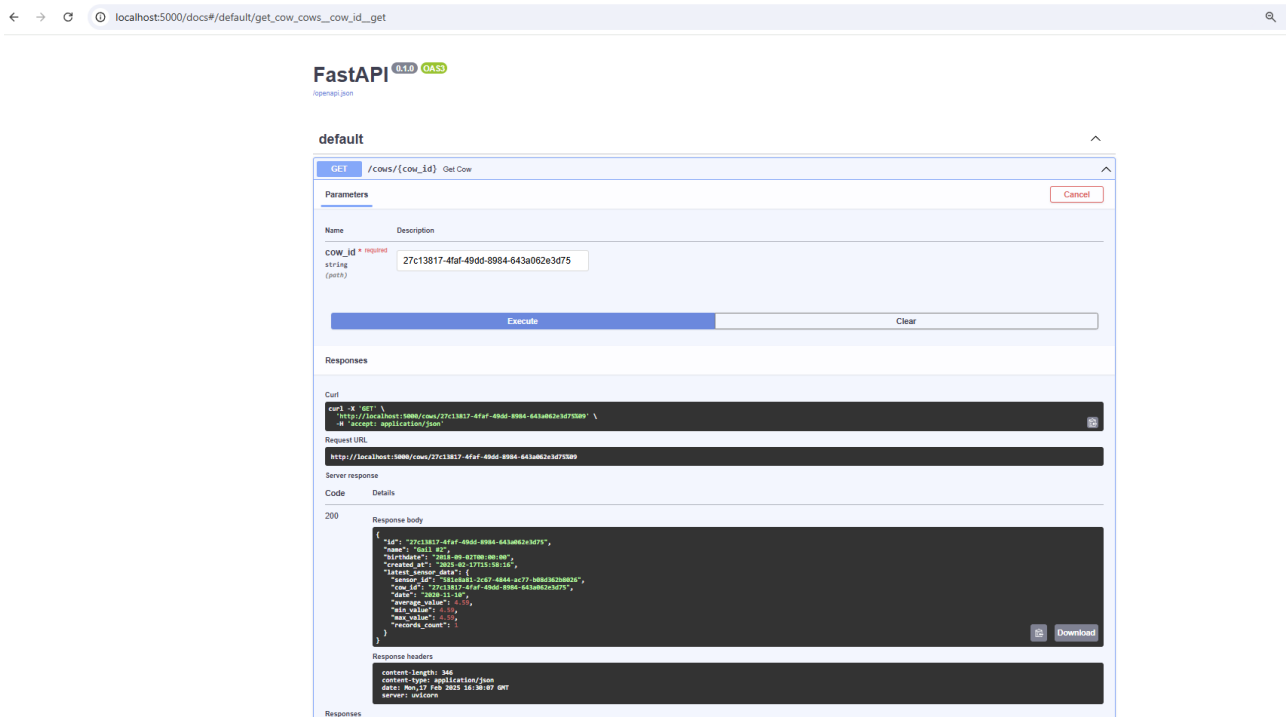
```
cd ../back-end
docker compose up -d
```

API Access:

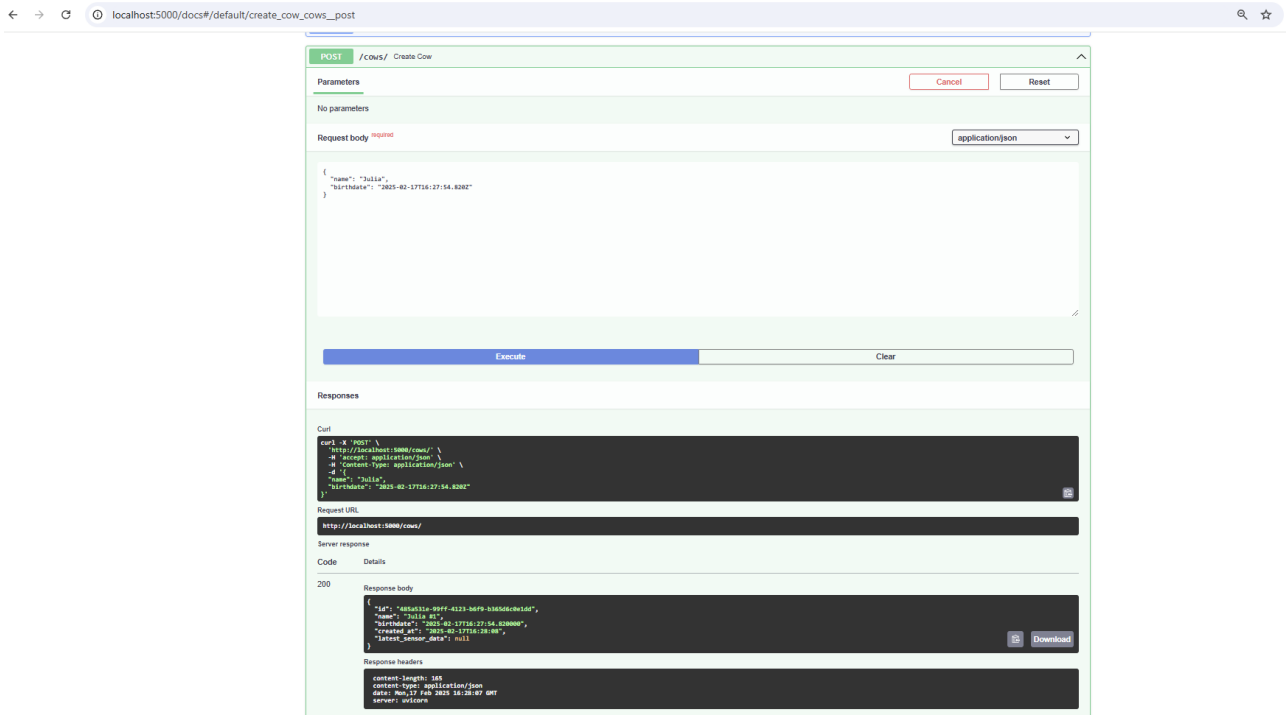
- Base URL: http://localhost:5000/
- API Docs: http://localhost:5000/docs

Key Endpoints:

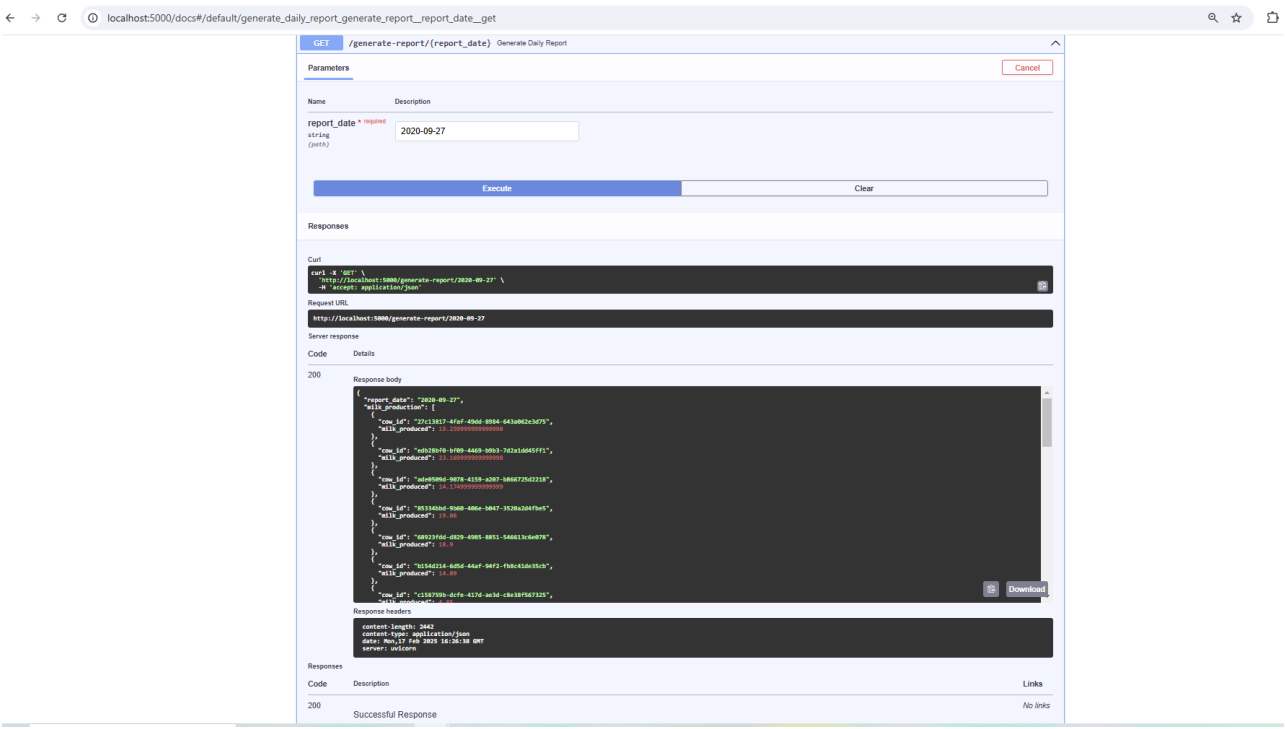
- `GET /cows/{cow_id}` - Retrieve cow data with latest sensor info.



- `POST /cows/` - Create a new cow.



- `GET /generate-report/{report_date}` - Generate a report for the given date.



There is a shell script to run the tests:

```
./run_tests.sh
```

This will run the tests from the `back-end` service.

Stopping the Containers

To stop all services, run in each directory:

```
docker compose down
```

To remove all data and images in each directory:

```
docker compose down -v --rmi all
```

TODOs for the Project

General TODOs

1. **Documentation**:
    - Add docstrings to all functions and classes to improve code readability and maintainability.

2. **Logging**:
    - Implement a more generic logging system to log messages from all services.
3. **Error Handling**:
    - Improve error handling in all scripts.

Pipelines

1. **Configuration Management**:
    - Sensitive information (such as database passwords) should be stored in a secure configuration management system.
    - Create a centralized configuration file for managing all configurations (e.g., database, Kafka) instead of hardcoding them into scripts.
2. **Data Validation**:
    - Implement more robust data validation in the `step2_cleansed_*` scripts to handle edge cases and unexpected data formats.
3. **Testing**:
    - Write unit tests for all critical functions and classes, especially for data processing and validation logic.
    - Implement integration tests to ensure that the entire data pipeline works as expected.
4. **DAG Management**:
    - Create a mechanism to dynamically manage and update Airflow DAGs without manual intervention.
    - Add error handling in the DAG tasks to ensure that failures are logged and retried appropriately.
5. **Performance Monitoring**:
    - Integrate performance monitoring tools to track the response times and resource usage of the pipelines services.
    - Set up alerts for critical failures.

Producer

1. **API Enhancements**:
    - Add more detailed error responses for the API to help clients understand what went wrong.
2. **Kafka Topic Management**:
    - Implement a method to delete or modify existing Kafka topics if needed.

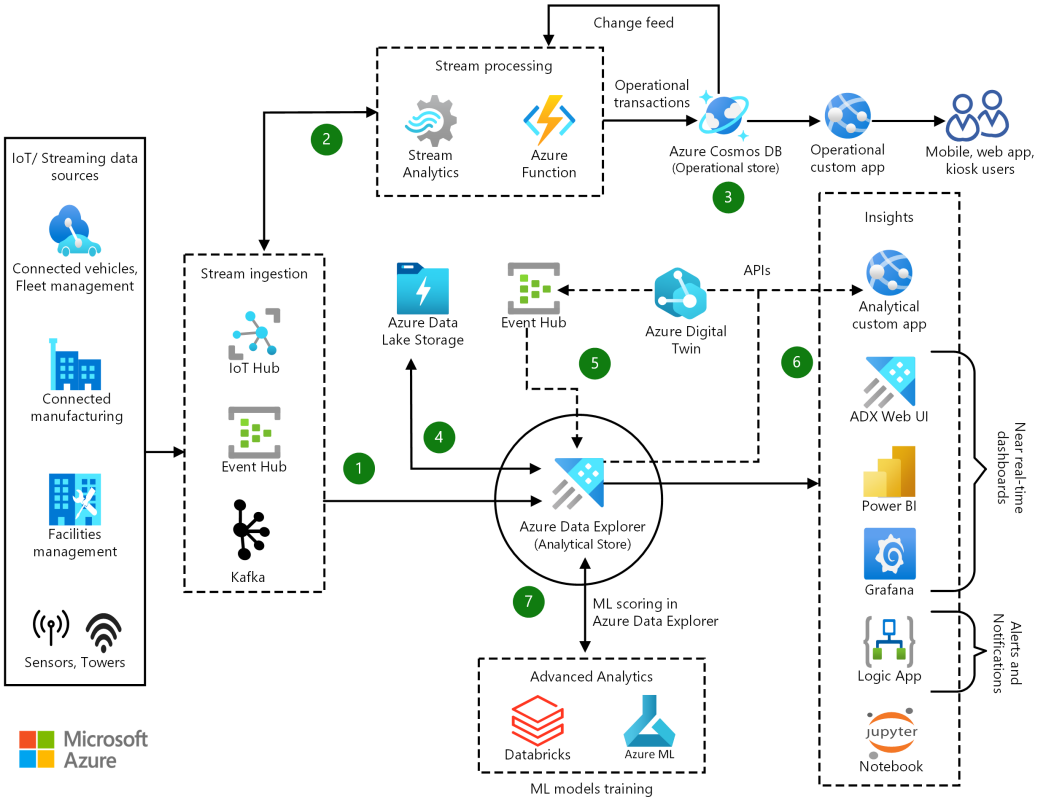
Infra

1. **Docker Optimization**:

- Optimize Dockerfiles by reducing the number of layers.
- Ensure that all services are using the latest stable versions of their dependencies.

2. **Tooling**:

- Migrate to Cloud Services or Kubernetes to manage the infrastructure.
- An improved version of this setup could follow this diagram excluding the stream processing section (Parts 2 and 3) in the diagram.



Back-end

1. **API Versioning**:

- Document the API endpoints and their expected inputs/outputs.

2. **Performance Monitoring**:

- Integrate performance monitoring tools to track the response times and resource usage of the backend services.
- Set up alerts for critical failures or performance degradation.

3. **Security Enhancements**:

- Implement authentication and authorization for the API endpoints to secure access.