Final Project - Data Engineer DigitalSkola

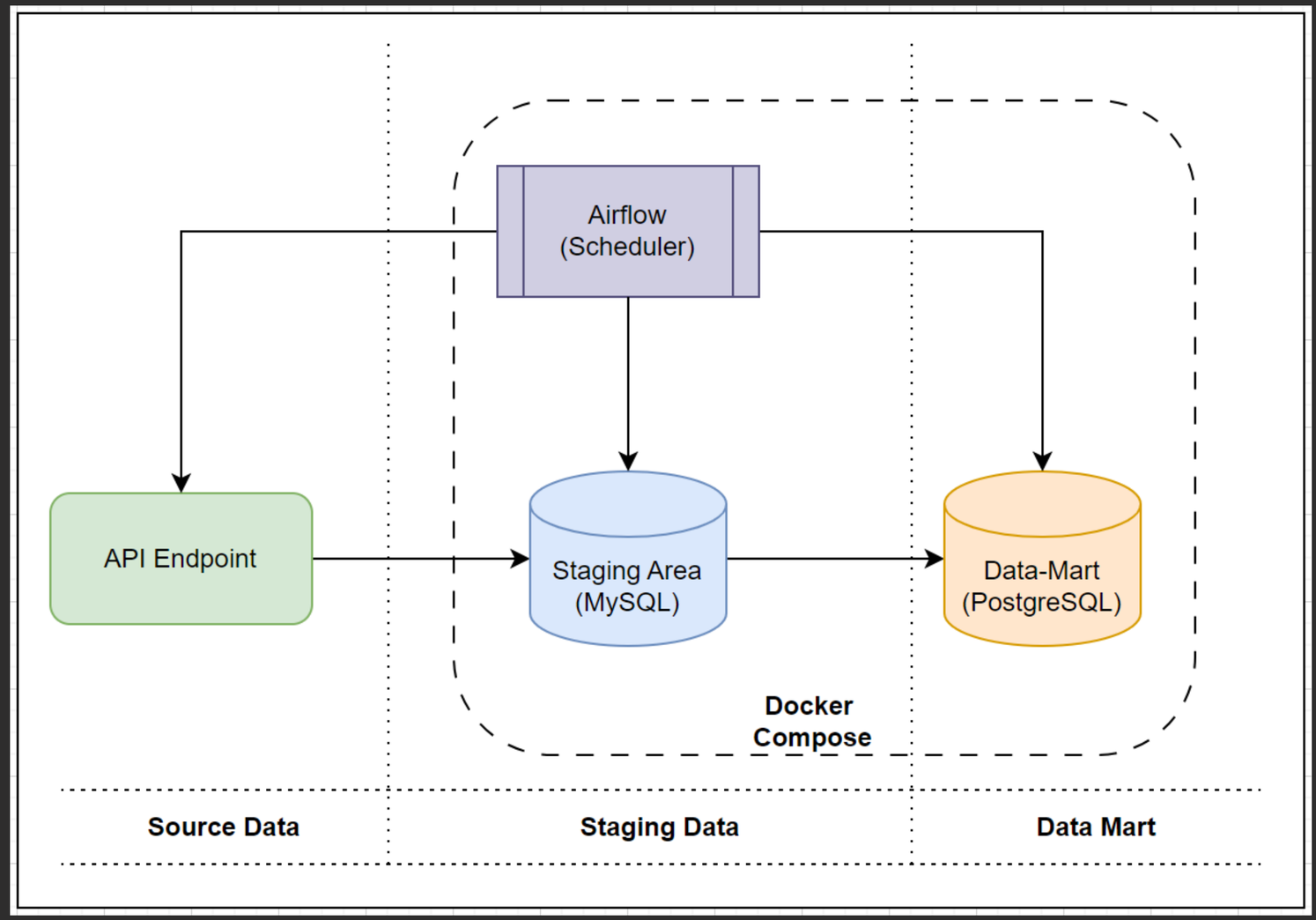
by Axel Ivanda Tanjung

Table of Content

Project Background

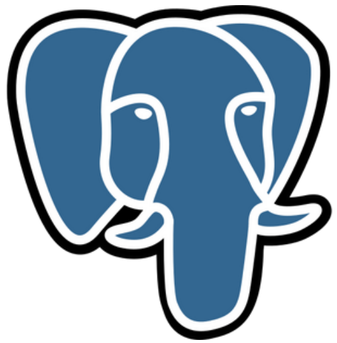
This final project is about Dockerize ETL Pipeline using ETL tools Airflow that extract Public API data from PIKOBAR, then load into MySQL (Staging Area) and finally aggregate the data and save into PostgreSQL.

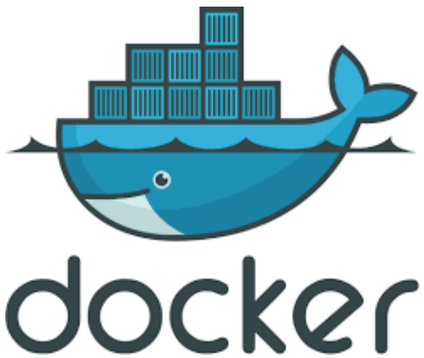
ETL Architecture Diagram



DigitalSkola

Tools





API Specification

- **API Endpoint** : http://103.150.197.96:5005
- **Path** : /api/v1/rekapitulasi_v2/jabar/harian
- **Content-Type** : application/json
- **Request Parameter** : {"level": "kab"}
- **Request method** : GET

REST API basics: CRUD, test & variable / **Get data**

GET http://103.150.197.96:5005/api/v1/rekapitulasi_v2/jabar/harian?level=kab

Params • Authorization Headers (6) Body Pre-request Script Tests • Settings

Body Cookies Headers (4) Test Results (1/1)

Pretty Raw Preview Visualize JSON


```
"data": {
    "content": [
        {
            "CLOSECONTACT": 274,
            "CONFIRMATION": 0,
            "PROBABLE": 26,
            "SUSPECT": 2210,
            "closecontact_dikarantina": 0,
            "closecontact_discarded": 274,
            "closecontact_meninggal": 0,
            "confirmation_meninggal": 0,
            "confirmation_sembuh": 0,
            "kode_kab": "3204",
            "kode_prov": "32",
            "nama_kab": "Kabupaten Bandung",
            "nama_prov": "Jawa Barat",
            "probable_diisolasi": 0,
            "probable_discarded": 0,
            "probable_meninggal": 26,
            "suspect_diisolasi": 31,
            "suspect_discarded": 2179,
            "suspect_meninggal": 0,
            "tanggal": "2020-08-05"
```

Project Step

Create Docker (MySQL, Airflow and PostgreSQL) in local computer

docker-compose.yml

```yaml
version: '3'
x-airflow-common:
  &airflow-common
  image: airflow:2.3.0
  environment:
    - AIRFLOW__CORE__EXECUTOR=LocalExecutor
    - AIRFLOW__CORE__SQL_ALCHEMY_CONN=postgresql+psycopg2://postgres:postgres@postgres:5432/airflow
    - AIRFLOW__CORE__FERNET_KEY=FB0o_zt4e3Ziq3LdUUO7F2Z95cvFFx16hU8jTeR1ASM=
    - AIRFLOW__CORE__LOAD_EXAMPLES=False
    - AIRFLOW__CORE__LOGGING_LEVEL=INFO
  volumes:
    - ./dags:/opt/airflow/dags
    - ./source:/opt/airflow/source
    - ./output:/opt/airflow/output
    - ./airflow-data/logs:/opt/airflow/logs
    - ./airflow-data/plugins:/opt/airflow/plugins
  depends_on:
    - postgres

services:
  postgres:
    image: postgres:12
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
      - POSTGRES_DB=airflow
      - POSTGRES_PORT=5432
    healthcheck:
      test: ["CMD", "pg_isready", "-U", "postgres"]
      interval: 5s
      retries: 5
    ports:
      - "5432:5432"
    networks:
      - airflow-spark
```

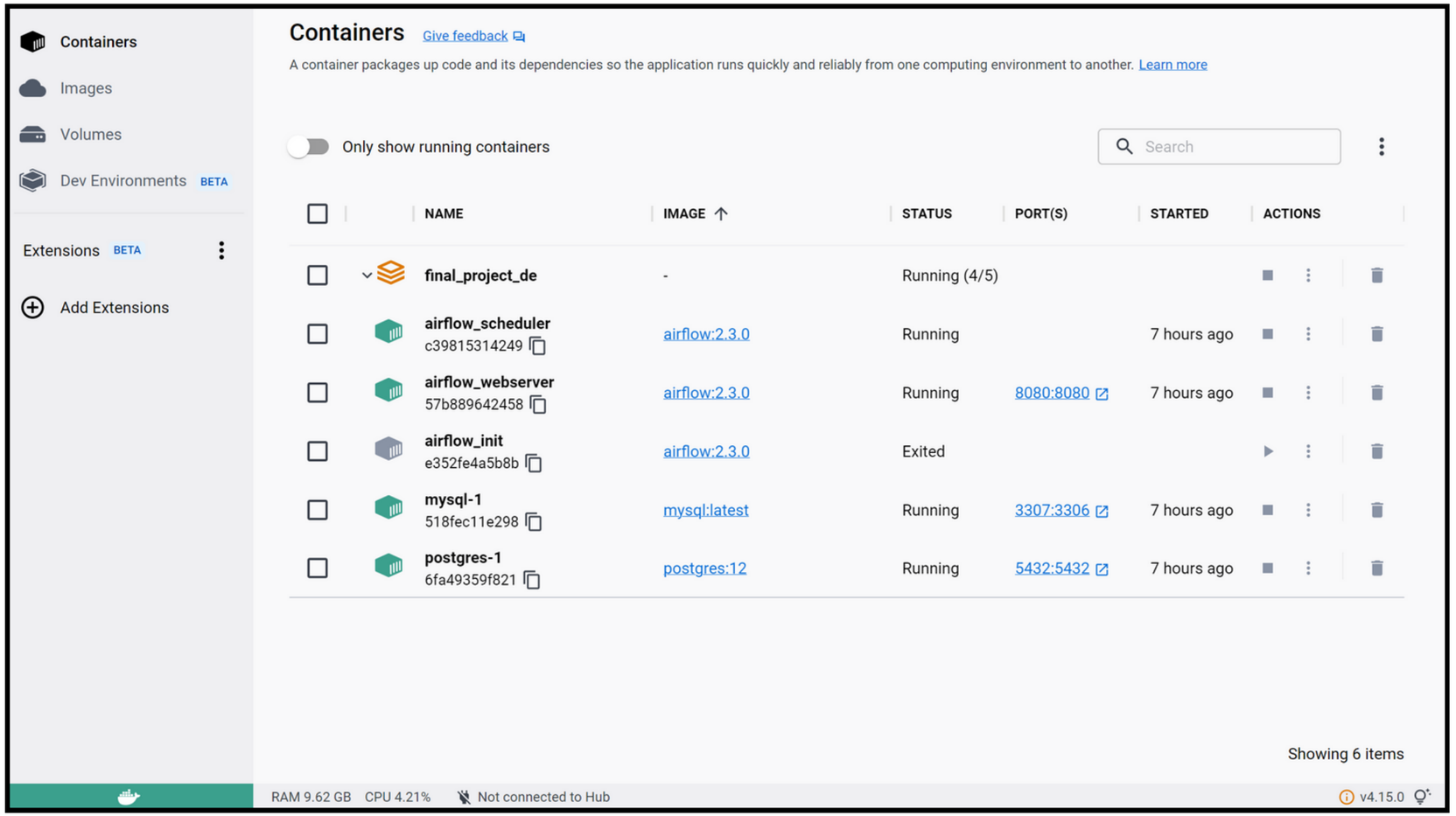
1. Define *docker-compose.yml* consist of:
   - airflow:2.3.0
     - airflow-init
     - airflow-scheduler
     - airflow-webserver
   - postgresql:12
   - mysql:latest

2. Define image, environtment, dependency, volume, port, network, etc in docker compose

3. Running “docker compose up” in terminal

DigitalSkola

Project Step



Project Step

Create Database in MySQL and PostgreSQL

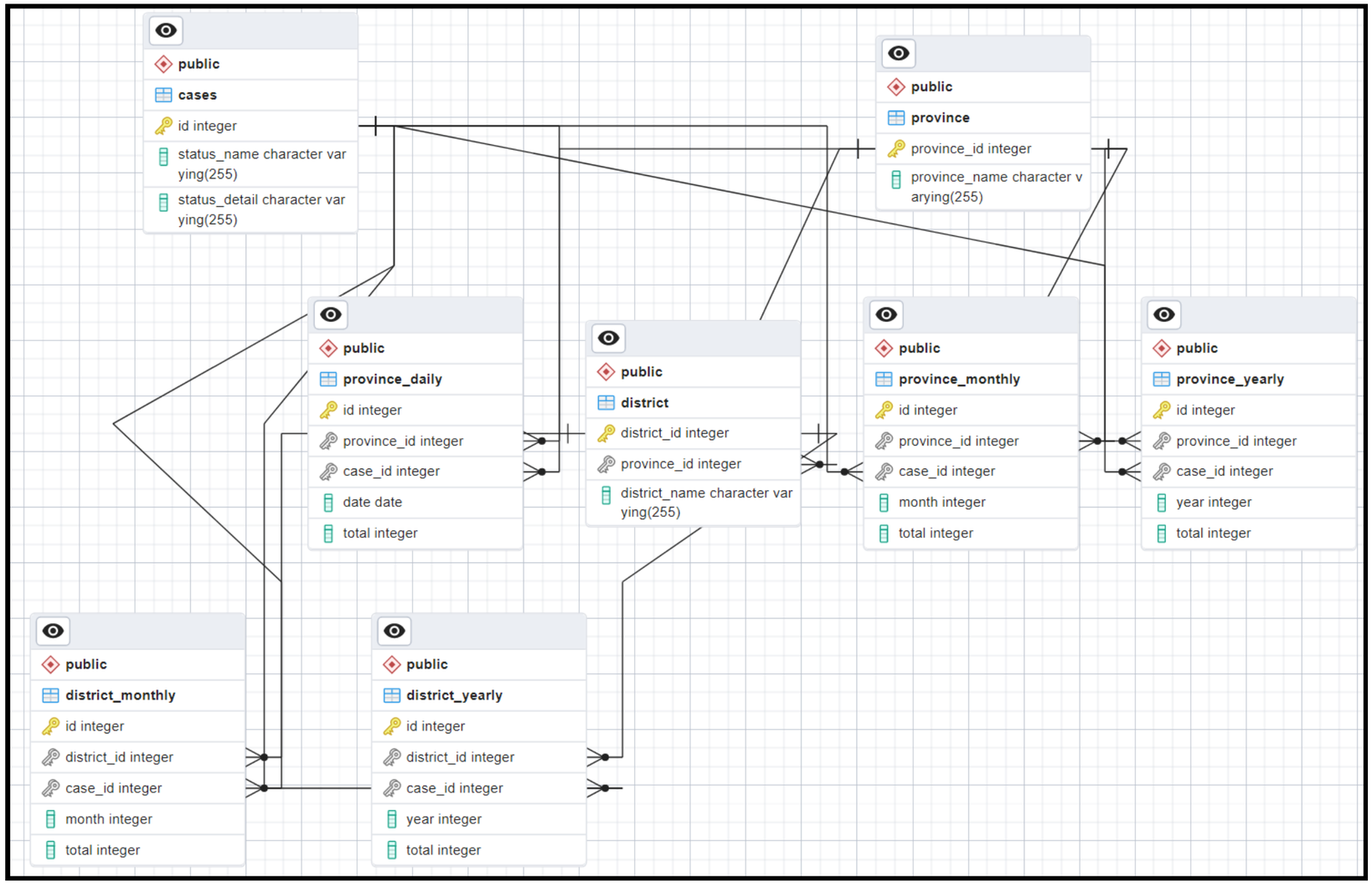
```sql
-- CREATE DATABASE
CREATE DATABASE IF NOT EXISTS staging_area;
USE staging_area;

-- CREATE TABLE
CREATE TABLE IF NOT EXISTS staging_table (
    CLOSECONTACT INT,
    CONFIRMATION INT,
    PROBABLE INT,
    SUSPECT INT,
    closecontact_dikarantina INT,
    closecontact_discarded INT,
    closecontact_meninggal INT,
    confirmation_meninggal INT,
    confirmation_sembuh INT,
    kode_kab VARCHAR(255),
    kode_prov VARCHAR(255),
    nama_kab VARCHAR(255),
    nama_prov VARCHAR(255),
    probable_diisolasi INT,
    probable_discarded INT,
    probable_meninggal INT,
    suspect_diisolasi INT,
    suspect_discarded INT,
    suspect_meninggal INT,
    tanggal DATE
);

DROP DATABASE staging_area;

SELECT * FROM staging_table st ;
```

Staging database in MySQL using as is schema like data source

```sql
-- CREATE DATABASE
CREATE DATABASE data_mart;

-- CREATE TABLES
CREATE TABLE Province (
    province_id SERIAL PRIMARY KEY,
    province_name VARCHAR(255)
);

CREATE TABLE District (
    district_id SERIAL PRIMARY KEY,
    province_id INT,
    district_name VARCHAR(255),
    FOREIGN KEY (province_id) REFERENCES Province(province_id)
);

CREATE TABLE Cases (
    Id SERIAL PRIMARY KEY,
    status_name VARCHAR(255) CHECK (status_name IN ('suspect', 'closecontact', 'probable',
    status_detail VARCHAR(255)
);

CREATE TABLE Province_Daily (
    Id SERIAL PRIMARY KEY,
    province_id INT,
    case_id INT,
    date DATE,
    total INT,
    FOREIGN KEY (province_id) REFERENCES Province(province_id),
    FOREIGN KEY (case_id) REFERENCES Cases(Id)
);
```

Data mart in postgres using Fact Table and Dimension Table Schema

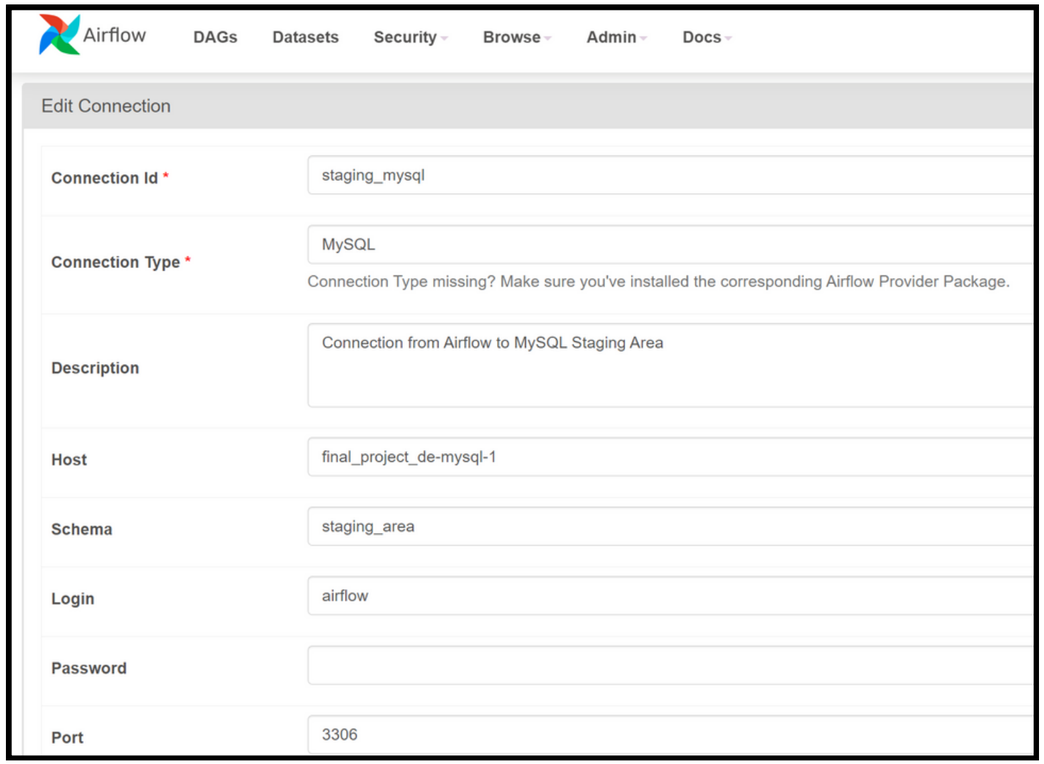
Project Step



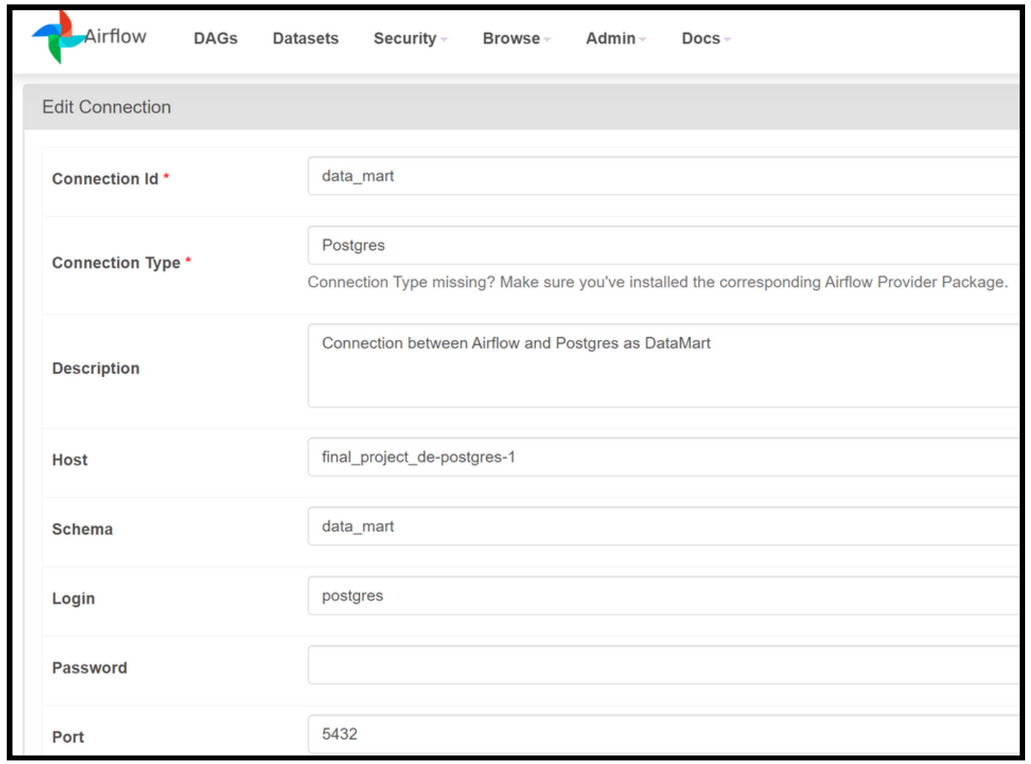
Fact Table and Dimension Table Schema

Project Step

Create Connection from Airflow to MySQL and PostgreSQL



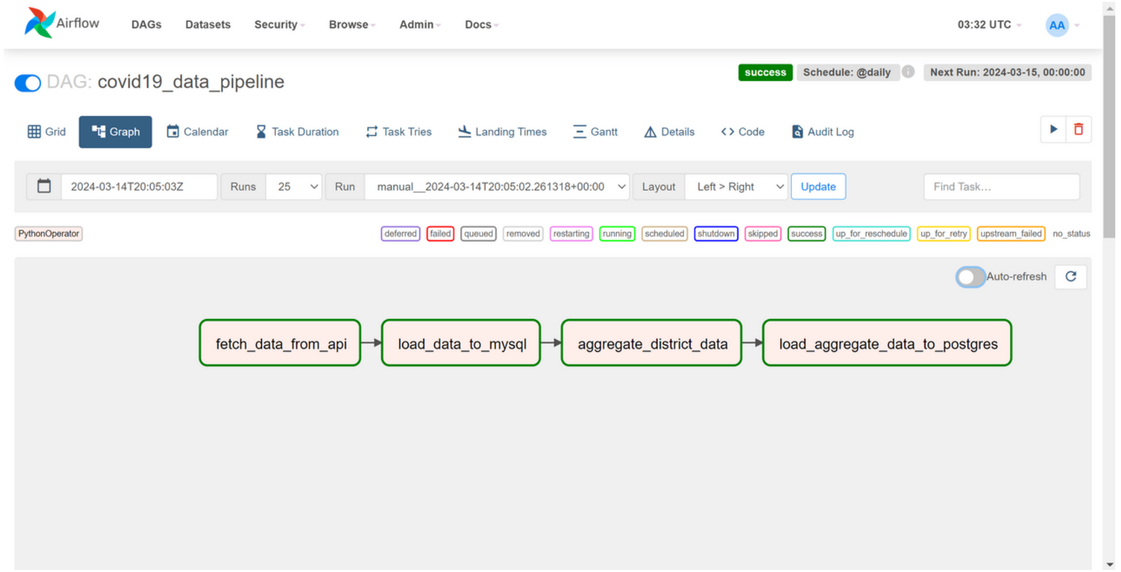
Airflow to MySQL



Airflow to PostgreSQL

Project Step

Create Directed Acyclic Graph




```python
# Define DAG
with DAG(
    dag_id='covid19_data_pipeline',
    default_args={
        'owner': 'airflow',
        'start_date': datetime(2024, 3, 11),
        'retries': 1
    },
    schedule_interval='@daily',
    catchup=True
) as dag:

    # Define tasks
    fetch_data_task = PythonOperator(
        task_id='fetch_data_from_api',
        python_callable=fetch_data_from_api,
        dag=dag
    )

    load_data_to_mysql_task = PythonOperator(
        task_id='load_data_to_mysql',
        python_callable=load_data_to_mysql,
        dag=dag
    )

    aggregate_district_data_task = PythonOperator(
        task_id='aggregate_district_data',
        python_callable=aggregate_district_data,
        dag=dag
    )

    load_aggregate_data_to_postgres_task = PythonOperator(
        task_id='load_aggregate_data_to_postgres',
        python_callable=load_aggregate_data_to_postgres,
        dag=dag
    )

    # Define task dependencies
    fetch_data_task >> load_data_to_mysql_task >> aggregate_district_data_task >> load_aggregate_data_to_postgres_task
```

Project Step

Fetch Data from API

```python
# Function to fetch data from API
def fetch_data_from_api():
    url = 'http://103.150.197.96:5005/api/v1/rekapitulasi_v2/jabar/harian?level=kab'
    response = requests.get(url)
    data = response.json()

    return data
    # Process data and return
```

Project Step

Load Data to MySQL (Staging Area)

```python
# Function to load data to MySQL
def load_data_to_mysql(**kwargs):
    ti = kwargs['ti']
    data = ti.xcom_pull(task_ids='fetch_data_from_api')
    records = data['data']['content']

    # Connect to MySQL database
    connection = mysql.connector.connect(
        host='final_project_de-mysql-1',
        user='airflow',
        password='airflow',
        database='staging_area'
    )

    cursor = connection.cursor()
    # Load data to MySQL
    # Example:
    # cursor.execute("INSERT INTO table_name (column1, column2) VALUES (%s, %s)", (value1, value2))
    # Insert data into MySQL table

    for record in records:
        insert_query = """
        INSERT INTO staging_table (CLOSECONTACT, CONFIRMATION, PROBABLE, SUSPECT, closecontact_dikarantina, closecontact_discarded, closecontact_meninggal,
                                   confirmation_meninggal, confirmation_sembuh, kode_kab, kode_prov, nama_kab, nama_prov, probable_diisolasi, probable_discarded,
                                   probable_meninggal, suspect_diisolasi, suspect_discarded, suspect_meninggal,tanggal)
        VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s,
                %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
        """
        cursor.execute(insert_query, (record['CLOSECONTACT'], record['CONFIRMATION'], record['PROBABLE'], record['SUSPECT'], record['closecontact_dikarantina'],
                                      record['closecontact_discarded'], record['closecontact_meninggal'], record['confirmation_meninggal'], record['confirmation_sembuh'], record['kode_kab'],
                                      record['kode_prov'], record['nama_kab'], record['nama_prov'], record['probable_diisolasi'], record['probable_discarded'],
                                      record['probable_meninggal'], record['suspect_diisolasi'], record['suspect_discarded'], record['suspect_meninggal'], record['tanggal']))

    connection.commit()
    connection.close()
```

Project Step

Aggregate Data on Staging (Dimension Table) - Province & District

```python
# Function to aggregate district data
def aggregate_district_data():
    # Fetch data from MySQL (Staging Area)
    # Aggregate district data
    # Connect to MySQL database
    connection = mysql.connector.connect(
        host='final_project_de-mysql-1',
        user='airflow',
        password='airflow',
        database='staging_area'
    )

    engine = create_engine('mysql+pymysql://airflow:airflow@final_project_de-mysql-1/staging_area')

    # Define your SQL query
    sql_query = "SELECT * FROM staging_table;"

    # Use pandas.read_sql() to execute the query and load data into a DataFrame
    df_staging = pd.read_sql(sql_query, connection)

    # Create Dimension Table
    # Create province table
    province_table = df_staging[['kode_prov', 'nama_prov']]
    province_table = province_table.drop_duplicates()
    province_table.rename(columns={'kode_prov':'province_id', 'nama_prov':'province_name'}, inplace=True)

    try:
        province_table.to_sql(name='Province', con=engine, if_exists='append', index=False)
    except:
        pass

    # # Create district table
    district_table = df_staging[['kode_kab', 'kode_prov', 'nama_kab']]
    district_table = district_table.drop_duplicates()
    district_table.rename(columns={'kode_kab':'district_id', 'kode_prov':'province_id', 'nama_kab':'district_name'}, inplace=True)

    try:
        district_table.to_sql(name='District', con=engine, if_exists='append', index=False)
    except:
        pass
```

Project Step

Aggregate Data on Staging (Dimension Table - Case)

```
# Create case table
case_table_dict = { 'case_id' : [1,2,3,4,5,6,7,8,9,10,11],
                    'status_name': ['CLOSECONTACT','CLOSECONTACT','CLOSECONTACT',
                         'CONFIRMATION','CONFIRMATION',
                         'PROBABLE','PROBABLE','PROBABLE',
                         'SUSPECT', 'SUSPECT', 'SUSPECT'],
                    'status_detail': ['closecontact_dikarantina', 'closecontact_discarded', 'closecontact_meninggal',
                           'confirmation_meninggal', 'confirmation_sembuh',
                           'probable_diisolasi', 'probable_discarded', 'probable_meninggal',
                           'suspect_diisolasi', 'suspect_discarded', 'suspect_meninggal']
                  }

case_table = pd.DataFrame(case_table_dict)

try:
    case_table.to_sql(name='Cases', con=engine, if_exists='replace', index=False)
except:
    pass
```

Project Step

Aggregate Data on Staging (Fact Table)

```
# Create Fact Table
# Aggregate for Province_Daily
# Group by province and calculate the total cases for each status
sql_query_province_daily = '''
SELECT
    CASE
        WHEN t.nama_prov IS NOT NULL THEN p.province_id
        ELSE NULL
    END AS province_id,
    t.tanggal AS date,
    t.CLOSECONTACT + t.CONFIRMATION + t.PROBABLE + t.SUSPECT AS total
FROM staging_table t
LEFT JOIN Province p ON t.nama_prov = p.province_name;
'''
#c.case_id AS case_id,
# LEFT JOIN Cases c ON t.tanggal = c.tanggal;
df_staging_province_daily = pd.read_sql(sql_query_province_daily, connection)

# Aggregate for Province_Daily
# Group by province and calculate the total cases for each status
sql_query_province_monthly = '''
SELECT
    CASE
        WHEN t.nama_prov IS NOT NULL THEN p.province_id
        ELSE NULL
    END AS province_id,
    MONTH(t.tanggal) AS month,
    SUM(t.CLOSECONTACT + t.CONFIRMATION + t.PROBABLE + t.SUSPECT) AS total
FROM staging_table t
LEFT JOIN Province p ON t.nama_prov = p.province_name
GROUP BY
    CASE
        WHEN t.nama_prov IS NOT NULL THEN p.province_id
        ELSE NULL
    END,
    MONTH(t.tanggal);
'''
df_staging_province_monthly = pd.read_sql(sql_query_province_monthly, connection)
```

```
# Aggregate for Province_Yearly
# Group by province and calculate the total cases for each status
sql_query_province_yearly = '''
SELECT
    CASE
        WHEN t.nama_prov IS NOT NULL THEN p.province_id
        ELSE NULL
    END AS province_id,
    YEAR(t.tanggal) AS year,
    SUM(t.CLOSECONTACT + t.CONFIRMATION + t.PROBABLE + t.SUSPECT) AS total
FROM staging_table t
LEFT JOIN Province p ON t.nama_prov = p.province_name
GROUP BY
    CASE
        WHEN t.nama_prov IS NOT NULL THEN p.province_id
        ELSE NULL
    END,
    YEAR(t.tanggal);
'''
df_staging_province_yearly = pd.read_sql(sql_query_province_yearly, connection)

# Aggregate for District_Monthly
# Group by province and calculate the total cases for each status
sql_query_district_monthly = '''
SELECT
    CASE
        WHEN t.nama_kab IS NOT NULL THEN d.district_id
        ELSE NULL
    END AS district_id,
    MONTH(t.tanggal) AS month,
    SUM(t.CLOSECONTACT + t.CONFIRMATION + t.PROBABLE + t.SUSPECT) AS total
FROM staging_table t
LEFT JOIN District d ON t.nama_kab = d.district_name
GROUP BY
CASE
    WHEN t.nama_kab IS NOT NULL THEN d.district_id
    ELSE NULL
END,
MONTH(t.tanggal);
'''
df_staging_district_monthly = pd.read_sql(sql_query_district_monthly, connection)
```

Project Step

Aggregate Data on Staging (Fact Table)

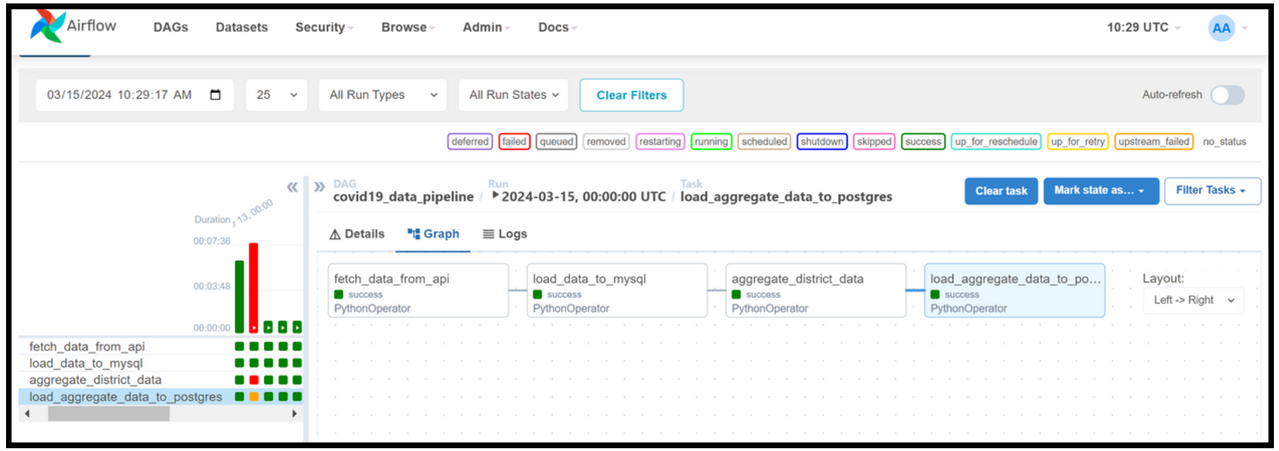
```
# Aggregate for District_yearly
# Group by province and calculate the total cases for each status
sql_query_district_yearly = '''
SELECT
    CASE
        WHEN t.nama_kab IS NOT NULL THEN d.district_id
        ELSE NULL
    END AS district_id,
    YEAR(t.tanggal) AS year,
    SUM(t.CLOSECONTACT + t.CONFIRMATION + t.PROBABLE + t.SUSPECT) AS total
FROM staging_table t
LEFT JOIN District d ON t.nama_kab = d.district_name
GROUP BY
CASE
    WHEN t.nama_kab IS NOT NULL THEN d.district_id
    ELSE NULL
END,
YEAR(t.tanggal);
'''
df_staging_district_yearly = pd.read_sql(sql_query_district_yearly, connection)

return df_staging_province_daily, df_staging_province_monthly, df_staging_province_yearly, df_staging_district_monthly, df_staging_district_yearly
```

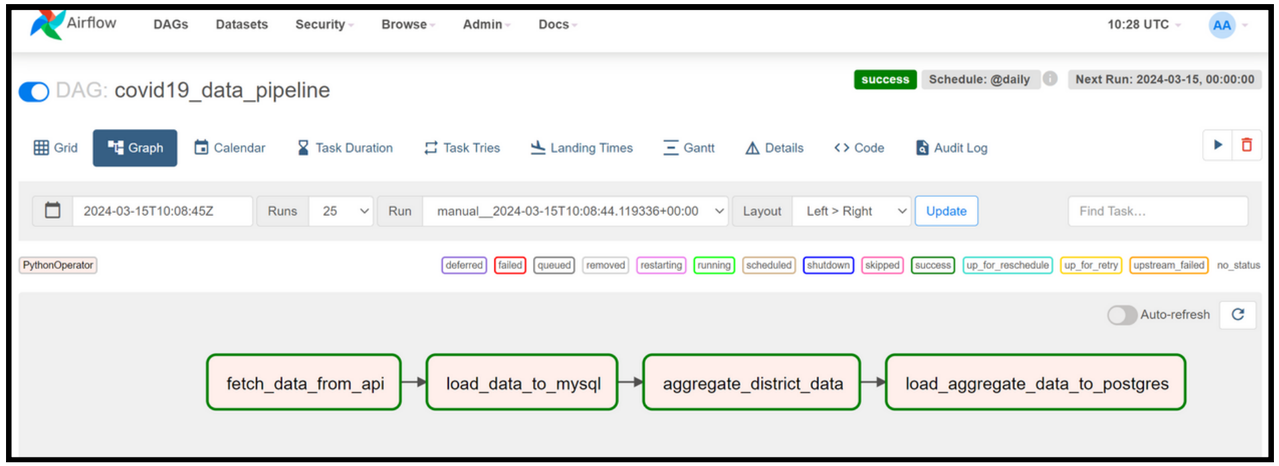
Project Step

Load Aggegate Data to Postgres

```python
# Function to load aggregate data to PostgreSQL
def load_aggregate_data_to_postgres(**kwargs):
    ti = kwargs['ti']
    # Retrieve DataFrames from XCom
    df_staging_province_daily = ti.xcom_pull(task_ids='aggregate_district_data')[0]
    df_staging_province_monthly = ti.xcom_pull(task_ids='aggregate_district_data')[1]
    df_staging_province_yearly = ti.xcom_pull(task_ids='aggregate_district_data')[2]
    df_staging_district_monthly = ti.xcom_pull(task_ids='aggregate_district_data')[3]
    df_staging_district_yearly = ti.xcom_pull(task_ids='aggregate_district_data')[4]

    # Connect to PostgreSQL database
    connection = psycopg2.connect(
        host='final_project_de-postgres-1',
        user='postgres',
        password='postgres',
        database='data_mart'
    )
    # Create a connection string
    connection_string = f'postgresql://postgres:postgres@final_project_de-postgres-1/data_mart'

    # Create SQLAlchemy engine
    engine = create_engine(connection_string)
    # Load aggregate data to PostgreSQL
    cursor = connection.cursor()

    # Write DataFrame to PostgreSQL database
    df_staging_province_daily.to_sql(name='Province_Daily', con=engine, if_exists='append', index=False, method='multi', chunksize=1000)
    df_staging_province_monthly.to_sql(name='Province_Monthly', con=engine, if_exists='append', index=False, method='multi', chunksize=1000)
    df_staging_province_yearly.to_sql(name='Province_Yearly', con=engine, if_exists='append', index=False, method='multi', chunksize=1000)
    df_staging_district_monthly.to_sql(name='District_Monthly', con=engine, if_exists='append', index=False, method='multi', chunksize=1000)
    df_staging_district_yearly.to_sql(name='District_Yearly', con=engine, if_exists='append', index=False, method='multi', chunksize=1000)

    connection.commit()
    connection.close()
```

Airflow Output






```python
with DAG(
    dag_id='covid19_data_pipeline',
    default_args={
        'owner': 'airflow',
        'start_date': datetime(2024, 3, 11),
        'retries': 1
    },
    schedule_interval='@daily',
    catchup=True
```

Thank You!