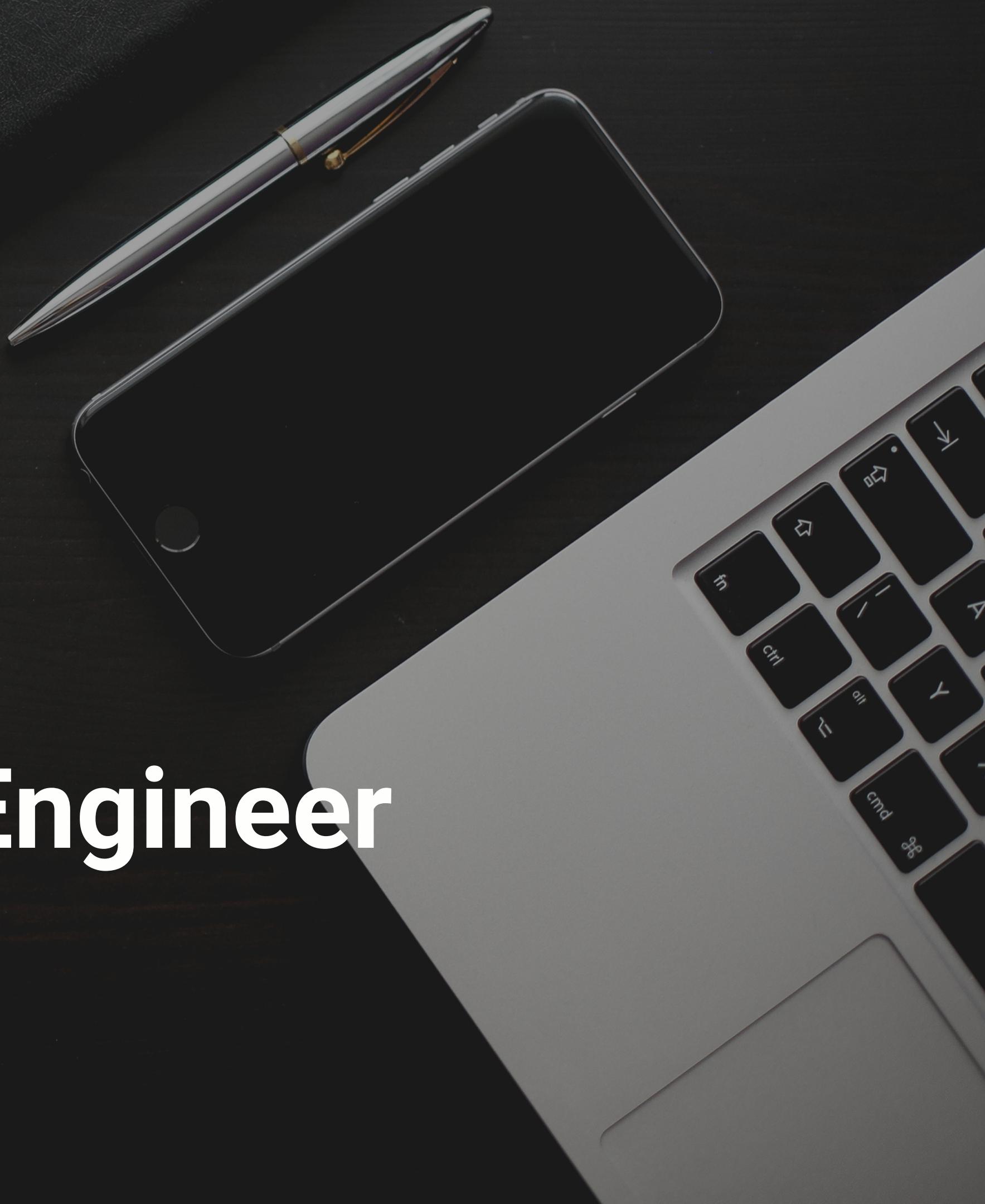




# Final Project - Data Engineer

# DigitalSkola

by Axel Ivanda Tanjung





# Table of Content

Project Background

ETL Architecture Diagram

Tools

API Specification

Project Step

Airflow Output

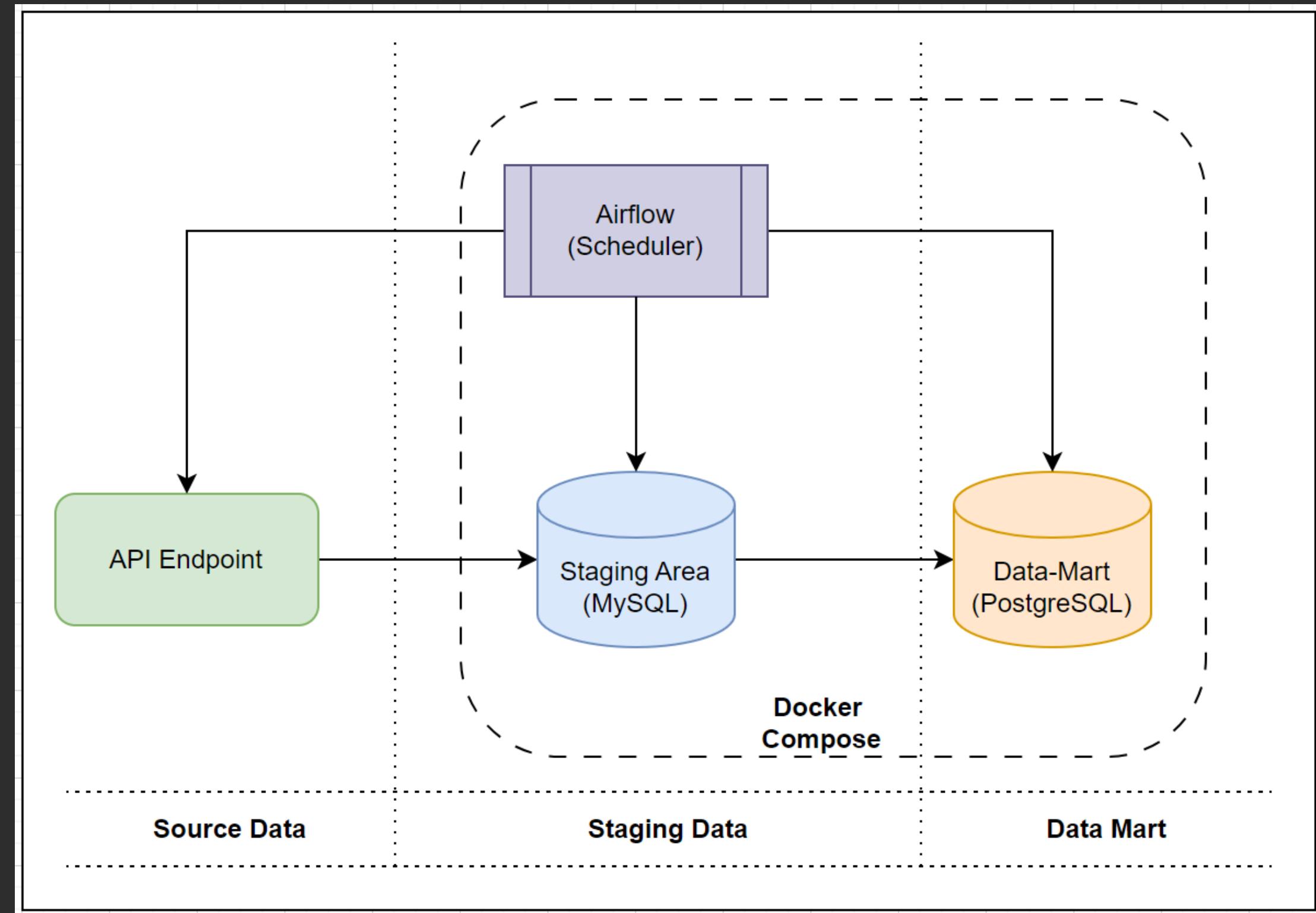
Data Mart Output

# Project Background

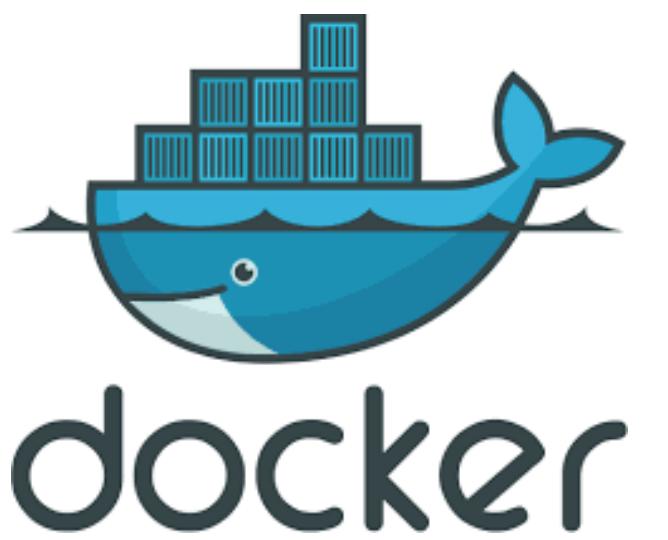
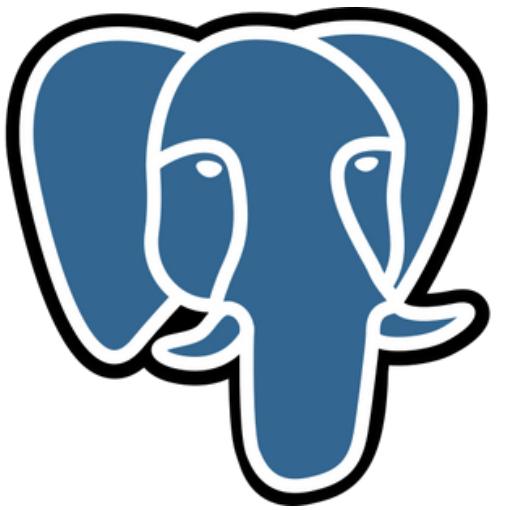
This final project is about Dockerize ETL Pipeline using ETL tools Airflow that extract Public API data from PIKOBAR, then load into MySQL (Staging Area) and finally aggregate the data and save into PostgreSQL.



# ETL Architecture Diagram

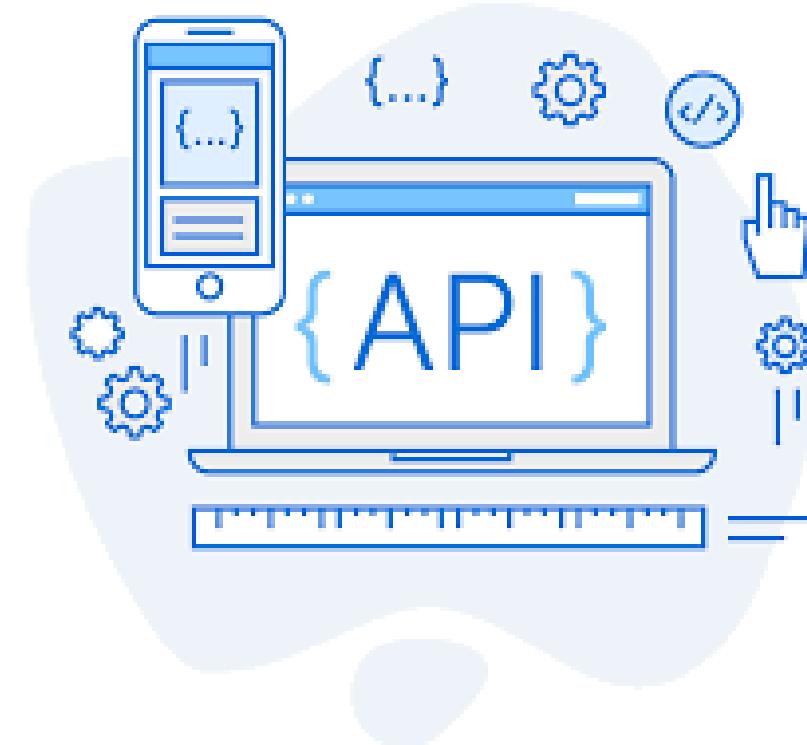


# Tools



# 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



HTTP 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 ↻

```
2   "data": {  
3     "content": [  
4       {  
5         "CLOSECONTACT": 274,  
6         "CONFIRMATION": 0,  
7         "PROBABLE": 26,  
8         "SUSPECT": 2210,  
9         "closecontact_dikarantina": 0,  
10        "closecontact_discarded": 274,  
11        "closecontact_meninggal": 0,  
12        "confirmation_meninggal": 0,  
13        "confirmation_sembuh": 0,  
14        "kode_kab": "3204",  
15        "kode_prov": "32",  
16        "nama_kab": "Kabupaten Bandung",  
17        "nama_prov": "Jawa Barat",  
18        "probable_diisolasi": 0,  
19        "probable_discarded": 0,  
20        "probable_meninggal": 26,  
21        "suspect_diisolasi": 31,  
22        "suspect_discarded": 2179,  
23        "suspect_meninggal": 0,  
24        "tanggal": "2020-08-05"  
      }  
    ]  
  }  
}
```

# Project Step

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

```
# docker-compose.yml
1  version: '3'
2  x-airflow-common:
3    &airflow-common
4      image: airflow:2.3.0
5      environment:
6        - AIRFLOW__CORE__EXECUTOR=LocalExecutor
7        - AIRFLOW__CORE__SQLALCHEMY_CONN=postgresql+psycopg2://postgres:postgres@postgres:5432/airflow
8        - AIRFLOW__CORE__FERNET_KEY=FB0o_zt4e3Ziq3LdUU07F2Z95cvFFx16hU8jTeR1ASM=
9        - AIRFLOW__CORE__LOAD_EXAMPLES=False
10       - AIRFLOW__CORE__LOGGING_LEVEL=INFO
11       volumes:
12         - ./dags:/opt/airflow/dags
13         - ./source:/opt/airflow/source
14         - ./output:/opt/airflow/output
15         - ./airflow-data/logs:/opt/airflow/logs
16         - ./airflow-data/plugins:/opt/airflow/plugins
17       depends_on:
18         - postgres
19
20   services:
21     postgres:
22       image: postgres:12
23       environment:
24         - POSTGRES_USER=postgres
25         - POSTGRES_PASSWORD=postgres
26         - POSTGRES_DB=airflow
27         - POSTGRES_PORT=5432
28       healthcheck:
29         test: ["CMD", "pg_isready", "-U", "postgres"]
30         interval: 5s
31         retries: 5
32       ports:
33         - "5432:5432"
34       networks:
35         - 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

# Project Step

**Containers** [Give feedback](#)

A container packages up code and its dependencies so the application runs quickly and reliably from one computing environment to another. [Learn more](#)

Only show running containers  ⋮

	NAME	IMAGE ↑	STATUS	PORT(S)	STARTED	ACTIONS
<input type="checkbox"/>	final_project_de	-	Running (4/5)			<span>⋮</span> <span>⋮</span> <span>⋮</span>
<input type="checkbox"/>	airflow_scheduler c39815314249 <span>⋮</span>	airflow:2.3.0	Running	8080:8080 <span>⋮</span>	7 hours ago	<span>⋮</span> <span>⋮</span> <span>⋮</span>
<input type="checkbox"/>	airflow_webserver 57b889642458 <span>⋮</span>	airflow:2.3.0	Running	8080:8080 <span>⋮</span>	7 hours ago	<span>⋮</span> <span>⋮</span> <span>⋮</span>
<input type="checkbox"/>	airflow_init e352fe4a5b8b <span>⋮</span>	airflow:2.3.0	Exited			<span>⋮</span> <span>⋮</span> <span>⋮</span>
<input type="checkbox"/>	mysql-1 518fec11e298 <span>⋮</span>	mysql:latest	Running	3307:3306 <span>⋮</span>	7 hours ago	<span>⋮</span> <span>⋮</span> <span>⋮</span>
<input type="checkbox"/>	postgres-1 6fa49359f821 <span>⋮</span>	postgres:12	Running	5432:5432 <span>⋮</span>	7 hours ago	<span>⋮</span> <span>⋮</span> <span>⋮</span>

Showing 6 items

⋮ RAM 9.62 GB CPU 4.21% ⋮ Not connected to Hub ⋮ v4.15.0 ⋮

# Project Step

## Create Database in MySQL and PostgreSQL

```

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

```

-- 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)
);

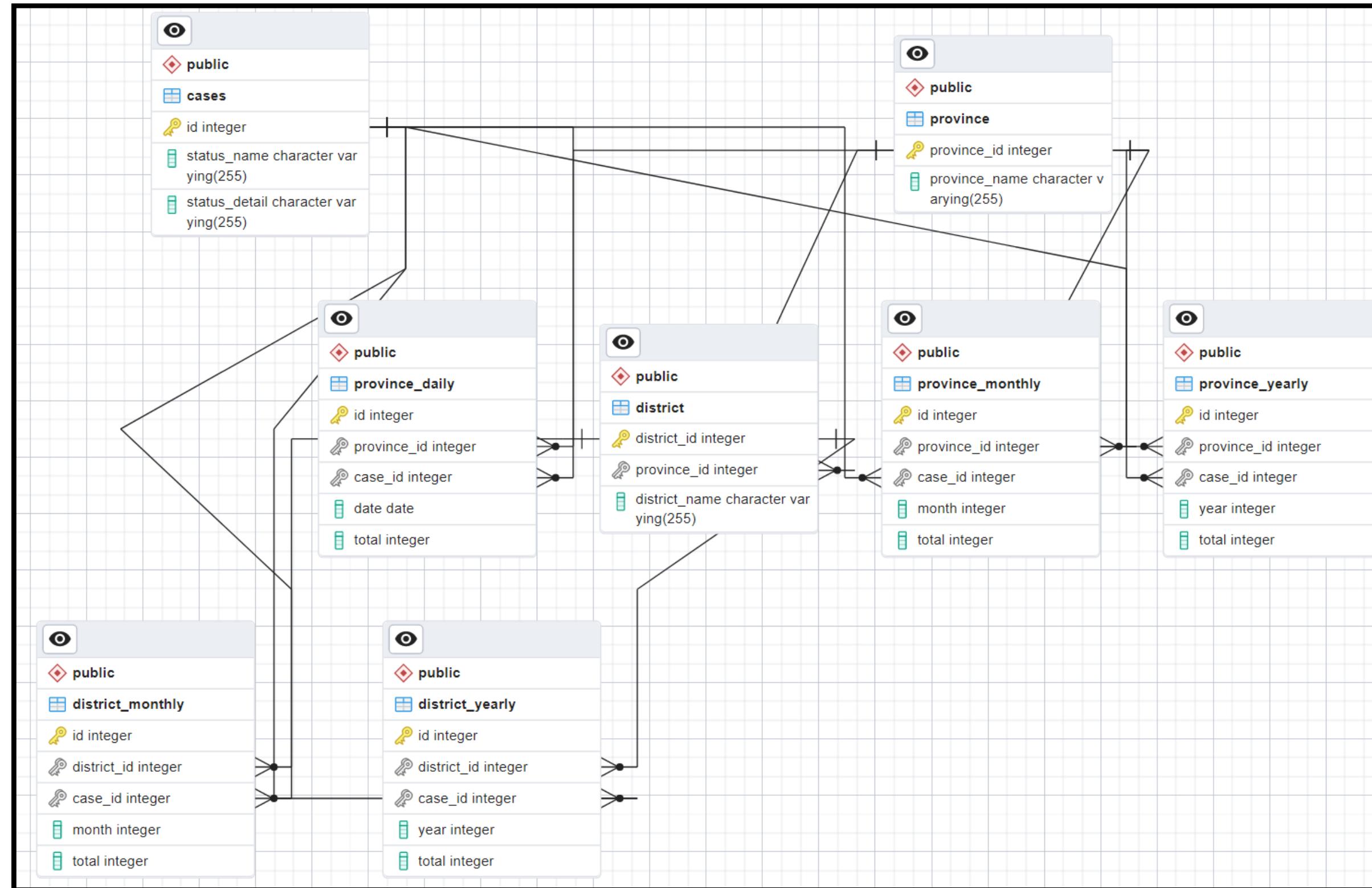
CREATE TABLE Cases (
    Id SERIAL PRIMARY KEY,
    status_name VARCHAR(255) CHECK (status_name IN ('suspect', 'closecontact', 'probable'),
    status_detail VARCHAR(255)
);

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

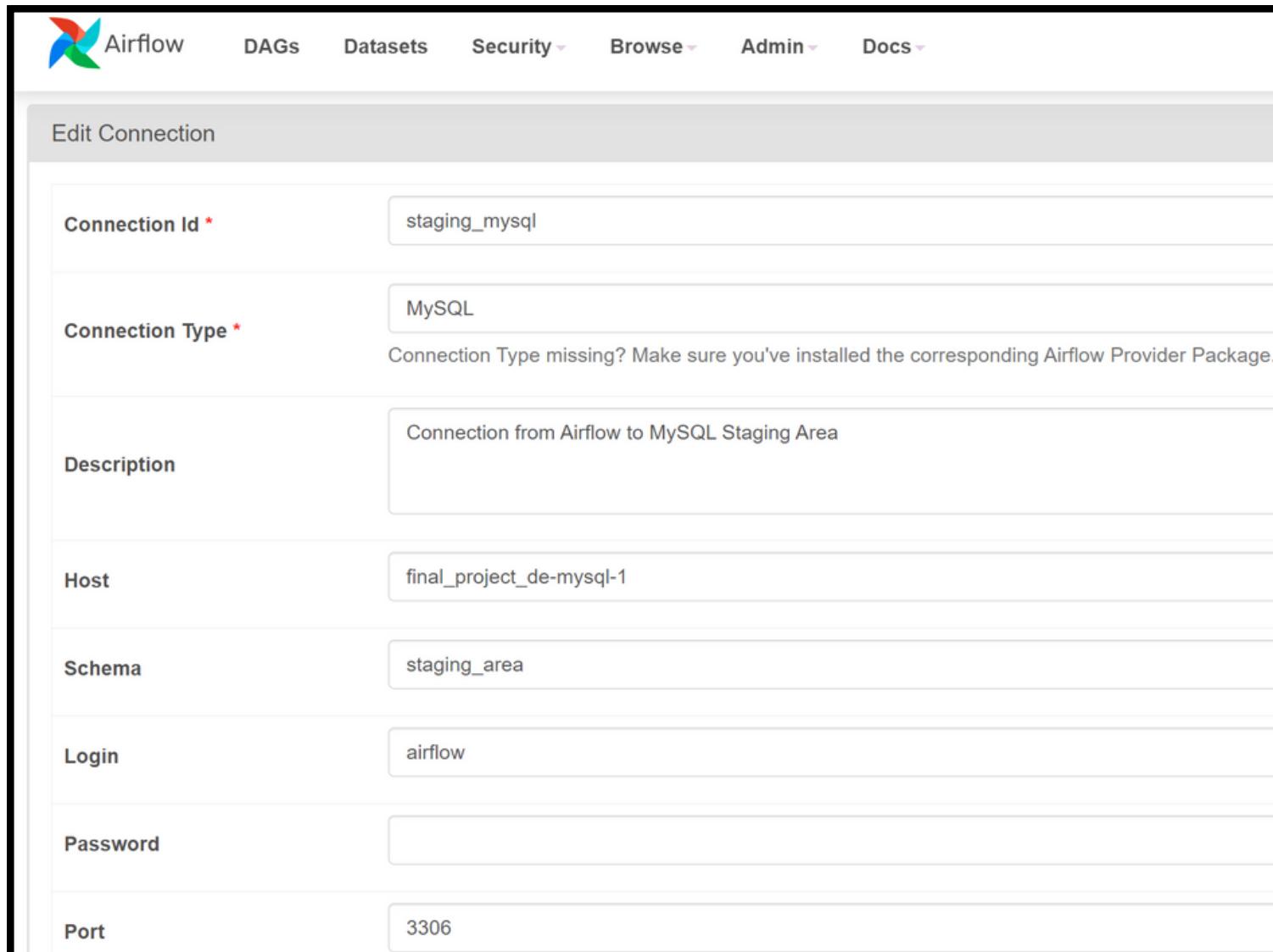
# Project Step



Fact Table and Dimension Table Schema

# Project Step

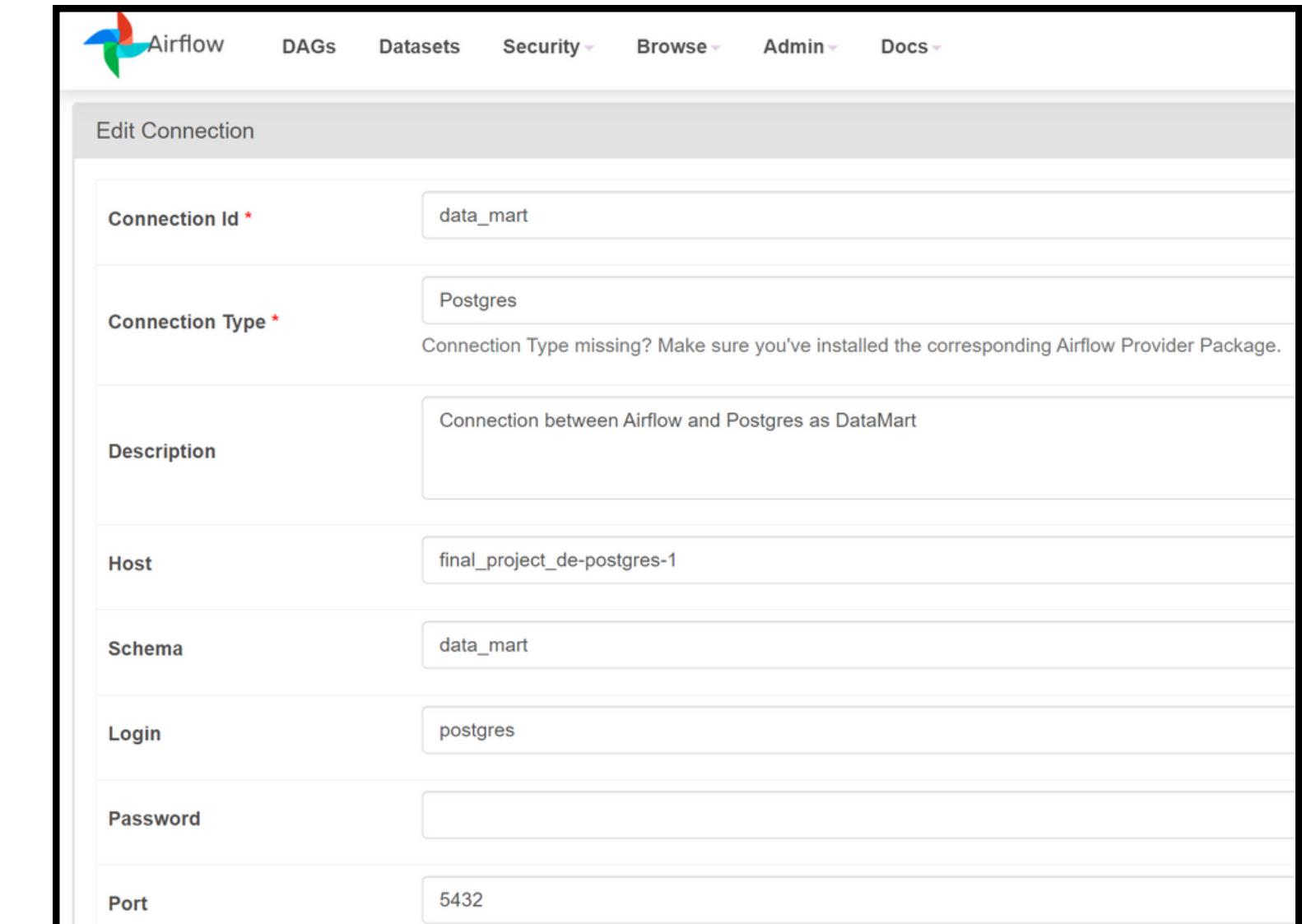
## Create Connection from Airflow to MySQL and PostgreSQL



The screenshot shows the 'Edit Connection' form in the Airflow web interface. The connection is named 'staging\_mysql' and is of type 'MySQL'. The description is 'Connection from Airflow to MySQL Staging Area'. The host is 'final\_project\_de-mysql-1', the schema is 'staging\_area', the login is 'airflow', and the port is 3306. The password field is empty.

Field	Value
Connection Id *	staging_mysql
Connection Type *	MySQL
Description	Connection from Airflow to MySQL Staging Area
Host	final_project_de-mysql-1
Schema	staging_area
Login	airflow
Password	
Port	3306

Airflow to MySQL



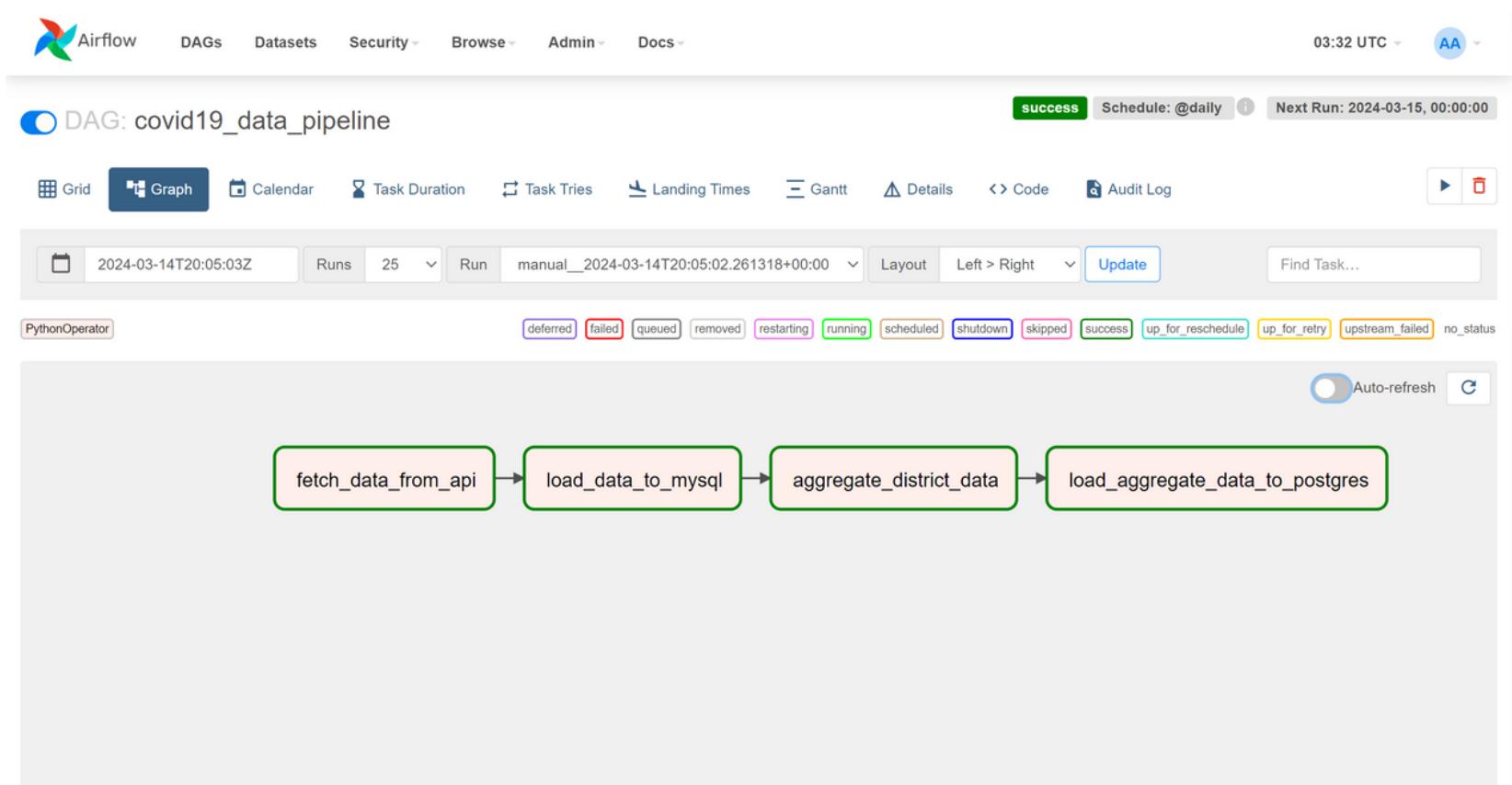
The screenshot shows the 'Edit Connection' form in the Airflow web interface. The connection is named 'data\_mart' and is of type 'Postgres'. The description is 'Connection between Airflow and Postgres as DataMart'. The host is 'final\_project\_de-postgres-1', the schema is 'data\_mart', the login is 'postgres', and the port is 5432. The password field is empty.

Field	Value
Connection Id *	data_mart
Connection Type *	Postgres
Description	Connection between Airflow and Postgres as DataMart
Host	final_project_de-postgres-1
Schema	data_mart
Login	postgres
Password	
Port	5432

Airflow to PostgreSQL

# Project Step

## Create Directed Acyclic Graph



```

# 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

```
# 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)

```
# 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)
        """
        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

```
# 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 Aggregate Data to Postgres

```
# 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]

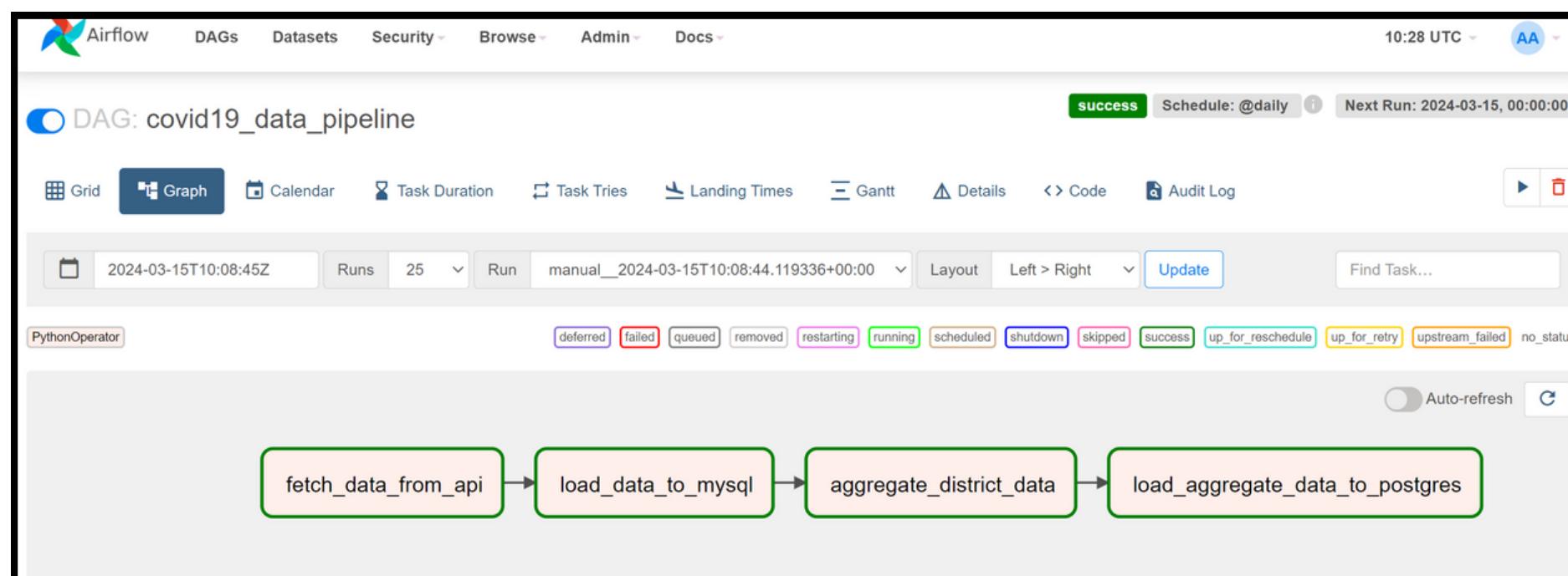
    # Connect to PostgreSQL database
    connection = psycopg2.connect(
        host='final_project_de-postgres-1',
        user='postgres',
        password='postgres',
        database='data_mart'
    )
    # Create a connection string
    connection_string = f'postgresql://postgres:postgres@final_project_de-postgres-1/data_mart'

    # 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



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

# Data Mart Output

## Output Table Province pada PostgreSQL

« T select \* from province\_daily pd | ↻ Enter a SQL expression to filter results (use ⌂)

Grid	123 id	123 province_id	⌚ date	123 total
1	1	32	2020-08-05	2,510
2	2	32	2020-08-05	1,317
3	3	32	2020-08-05	7,696
4	4	32	2020-08-05	163
5	5	32	2020-08-05	3,373
6	6	32	2020-08-05	0
7	7	32	2020-08-05	743
8	8	32	2020-08-05	5,270
9	9	32	2020-08-05	1,948
10	10	32	2020-08-05	7,594
11	11	32	2020-08-05	2,462
12	12	32	2020-08-05	1,098
13	13	32	2020-08-05	1,110
14	14	32	2020-08-05	1,224
15	15	32	2020-08-05	6,243
16	16	32	2020-08-05	5,075

« T select \* from province\_monthly pm | ↻ Enter a SQL expression to filter results (use ⌂)

Grid	123 id	123 province_id	123 month	123 total
1	1	32	8	3,508,645

province\_yearly 1 ×

« T select \* from province\_yearly py | ↻ Enter a SQL expression to filter results (use ⌂)

Grid	123 id	123 province_id	123 year	123 total
1	1	32	2,020	3,508,645

# Data Mart Output

## Output Table District pada PostgreSQL

**district\_monthly 1**

```
select * from district_monthly dm | Enter a SQL expression to filter results (use <=, >=, etc.)
```

Grid	123 id	123 district_id	123 month	123 total
1	1	3,204	8	108,500
2	2	3,217	8	51,380
3	3	3,216	8	301,210
4	4	3,201	8	191,590
5	5	3,207	8	119,105
6	6	3,203	8	48,965
7	7	3,209	8	29,400
8	8	3,205	8	186,620
9	9	3,212	8	85,190
10	10	3,215	8	281,645
11	11	3,208	8	90,265
12	12	3,210	8	43,715
13	13	3,218	8	39,725
14	14	3,214	8	47,075
15	15	3,213	8	223,685
...	...	...	...	...

**district\_yearly 1**

```
select * from district_yearly py | Enter a SQL expression to filter results (use <=, >=, etc.)
```

Grid	123 id	123 district_id	123 year	123 total
1	1	3,216	2,020	108,500
2	2	3,201	2,020	51,380
3	3	3,216	2,020	301,210
4	4	3,201	2,020	191,590
5	5	3,207	2,020	119,105
6	6	3,203	2,020	48,965
7	7	3,209	2,020	29,400
8	8	3,205	2,020	186,620
9	9	3,212	2,020	85,190
10	10	3,215	2,020	281,645
11	11	3,208	2,020	90,265
12	12	3,210	2,020	43,715
13	13	3,218	2,020	39,725
14	14	3,214	2,020	47,075
15	15	3,213	2,020	223,685
...	...	...	...	...



# Thank You!