A black and white logo

AI-generated content may be incorrect.**A close up of a logo

AI-generated content may be incorrect.**

**PT Dayamitra Telekomunikasi Tbk.**

Jalan Gatot Subroto Kav 52, Jakarta Selatan

www.mitratel.co.id | (+62) 2793 3363

**USER GUIDE PEMBUATAN DAGS APACHE AIRFLOW**

Prepared by:

**Muhammad Farras Majid**

Versi:

**1.0.0 (12 Juni 2025)**

Table of Contents

[**I.** **Pengenalan** 3](#_Toc200640452)

[**1.** **Latar Belakang** 3](#_Toc200640453)

[**2.** **Tujuan Panduan** 3](#_Toc200640454)

[**3.** **Penjelasan Apache Airflow** 3](#_Toc200640455)

[**4.** **Peran CDS dalam CDC** 4](#_Toc200640456)

[**5.** **Arsitektur Umum Apache Airflow** 4](#_Toc200640457)

[**II.** **Memulai Aplikasi** 5](#_Toc200640458)

[**A.** **Instalasi** 5](#_Toc200640459)

[**1.** **Kebutuhan Sistem** 5](#_Toc200640460)

[**2.** **Instalasi** 6](#_Toc200640461)

[**III.** **Pembuatan DAG File di VSCode** 7](#_Toc200640462)

[**A.** **Buat Baru File Python** 7](#_Toc200640463)

[**B.** **Pemasangan DAG pada Apache AirFlow** 8](#_Toc200640464)

[**C.** **Cara Mengoperasikan Apache AirFlow** 9](#_Toc200640465)

[**IV.** **Bantuan Lebih Lanjut** 12](#_Toc200640466)

[**A.** **Support** 12](#_Toc200640467)

[**1.** **Servicedesk Plus (Open Ticket)** 12](#_Toc200640468)

[**2.** **WhatsApp Helpdesk** 12](#_Toc200640469)

[**V.** **Lampiran Contoh Code** 13](#_Toc200640470)

[**A.** **Truncate** 13](#_Toc200640471)

[**B.** **Delta** 16](#_Toc200640472)

[**C.** **Truncate SAP** 21](#_Toc200640473)

[**D.** **Truncate DW** 30](#_Toc200640474)

[**E.** **Truncate DM** 35](#_Toc200640475)

## **Pengenalan**

### **Latar Belakang**

Dalam pengelolaan data modern, orkestrasi alur kerja (*workflow orchestration*) menjadi hal yang sangat penting untuk memastikan proses-proses berjalan secara terjadwal, dapat diulang, dan dapat dipantau. Apache Airflow menjadi salah satu tools open-source yang banyak digunakan untuk mengelola alur kerja data karena fleksibilitas, skalabilitas, dan dukungan komunitas yang kuat. Dalam lingkungan kerja yang dinamis, seperti integrasi data dari berbagai sumber, automasi pipeline ETL (Extract, Transform, Load), atau pemantauan proses bisnis, Airflow memegang peranan penting untuk mengorkestrasi tugas-tugas ini secara sistematis.

### **Tujuan Panduan**

Panduan ini disusun untuk memberikan pemahaman dan petunjuk teknis dalam:

* Mengenal konsep dasar Apache Airflow dan komponennya.
* Menjelaskan apa itu DAG dalam konteks Airflow.
* Memberikan panduan teknis dalam membuat dan menjalankan DAG.
* Menyajikan struktur arsitektur umum Airflow untuk memberikan gambaran bagaimana komponen-komponen saling berinteraksi.
* Menyediakan praktik terbaik dalam membuat DAG yang efisien, aman, dan dapat diskalakan.

Panduan ini ditujukan bagi engineer, data analyst, atau siapa pun yang baru mulai menggunakan Apache Airflow maupun yang ingin memperdalam pemahaman tentang cara kerja DAGs secara teknis.

### **Penjelasan Apache Airflow**

Apache Airflow adalah platform orkestrasi alur kerja berbasis Python yang digunakan untuk menjadwalkan dan memantau pipeline data. Airflow bekerja berdasarkan prinsip *Directed Acyclic Graphs* (DAGs), yang menggambarkan hubungan antar tugas (*tasks*) dan urutan eksekusinya.

Airflow menyediakan berbagai fitur, antara lain:

* Penjadwalan otomatis tugas.
* Dependency management antar tugas.
* Logging, pemantauan, dan retry otomatis.
* Dukungan plugin dan ekstensi untuk berbagai sistem data (MySQL, PostgreSQL, AWS, GCP, dll).

Airflow sangat ideal untuk pipeline ETL, data warehousing, machine learning workflow, dan proses batch lainnya.

### **Peran CDS dalam CDC**

DAG adalah singkatan dari **Directed Acyclic Graph**, yang dalam konteks Airflow adalah representasi grafis dari urutan eksekusi tugas-tugas (*tasks*) yang tidak memiliki siklus.

Karakteristik DAG dalam Airflow:

* **Directed**: Eksekusi tugas mengikuti arah panah (arah dependensi).
* **Acyclic**: Tidak boleh ada siklus atau lingkaran dalam dependensi antar tugas.
* **Graph**: Sekumpulan node (task) yang saling berhubungan.

DAG mendefinisikan:

* **Schedule interval**: Seberapa sering DAG dijalankan (misalnya, setiap hari jam 7 pagi).
* **Tasks**: Tugas-tugas individu, seperti memanggil API, menjalankan query SQL, memproses file, dll.
* **Dependencies**: Urutan antar-tugas, misalnya: extract\_data >> transform\_data >> load\_data.

Contoh sederhana DAG:

from airflow import DAG

from airflow.operators.python\_operator import PythonOperator

from datetime import datetime

def greet():

print("Hello from Airflow!")

with DAG(dag\_id="example\_dag", start\_date=datetime(2025, 1, 1), schedule\_interval="@daily", catchup=False) as dag:

greet\_task = PythonOperator(task\_id="greet", python\_callable=greet)

### **Arsitektur Umum Apache Airflow**

Apache Airflow terdiri dari beberapa komponen utama:

1. **Scheduler**
   * Bertanggung jawab menjadwalkan tugas berdasarkan DAG dan menentukan kapan eksekusi dilakukan.
2. **Web Server (UI)**
   * Antarmuka pengguna berbasis web untuk memantau, mengelola, dan menjalankan DAG.
3. **Worker**
   * Menjalankan tugas DAG. Airflow dapat menggunakan executor seperti CeleryExecutor, LocalExecutor, atau KubernetesExecutor.
4. **Metadata Database**
   * Menyimpan semua informasi mengenai DAG, task instance, log eksekusi, dan status.
5. **Executor**
   * Komponen yang menentukan bagaimana dan di mana tugas dijalankan (lokal, paralel, distributed).
6. **CLI / API**
   * Interface untuk menjalankan perintah Airflow melalui terminal atau REST API.

**Diagram Arsitektur Sederhana:**

A screenshot of a computer program

AI-generated content may be incorrect.

## **Memulai Aplikasi**

### **Instalasi**

### **Kebutuhan Sistem**

Sebelum memulai proses pengembangan Apache Airflo, beberapa perangkat lunak dan tools pengembangan perlu dipersiapkan untuk memastikan integrasi dan konektivitas ke sistem Data Warehouse berjalan dengan baik.

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Nama Aplikasi** | **Versi Minimum** | **Fungsi Utama** |
| 1 | **Mobaxterm** | Terbaru | UI untuk koneksi ke server Data Warehouse |
| 2 | **Visual Studio Code** | Terbaru | IDE yang dibutuhkan untuk menulis script DAGS, disarankan untuk menginstall plugin Python |

Table 1. Table kebutuhan aplikasi

### **Instalasi**

Bab ini memberikan panduan lengkap mengenai instalasi dan konfigurasi Mobaxterm, Java Development Kit (JDK), dan SAP GUI.

#### **Instalasi MobaXterm (Windows)**

**MobaXterm** adalah aplikasi terminal untuk Windows yang menyediakan banyak fungsi, termasuk SSH, SFTP, dan X11 forwarding, sangat berguna untuk bekerja dengan server Linux secara remote.

**Langkah-langkah:**

1. **Buka situs resmi:**
   1. Kunjungi: https://mobaxterm.mobatek.net/
2. **Unduh MobaXterm Home Edition (Free):**
   1. Klik tombol **“Download now”**.
   2. Pilih **Installer edition** (bukan Portable), agar bisa diinstal seperti aplikasi biasa.
   3. File yang diunduh akan bernama misalnya MobaXterm\_installer\_xx.x.exe.
3. **Instal MobaXterm:**
   1. Klik dua kali file .exe tersebut.
   2. Ikuti petunjuk instalasi (klik **Next**, lalu **Install**).
   3. Setelah selesai, klik **Finish**.
4. **Jalankan MobaXterm:**
   1. Buka dari Start Menu atau klik shortcut di desktop.
   2. MobaXterm siap digunakan untuk koneksi SSH/SFTP ke server remote.

#### **Instalasi Visual Studio Code (VS Code)**

**Visual Studio Code** adalah editor kode sumber ringan namun sangat powerful, mendukung banyak bahasa pemrograman termasuk Python, JavaScript, dan lainnya.

**Langkah-langkah:**

1. **Buka situs resmi:**
   1. Kunjungi: <https://code.visualstudio.com/>
2. **Unduh Installer:**
   1. Klik tombol **“Download for Windows”**.
   2. File akan bernama misalnya VSCodeUserSetup-x64-x.x.x.exe.
3. **Instal VS Code:**
   1. Klik dua kali file instalasi .exe.
   2. Ikuti wizard instalasi:
      1. Centang opsi “Add to PATH” (agar bisa dipanggil dari terminal/CMD).
      2. Klik **Next** sampai selesai, lalu **Install**.
   3. Klik **Finish** untuk menjalankan VS Code.
4. **(Opsional) Instal Extension:**
   1. Buka tab **Extensions** (ikon kotak di sidebar kiri).
   2. Cari dan instal extension yang dibutuhkan seperti:
      1. Python (by Microsoft)
      2. Remote - SSH (untuk mengedit file di server remote)
      3. Prettier (untuk auto-formatting)
      4. Docker, GitLens, dll sesuai kebutuhan.

## **Pembuatan DAG File di VSCode**

Bab ini membahas tahapan inti dalam proses pembuatan DAG menggunakan VSCode.

### **Buat Baru File Python**

Buat file baru di VSCode dengan extension .py

1. Buka VSCode → File → New File…

A screenshot of a computer program

AI-generated content may be incorrect.

1. Inputkan nama baru untuk DAG anda dengan extension .py

A screenshot of a computer

AI-generated content may be incorrect.

1. Buat file anda di direktori kerja anda, klik Create File

A screenshot of a computer

AI-generated content may be incorrect.

1. Anda akan menerima file script baru yang siap untuk dituliskan code python

A screenshot of a computer

AI-generated content may be incorrect.

1. Untuk referensi contoh code yang sudah ada, bisa merujuk pada lampiran di akhir User Guide untuk masing-masing kategori jenis penarikan.\
2. Untuk menyimpan script yang telah anda buat, klik Ctrl + S

### **Pemasangan DAG pada Apache AirFlow**

Untuk memasang DAG pada Apache AirFlow, anda membutuhkan akses server dwhapps.mitratel.co.id dengan menggunakan Mobaxterm, credentials bisa diberikan oleh tim IT Operation.

1. Di mobaxterm, pada server dwhapps, pada SFTP, navigasi ke folder /root/airflow/dags

A screenshot of a computer

AI-generated content may be incorrect.

1. Drag file script DAG yang telah anda buat ke folder tersebut, kemudian tunggu sampai file berhasil diupload
2. Pada command bash, inputkan 4 baris perintah berikut:

kill $(ps aux | grep 'airflow' | awk '{print $2}')

airflow kerberos -D

airflow scheduler -D

airflow webserver -D

1. Kemudian buka browser web, masukkan url <https://dwhapps.mitratel.co.id:8080>. Apabila diminta user password, masukkan username **airflow** password **airflow**.
2. DAG yang anda buat akan muncul dengan status **Non-Aktif**, anda bisa mengaktifkan DAG yang anda buat dengan mengklik tombol switch di sebelah kiri

A screenshot of a computer

AI-generated content may be incorrect.

### **Cara Mengoperasikan Apache AirFlow**

Untuk menggunakan Apache AirFlow sebagai interface Datawarehouse, buka browser web dengan URL <https://dwhapps.mitratel.co.id:8080/home>

1. Buka halaman utama AirFlow

A screenshot of a computer

AI-generated content may be incorrect.

1. Untuk Unpause DAG, klik tombol switch warna biru di sebelah kiri DAG yang anda buat

A screenshot of a computer

AI-generated content may be incorrect. A screenshot of a computer

AI-generated content may be incorrect.

1. Status history DAG bisa dilihat di kolom **Runs** , Hijau Tua menandakan Run Berhasil, Hijau Muda menandakan Run On Progress, Merah menandakan Run Failed

A screenshot of a number

AI-generated content may be incorrect.

1. Schedule bisa dilihat di kolom **Schedule**, mengikuti format Cronjob

A screenshot of a schedule

AI-generated content may be incorrect.

1. Action bar terdiri dari **Trigger DAG, Reparse DAG, dan Delete DAG**

A screenshot of a phone

AI-generated content may be incorrect.

1. Kemudian klik salah satu detail DAG, maka akan tampil halaman sebagai berikut:

A screenshot of a computer

AI-generated content may be incorrect.

1. Untuk melihat historikal log dari Run yang Failed bisa klik pada kotak di sebelah kiri dari masing-masing step

A screenshot of a computer

AI-generated content may be incorrect.

1. Untuk menjalankan DAG secara manual (**Trigger DAG**), bisa klik pada tombol kanan atas pada action bar.

A screenshot of a computer

AI-generated content may be incorrect.

1. Apabila anda telah melakukan perubahan script, kemudian anda ingin menjalankan ulang script DAG, lakukan **Reparse DAG** terlebih dahulu sebelum melakukan Trigger DAG.

A screenshot of a computer

AI-generated content may be incorrect.

1. Apabila anda ingin menghapus DAG yang telah anda buat, bisa klik pada **Delete DAG**

A screenshot of a computer

AI-generated content may be incorrect.

## **Bantuan Lebih Lanjut**

Apabila anda membutuhkan bantuan lebih lanjut terkait HCM OneFlux, atau mengalami kendala saat menggunakan aplikasi, bisa menghubungi tim HCM atau IT dibawah ini

## **Support**

## **Servicedesk Plus (Open Ticket)**

Bisa melakukan submit ticket help support ke website <https://servicedeck.mitratel.co.id>

## **WhatsApp Helpdesk**

Untuk menghubungi admin secara langsung bisa menghubungi Helpdesk HCM di 0812-2356-9040 atau Helpdesk IT di 0811-9799-556

## **Lampiran Contoh Code**

## **Truncate**

import json

import pandas as pd

import random

from airflow import DAG

from datetime import datetime, timedelta

from airflow.operators.python import PythonOperator

from airflow.providers.mysql.hooks.mysql import MySqlHook

from airflow.providers.postgres.hooks.postgres import PostgresHook

from airflow\_clickhouse\_plugin.hooks.clickhouse import ClickHouseHook

MYSQL\_CONN\_ID = "db\_oneflux\_tms"

CLICKHOUSE\_CONN\_ID = "clickhouse\_mitratel"

MYSQL\_DATABASE = "db\_tms"

MYSQL\_TABLE = "tabBAST 1"

CLICKHOUSE\_DATABASE = "spmk"

CLICKHOUSE\_TABLE = "tabbast1"

FILE\_PATH = "/tmp/debug\_DL\_spmk\_tabbast1.json"

DAG\_ID = "DL\_spmk\_tabbast1"

DAG\_INTERVAL = "0 17 \* \* \*"

CHUNK\_SIZE = 5000

LOG\_CONN\_ID = "airflow\_logs\_mitratel"

LOG\_TABLE = "airflow\_logs"

LOG\_TYPE = "truncate"

LOG\_KATEGORI = "Data Lake"

default\_args = {

    'owner': 'airflow',

    'depends\_on\_past': False,

    'start\_date': datetime(2024, 2, 1, 0, 0),

    # 'retries': 1,

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

}

dag = DAG(

    dag\_id=DAG\_ID,

    schedule\_interval= DAG\_INTERVAL,

    default\_args=default\_args,

    catchup=False

)

def log\_status(process\_name, mark, status, error\_message=None):

    """Insert or update the log table."""

    pg\_hook = PostgresHook(postgres\_conn\_id=LOG\_CONN\_ID)

    conn = pg\_hook.get\_conn()

    cursor = conn.cursor()

    now = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

    dag\_name = f"{CLICKHOUSE\_DATABASE}.{CLICKHOUSE\_TABLE}"

    if status == "pending":

        cursor.execute(

            f"""

            INSERT INTO {LOG\_TABLE} (process\_name, dag\_name, type, status, start\_time, mark, kategori)

            VALUES (%s, %s, %s, %s, %s, %s, %s)

            """, (process\_name, dag\_name, LOG\_TYPE, status, now, mark, LOG\_KATEGORI)

        )

    else:

        cursor.execute(

            f"""

            UPDATE {LOG\_TABLE} SET status = %s, end\_time = %s, error\_message = %s

            WHERE process\_name = %s AND status = 'pending' AND dag\_name = %s AND mark = %s AND kategori = %s

            """, (status, now, error\_message, process\_name, dag\_name, mark, LOG\_KATEGORI)

        )

    conn.commit()

    cursor.close()

    conn.close()

def extract\_from\_mysql(\*\*kwargs):

    """Extract data from MySQL and store it as a DataFrame."""

    random\_value = random.randint(1000, 9999)  # Angka acak 4 digit

    timestamp = datetime.now().strftime("%Y%m%d%H%M%S")  # Format YYYYMMDDHHMMSS

    RANDOM\_VALUE = f"{random\_value}\_{timestamp}"

    process\_name = "extract\_mysql"

    log\_status(process\_name, RANDOM\_VALUE, "pending")

    try:

        mysql\_hook = MySqlHook(mysql\_conn\_id=MYSQL\_CONN\_ID)

        sql = f"SELECT \* FROM `{MYSQL\_DATABASE}`.`{MYSQL\_TABLE}`"

        engine = mysql\_hook.get\_sqlalchemy\_engine()

        chunks = []

        with engine.connect() as connection:

            result = connection.execution\_options(stream\_results=True).execute(sql)

            while True:

                rows = result.fetchmany(CHUNK\_SIZE)

                if not rows:

                    break

                df\_chunk = pd.DataFrame(rows, columns=result.keys())

                chunks.append(df\_chunk)

        df = pd.concat(chunks, ignore\_index=True).astype(str)

        df.to\_json(FILE\_PATH, orient="records", default\_handler=str)

        kwargs['ti'].xcom\_push(key="file\_path", value=FILE\_PATH)

        kwargs['ti'].xcom\_push(key="random\_value", value=RANDOM\_VALUE)

        log\_status(process\_name, RANDOM\_VALUE, "success")

    except Exception as e:

        log\_status(process\_name, RANDOM\_VALUE, "failed", str(e))

        raise

def load\_to\_clickhouse(\*\*kwargs):

    """Load extracted data into ClickHouse."""

    random\_value = kwargs['ti'].xcom\_pull(task\_ids="extract\_mysql", key="random\_value")

    process\_name = "load\_clickhouse"

    log\_status(process\_name, random\_value, "pending")

    try:

        ti = kwargs['ti']

        file\_path = ti.xcom\_pull(task\_ids="extract\_mysql", key="file\_path")

        if not file\_path:

            raise ValueError("No file path received from MySQL extraction.")

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

            data\_dict = json.load(f)

        df = pd.DataFrame(data\_dict, dtype=str)

        df = df.applymap(lambda x: x.replace("\\", "") if isinstance(x, str) else x)

        df = df.applymap(lambda x: x.replace("'", "\\'") if isinstance(x, str) else x)

        hook = ClickHouseHook(clickhouse\_conn\_id=CLICKHOUSE\_CONN\_ID)

        hook.execute(f"TRUNCATE TABLE `{CLICKHOUSE\_DATABASE}`.`{CLICKHOUSE\_TABLE}`")

        insert\_query = f"INSERT INTO `{CLICKHOUSE\_DATABASE}`.`{CLICKHOUSE\_TABLE}` VALUES"

        values = ', '.join(

            f"""({', '.join(f"'{x}'" for x in row)})"""

            for row in df.itertuples(index=False, name=None)

        )

        hook.execute(insert\_query + values)

        log\_status(process\_name, random\_value, "success")

    except Exception as e:

        log\_status(process\_name, random\_value, "failed", str(e))

        raise

extract\_mysql = PythonOperator(

    task\_id="extract\_mysql",

    python\_callable=extract\_from\_mysql,

    do\_xcom\_push=False,

    dag=dag,

)

load\_clickhouse = PythonOperator(

    task\_id="load\_clickhouse",

    python\_callable=load\_to\_clickhouse,

    do\_xcom\_push=False,

    provide\_context=True,

    dag=dag,

)

extract\_mysql >> load\_clickhouse

## **Delta**

import json

import pandas as pd

import random

from airflow import DAG

from datetime import datetime, timedelta

from airflow.operators.python import PythonOperator

from airflow.utils.task\_group import TaskGroup

from airflow.providers.mysql.hooks.mysql import MySqlHook

from airflow.providers.postgres.hooks.postgres import PostgresHook

from airflow\_clickhouse\_plugin.hooks.clickhouse import ClickHouseHook

MYSQL\_CONN\_ID = "db\_oneflux\_tms"

CLICKHOUSE\_CONN\_ID = "clickhouse\_mitratel"

MYSQL\_DATABASE = "db\_tms"

MYSQL\_TABLE = "tabSite Visit Report"

CLICKHOUSE\_DATABASE = "tms"

CLICKHOUSE\_TABLE = "tabsitevisitreport"

FILE\_PATH = "/tmp/batch\_report\_"

DAG\_ID = "DL\_tms\_tabsitevisitreport"

DAG\_INTERVAL = "1-59/5 \* \* \* \*"

CHUNK\_SIZE = 500

BATCH\_SIZE = 1500

BATCH\_NO = 5

LOG\_CONN\_ID = "airflow\_logs\_mitratel"

LOG\_TABLE = "airflow\_logs"

LOG\_TYPE = "delta"

LOG\_KATEGORI = "Data Lake"

default\_args = {

    'owner': 'airflow',

    'depends\_on\_past': False,

    'start\_date': datetime(2024, 2, 1, 19, 5),

}

dag = DAG(

    dag\_id=DAG\_ID,

    schedule\_interval= DAG\_INTERVAL,

    default\_args=default\_args,

    catchup=False

)

def log\_status(process\_name, mark, status, error\_message=None):

    """Insert or update the log table."""

    pg\_hook = PostgresHook(postgres\_conn\_id=LOG\_CONN\_ID)

    conn = pg\_hook.get\_conn()

    cursor = conn.cursor()

    now = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

    dag\_name = f"{CLICKHOUSE\_DATABASE}.{CLICKHOUSE\_TABLE}"

    if status == "pending":

        cursor.execute(

            f"""

            INSERT INTO {LOG\_TABLE} (process\_name, dag\_name, type, status, start\_time, mark, kategori)

            VALUES (%s, %s, %s, %s, %s, %s, %s)

            """, (process\_name, dag\_name, LOG\_TYPE, status, now, mark, LOG\_KATEGORI)

        )

    else:

        cursor.execute(

            f"""

            UPDATE {LOG\_TABLE} SET status = %s, end\_time = %s, error\_message = %s

            WHERE process\_name = %s AND status = 'pending' AND dag\_name = %s AND mark = %s AND kategori = %s

            """, (status, now, error\_message, process\_name, dag\_name, mark, LOG\_KATEGORI)

        )

    conn.commit()

    cursor.close()

    conn.close()

def get\_latest\_modified(\*\*kwargs):

    process\_name = "get\_latest\_modified"

    random\_value = random.randint(1000, 9999)  # Angka acak 4 digit

    timestamp = datetime.now().strftime("%Y%m%d%H%M%S")  # Format YYYYMMDDHHMMSS

    RANDOM\_VALUE = f"{random\_value}\_{timestamp}"

    log\_status(process\_name, RANDOM\_VALUE, "pending")

    try:

        hook = ClickHouseHook(clickhouse\_conn\_id=CLICKHOUSE\_CONN\_ID)

        sql = f"SELECT MAX(modified) FROM `{CLICKHOUSE\_DATABASE}`.`{CLICKHOUSE\_TABLE}`"

        result = hook.execute(sql)

        latest\_modified = result[0][0] if result and result[0] and result[0][0] else "1900-01-01 00:00:00"

        ti = kwargs['ti']

        ti.xcom\_push(key='latest\_modified', value=latest\_modified)

        del hook

        kwargs['ti'].xcom\_push(key="random\_value", value=RANDOM\_VALUE)

        log\_status(process\_name, RANDOM\_VALUE, "success")

    except Exception as e:

        log\_status(process\_name, RANDOM\_VALUE, "failed", str(e))

        raise

def count\_total\_rows(\*\*kwargs):

    random\_value = kwargs['ti'].xcom\_pull(task\_ids="get\_latest\_modified", key="random\_value")

    process\_name = "count\_total\_rows"

    log\_status(process\_name, random\_value, "pending")

    try:

        ti = kwargs['ti']

        latest\_modified = ti.xcom\_pull(task\_ids='get\_latest\_modified', key='latest\_modified')

        mysql\_hook = MySqlHook(mysql\_conn\_id=MYSQL\_CONN\_ID)

        sql = f"SELECT COUNT(\*) FROM `{MYSQL\_DATABASE}`.`{MYSQL\_TABLE}` WHERE modified > '{latest\_modified}'"

        total\_rows = mysql\_hook.get\_first(sql)[0]

        ti.xcom\_push(key='total\_rows', value=total\_rows)

        del mysql\_hook

        log\_status(process\_name, random\_value, "success")

    except Exception as e:

        log\_status(process\_name, random\_value, "failed", str(e))

        raise

def extract\_batch(batch\_number, \*\*kwargs):

    random\_value = kwargs['ti'].xcom\_pull(task\_ids="get\_latest\_modified", key="random\_value")

    process\_name = f"extract\_batch\_{batch\_number}"

    log\_status(process\_name, random\_value, "pending")

    mysql\_hook = MySqlHook(mysql\_conn\_id=MYSQL\_CONN\_ID, connect\_args={"connect\_timeout": 600000, "autocommit": True})

    engine = mysql\_hook.get\_sqlalchemy\_engine()

    try:

        ti = kwargs['ti']

        total\_rows = ti.xcom\_pull(task\_ids='count\_rows', key='total\_rows')

        latest\_modified = ti.xcom\_pull(task\_ids='get\_latest\_modified', key='latest\_modified')

        offset = batch\_number \* BATCH\_SIZE

        if offset >= total\_rows:

            # Tetap buat file JSON kosong jika batch ini tidak diperlukan

            empty\_path = f"{FILE\_PATH}{batch\_number}.json"

            with open(empty\_path, "w") as f:

                json.dump([], f)  # Menyimpan daftar kosong sebagai JSON

            ti.xcom\_push(key=f"file\_path\_{batch\_number}", value=empty\_path)

            log\_status(process\_name, random\_value, "success")

            return

        sql = f"""

            SELECT \* FROM `{MYSQL\_DATABASE}`.`{MYSQL\_TABLE}`

            WHERE modified > '{latest\_modified}'

            ORDER BY modified ASC

            LIMIT {BATCH\_SIZE} OFFSET {offset}

        """

        chunks = []

        chunk\_size = CHUNK\_SIZE  # Ambil 5000 baris per batch

        with engine.connect() as connection:

            result = connection.execution\_options(stream\_results=True).execute(sql)

            while True:

                rows = result.fetchmany(chunk\_size)  # Ambil batch

                if not rows:

                    break  # Keluar jika tidak ada data lagi

                df\_chunk = pd.DataFrame(rows, columns=result.keys())  # Konversi ke DataFrame

                chunks.append(df\_chunk)  # Simpan batch

        if chunks:

            df = pd.concat(chunks, ignore\_index=True).astype(str)

        else:

            df = pd.DataFrame()  # DataFrame kosong jika tidak ada data

        df.to\_json(f"{FILE\_PATH}{batch\_number}.json", orient="records", default\_handler=str)

        ti.xcom\_push(key=f"file\_path\_{batch\_number}", value=f"{FILE\_PATH}{batch\_number}.json")

        log\_status(process\_name, random\_value, "success")

    except Exception as e:

        log\_status(process\_name, random\_value, "failed", str(e))

        raise RuntimeError(f"Error during SQL execution: {e}")

    finally:

        engine.dispose()  # Pastikan koneksi ditutup setelah digunakan

def load\_to\_clickhouse(\*\*kwargs):

    random\_value = kwargs['ti'].xcom\_pull(task\_ids="get\_latest\_modified", key="random\_value")

    process\_name = "load\_clickhouse"

    log\_status(process\_name, random\_value, "pending")

    try:

        hook = ClickHouseHook(clickhouse\_conn\_id=CLICKHOUSE\_CONN\_ID)

        for batch\_number in range(BATCH\_NO):

            file\_path = f"{FILE\_PATH}{batch\_number}.json"

            if not file\_path:

                log\_status(process\_name, random\_value, "failed", str(e))

                raise ValueError("No file path received from MySQL extraction.")

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

                data\_dict = json.load(f)

            if not data\_dict:

                continue

            # Convert to DataFrame with explicit dtype

            df = pd.DataFrame(data\_dict, dtype=str)

            df = df.applymap(lambda x: x.replace("\\", "") if isinstance(x, str) else x)

            df = df.applymap(lambda x: x.replace("'", "\\'") if isinstance(x, str) else x)

            hook = ClickHouseHook(clickhouse\_conn\_id=CLICKHOUSE\_CONN\_ID)

            insert\_query = f"INSERT INTO `{CLICKHOUSE\_DATABASE}`.`{CLICKHOUSE\_TABLE}` VALUES"

            values = ', '.join(

                f"""({', '.join(f"'{x}'" for x in row)})"""

                for row in df.itertuples(index=False, name=None)

            )

            hook.execute(insert\_query + values)

        log\_status(process\_name, random\_value, "success")

    except Exception as e:

        log\_status(process\_name, random\_value, "failed", str(e))

        raise

def create\_extraction\_tasks(dag):

    with TaskGroup("extraction\_tasks", dag=dag) as extraction\_tasks:

        for batch\_number in range(BATCH\_NO):

            PythonOperator(

                task\_id=f"extract\_batch\_{batch\_number}",

                python\_callable=extract\_batch,

                op\_kwargs={'batch\_number': batch\_number},

                provide\_context=True,

                retries=15,

                retry\_delay=timedelta(seconds=30),

                dag=dag,

            )

    return extraction\_tasks

get\_latest\_modified\_task = PythonOperator(

    task\_id="get\_latest\_modified",

    python\_callable=get\_latest\_modified,

    provide\_context=True,

    dag=dag

)

count\_rows\_task = PythonOperator(

    task\_id="count\_rows",

    python\_callable=count\_total\_rows,

    provide\_context=True,

    dag=dag

)

extraction\_tasks = create\_extraction\_tasks(dag)

load\_clickhouse\_task = PythonOperator(

    task\_id="load\_clickhouse",

    python\_callable=load\_to\_clickhouse,

    provide\_context=True,

    dag=dag

)

get\_latest\_modified\_task >> count\_rows\_task >> extraction\_tasks >> load\_clickhouse\_task

## **Truncate SAP**

from airflow import DAG

import random

import logging

from airflow.operators.python import PythonOperator, BranchPythonOperator

from datetime import datetime, timedelta

import requests

import os

import glob

from lxml import etree

from requests.auth import HTTPBasicAuth

from airflow.providers.postgres.hooks.postgres import PostgresHook

from airflow\_clickhouse\_plugin.hooks.clickhouse import ClickHouseHook

# Constants

BASE\_URL = 'http://dmterpqas.mitratel.co.id:8021/sap/opu/odata/sap/ZCDC\_AFIH\_1\_SRV/FactsOfZCDCAFIH?sap-client=200'

DELTA\_DISCOVERY\_URL = 'http://dmterpqas.mitratel.co.id:8021/sap/opu/odata/sap/ZCDC\_AFIH\_1\_SRV/DeltaLinksOfFactsOfZCDCAFIH?sap-client=200'

NEXT\_URL = 'http://dmterpqas.mitratel.co.id:8021/sap/opu/odata/sap/ZCDC\_AFIH\_1\_SRV/'

HEADERS = {

    'Accept': 'application/xml',

    'Prefer': 'odata.track-changes,odata.maxpagesize=5000'

}

USERNAME = 'xxxxxxxxx'

PASSWORD = ‘xxxxxxxxx'

DELTA\_LINK\_PATH = '/tmp/sap\_afih\_delta\_link.txt'

XML\_DIR = '/tmp/sap\_afih'

DAG\_ID = "DL\_sap\_afih"

DAG\_INTERVAL = "5 17 \* \* \*"

CLICKHOUSE\_CONN\_ID = "clickhouse\_mitratel"

CLICKHOUSE\_DATABASE = "sap"

CLICKHOUSE\_TABLE = "afih"

LOG\_CONN\_ID = "airflow\_logs\_mitratel"

LOG\_TABLE = "airflow\_logs"

LOG\_TYPE = "delta"

LOG\_KATEGORI = "Data Lake"

INSERT\_QUERY = """

                    INSERT INTO `sap`.`afih`

                    (`AUFNR`, `ARTPR`,`PRIOK`,`EQUNR`,`BAUTL`,`ILOAN`,`ILOAI`,`ANLZU`,`IWERK`,`INGPR`,`APGRP`,

                    `PMOBJTY`,`GEWRK`,`KUNUM`,`ANING`,`GAUZT`,`GAUEH`,`ANLBD`,`ANLVD`,`ANLBZ`,`ANLVZ`,

                    `INSPK`,`DATAN`,`WARPL`,`ABNUM`,`WAPOS`,`LAUFN`,`REVNR`,`ADDAT`,`ADUHR`,`IPHAS`,

                    `ILART`,`QMNUM`,`HISDA`,`AKKNZ`,`PLKNZ`,`SERIALNR`,`SERMAT`,`DEVICEID`,`SCREENTY`,

                    `ADPSP`,`RSUPG`,`CHANGEDDATETIME`,`MAINTORDPERSONRESPONSIBLE`,`MAINTORDOVRLPROCPHASE`,

                    `MAINTORDOVRLPROCSUBPHASE`,`DUMMYAFIHINCLEEWPS`,`OBJNR`,`MEQUI`,`MHIOADDATE`,`MHIOADTIME`,

                    `USERMODE`,`UII`,`DATAMODELVERSION`,`SERVICEDOCTYPE`,`SERVICEDOCID`,`SERVICEDOCITEMID`,

                    `ISBILLABLE`,`FLDLOGSDELIVISHELDONSHORE`,`ORCOD`,`PRMAN`,`PRVAL`,`TWTDE`,`MSTCK`,

                    `LACDDATE`,`OLDLACDDATE`,`ODQ\_CHANGEMODE`,`ODQ\_ENTITYCNTR`)

                    VALUES

                    (%(AUFNR)s, %(ARTPR)s, %(PRIOK)s, %(EQUNR)s, %(BAUTL)s, %(ILOAN)s, %(ILOAI)s,

                    %(ANLZU)s, %(IWERK)s, %(INGPR)s, %(APGRP)s, %(PMOBJTY)s, %(GEWRK)s, %(KUNUM)s,

                    %(ANING)s, %(GAUZT)s, %(GAUEH)s, %(ANLBD)s, %(ANLVD)s, %(ANLBZ)s, %(ANLVZ)s,

                    %(INSPK)s, %(DATAN)s, %(WARPL)s, %(ABNUM)s, %(WAPOS)s, %(LAUFN)s, %(REVNR)s,

                    %(ADDAT)s, %(ADUHR)s, %(IPHAS)s, %(ILART)s, %(QMNUM)s, %(HISDA)s, %(AKKNZ)s,

                    %(PLKNZ)s, %(SERIALNR)s, %(SERMAT)s, %(DEVICEID)s, %(SCREENTY)s, %(ADPSP)s,

                    %(RSUPG)s, %(CHANGEDDATETIME)s, %(MAINTORDPERSONRESPONSIBLE)s,

                    %(MAINTORDOVRLPROCPHASE)s, %(MAINTORDOVRLPROCSUBPHASE)s, %(DUMMYAFIHINCLEEWPS)s,

                    %(OBJNR)s, %(MEQUI)s, %(MHIOADDATE)s, %(MHIOADTIME)s, %(USERMODE)s, %(UII)s,

                    %(DATAMODELVERSION)s, %(SERVICEDOCTYPE)s, %(SERVICEDOCID)s, %(SERVICEDOCITEMID)s,

                    %(ISBILLABLE)s, %(FLDLOGSDELIVISHELDONSHORE)s, %(ORCOD)s, %(PRMAN)s, %(PRVAL)s,

                    %(TWTDE)s, %(MSTCK)s, %(LACDDATE)s, %(OLDLACDDATE)s, %(ODQ\_CHANGEMODE)s,

                    %(ODQ\_ENTITYCNTR)s)

                """

INSERT\_QUERY\_2 = """

                    INSERT INTO `sap`.`afih`

                    (`AUFNR`, `ARTPR`,`PRIOK`,`EQUNR`,`BAUTL`,`ILOAN`,`ILOAI`,`ANLZU`,`IWERK`,`INGPR`,`APGRP`,

                    `PMOBJTY`,`GEWRK`,`KUNUM`,`ANING`,`GAUZT`,`GAUEH`,`ANLBD`,`ANLVD`,`ANLBZ`,`ANLVZ`,

                    `INSPK`,`DATAN`,`WARPL`,`ABNUM`,`WAPOS`,`LAUFN`,`REVNR`,`ADDAT`,`ADUHR`,`IPHAS`,

                    `ILART`,`QMNUM`,`HISDA`,`AKKNZ`,`PLKNZ`,`SERIALNR`,`SERMAT`,`DEVICEID`,`SCREENTY`,

                    `ADPSP`,`RSUPG`,`CHANGEDDATETIME`,`MAINTORDPERSONRESPONSIBLE`,`MAINTORDOVRLPROCPHASE`,

                    `MAINTORDOVRLPROCSUBPHASE`,`DUMMYAFIHINCLEEWPS`,`OBJNR`,`MEQUI`,`MHIOADDATE`,`MHIOADTIME`,

                    `USERMODE`,`UII`,`DATAMODELVERSION`,`SERVICEDOCTYPE`,`SERVICEDOCID`,`SERVICEDOCITEMID`,

                    `ISBILLABLE`,`FLDLOGSDELIVISHELDONSHORE`,`ORCOD`,`PRMAN`,`PRVAL`,`TWTDE`,`MSTCK`,

                    `LACDDATE`,`OLDLACDDATE`,`ODQ\_CHANGEMODE`,`ODQ\_ENTITYCNTR`)

                    VALUES

                """

os.makedirs(XML\_DIR, exist\_ok=True)

def log\_status(process\_name, mark, status, error\_message=None):

    """Insert or update the log table."""

    pg\_hook = PostgresHook(postgres\_conn\_id=LOG\_CONN\_ID)

    conn = pg\_hook.get\_conn()

    cursor = conn.cursor()

    now = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

    dag\_name = f"{CLICKHOUSE\_DATABASE}.{CLICKHOUSE\_TABLE}"

    if status == "pending":

        cursor.execute(

            f"""

            INSERT INTO {LOG\_TABLE} (process\_name, dag\_name, type, status, start\_time, mark, kategori)

            VALUES (%s, %s, %s, %s, %s, %s, %s)

            """, (process\_name, dag\_name, LOG\_TYPE, status, now, mark, LOG\_KATEGORI)

        )

    else:

        cursor.execute(

            f"""

            UPDATE {LOG\_TABLE} SET status = %s, end\_time = %s, error\_message = %s

            WHERE process\_name = %s AND status = 'pending' AND dag\_name = %s AND mark = %s AND kategori = %s

            """, (status, now, error\_message, process\_name, dag\_name, mark, LOG\_KATEGORI)

        )

    conn.commit()

    cursor.close()

    conn.close()

def choose\_fetch\_mode(\*\*kwargs):

    random\_value = random.randint(1000, 9999)  # Angka acak 4 digit

    timestamp = datetime.now().strftime("%Y%m%d%H%M%S")  # Format YYYYMMDDHHMMSS

    RANDOM\_VALUE = f"{random\_value}\_{timestamp}"

    process\_name = "choose\_fetch\_mode"

    log\_status(process\_name, RANDOM\_VALUE, "pending")

    try:

        kwargs['ti'].xcom\_push(key="random\_value", value=RANDOM\_VALUE)

        log\_status(process\_name, RANDOM\_VALUE, "success")

    except Exception as e:

        log\_status(process\_name, RANDOM\_VALUE, "failed", str(e))

        raise

    return 'fetch\_sap\_delta' if os.path.exists(DELTA\_LINK\_PATH) else 'fetch\_sap\_cdc\_initial'

def fetch\_sap\_cdc\_initial(\*\*kwargs):

    random\_value = kwargs['ti'].xcom\_pull(task\_ids="branch\_initial\_or\_delta", key="random\_value")

    process\_name = "fetch\_sap\_cdc\_initial"

    log\_status(process\_name, random\_value, "pending")

    try:

        pattern = os.path.join(XML\_DIR, 'batch\*.xml')

        for file\_path in glob.glob(pattern):

            os.remove(file\_path)

        url = BASE\_URL + '&InitialLoad=true'

        idx = 0

        while url:

            response = requests.get(url, headers=HEADERS, auth=HTTPBasicAuth(USERNAME, PASSWORD))

            response.raise\_for\_status()

            file\_path = os.path.join(XML\_DIR, f'batch\_{idx}.xml')

            with open(file\_path, 'wb') as f:

                f.write(response.content)

            root = etree.fromstring(response.content)

            ns = {'atom': 'http://www.w3.org/2005/Atom'}

            next\_link = root.find("atom:link[@rel='next']", namespaces=ns)

            url = NEXT\_URL + next\_link.attrib['href'] if next\_link is not None else None

            idx += 1

        log\_status(process\_name, random\_value, "success")

    except Exception as e:

        log\_status(process\_name, random\_value, "failed", str(e))

        raise

def store\_initial\_delta\_link(\*\*kwargs):

    random\_value = kwargs['ti'].xcom\_pull(task\_ids="branch\_initial\_or\_delta", key="random\_value")

    process\_name = "store\_initial\_delta\_link"

    log\_status(process\_name, random\_value, "pending")

    try:

        response = requests.get(DELTA\_DISCOVERY\_URL, auth=HTTPBasicAuth(USERNAME, PASSWORD))

        response.raise\_for\_status()

        root = etree.fromstring(response.content)

        # Use namespaces for accurate XML parsing

        ns = {

            'atom': 'http://www.w3.org/2005/Atom',

            'm': 'http://schemas.microsoft.com/ado/2007/08/dataservices/metadata',

            'd': 'http://schemas.microsoft.com/ado/2007/08/dataservices'

        }

        changes\_after\_href = None

        delta\_token = None

        for entry in root.findall('atom:entry', namespaces=ns):

            # Find link containing "ChangesAfter"

            for link in entry.findall('atom:link', namespaces=ns):

                href = link.attrib.get('href', '')

                if 'ChangesAfter' in href:

                    changes\_after\_href = href

                    break

            # Get the DeltaToken

            props = entry.find('.//m:properties', namespaces=ns)

            if props is not None:

                token\_elem = props.find('d:DeltaToken', namespaces=ns)

                if token\_elem is not None:

                    delta\_token = token\_elem.text

            if changes\_after\_href and delta\_token:

                break  # Found everything we need

        if not (changes\_after\_href and delta\_token):

            log\_status(process\_name, random\_value, "failed", "Missing 'ChangesAfter' link or DeltaToken in delta discovery response.")

            raise Exception("Missing 'ChangesAfter' link or DeltaToken in delta discovery response.")

        # Write both parts to file

        with open(DELTA\_LINK\_PATH, 'w') as f:

            f.write(f"{changes\_after\_href}?sap-client=200&!deltatoken='{delta\_token}'")

        log\_status(process\_name, random\_value, "success")

    except Exception as e:

        log\_status(process\_name, random\_value, "failed", str(e))

        raise

def fetch\_sap\_delta(\*\*kwargs):

    random\_value = kwargs['ti'].xcom\_pull(task\_ids="branch\_initial\_or\_delta", key="random\_value")

    process\_name = "fetch\_sap\_delta"

    log\_status(process\_name, random\_value, "pending")

    try:

        pattern = os.path.join(XML\_DIR, 'delta\*.xml')

        for file\_path in glob.glob(pattern):

            os.remove(file\_path)

        with open(DELTA\_LINK\_PATH, 'r') as f:

            url = NEXT\_URL + f.read()

        idx = 0

        while url:

            response = requests.get(url, auth=HTTPBasicAuth(USERNAME, PASSWORD))

            response.raise\_for\_status()

            file\_path = os.path.join(XML\_DIR, f'delta\_{idx}.xml')

            with open(file\_path, 'wb') as f:

                f.write(response.content)

            root = etree.fromstring(response.content)

            ns = {'atom': 'http://www.w3.org/2005/Atom'}

            next\_link = root.find("atom:link[@rel='delta']", namespaces=ns)

            if next\_link is not None:

                with open(DELTA\_LINK\_PATH, 'w') as f:

                    u = next\_link.attrib['href']

                    if "?!deltatoken=" in u:

                        u = u.replace("?!deltatoken=", "&!deltatoken=", 1)

                    f.write(u)

                break

            next\_link = root.find("atom:link[@rel='next']", namespaces=ns)

            url = NEXT\_URL + next\_link.attrib['href'] if next\_link is not None else None

            idx += 1

        log\_status(process\_name, random\_value, "success")

    except Exception as e:

        log\_status(process\_name, random\_value, "failed", str(e))

        raise

def parse\_xml\_init():

    ns = {

        'atom': 'http://www.w3.org/2005/Atom',

        'm': 'http://schemas.microsoft.com/ado/2007/08/dataservices/metadata',

        'd': 'http://schemas.microsoft.com/ado/2007/08/dataservices'

    }

    records = []

    for filename in os.listdir(XML\_DIR):

        if filename.startswith("batch") and filename.endswith(".xml"):

            with open(os.path.join(XML\_DIR, filename), 'rb') as f:

                root = etree.parse(f).getroot()

                for entry in root.findall('atom:entry', namespaces=ns):

                    content = entry.find('atom:content', namespaces=ns)

                    properties = content.find('m:properties', namespaces=ns)

                    record = {elem.tag.split('}')[1]: elem.text for elem in properties}

                    record['ODQ\_CHANGEMODE'] = record.get('ODQ\_CHANGEMODE', 'I')

                    records.append(record)

    return records

def parse\_xml():

    ns = {

        'atom': 'http://www.w3.org/2005/Atom',

        'm': 'http://schemas.microsoft.com/ado/2007/08/dataservices/metadata',

        'd': 'http://schemas.microsoft.com/ado/2007/08/dataservices'

    }

    records = []

    for filename in os.listdir(XML\_DIR):

        if filename.startswith("delta") and filename.endswith(".xml"):

            with open(os.path.join(XML\_DIR, filename), 'rb') as f:

                root = etree.parse(f).getroot()

                for entry in root.findall('atom:entry', namespaces=ns):

                    content = entry.find('atom:content', namespaces=ns)

                    properties = content.find('m:properties', namespaces=ns)

                    record = {elem.tag.split('}')[1]: elem.text for elem in properties}

                    record['ODQ\_CHANGEMODE'] = record.get('ODQ\_CHANGEMODE', 'I')

                    records.append(record)

    return records

def load\_to\_clickhouse\_init(\*\*kwargs):

    random\_value = kwargs['ti'].xcom\_pull(task\_ids="branch\_initial\_or\_delta", key="random\_value")

    process\_name = "load\_to\_clickhouse\_init"

    log\_status(process\_name, random\_value, "pending")

    try:

        client = ClickHouseHook(clickhouse\_conn\_id=CLICKHOUSE\_CONN\_ID)

        records = parse\_xml\_init()

        # logging.warning(INSERT\_QUERY, records[0])

        # for r in records:

        #     if r['ODQ\_CHANGEMODE'] == 'D':

        #         aufnr = r['AUFNR']

        #         DELETE\_QUERY =  f"ALTER TABLE `sap`.`afko` DELETE  WHERE AUFNR = '{aufnr}'"

        #         client.execute(DELETE\_QUERY)

        #     else:

        #         client.execute(INSERT\_QUERY, r)

        client.execute(INSERT\_QUERY\_2, records)

        log\_status(process\_name, random\_value, "success")

    except Exception as e:

        log\_status(process\_name, random\_value, "failed", str(e))

        raise

def load\_to\_clickhouse(\*\*kwargs):

    random\_value = kwargs['ti'].xcom\_pull(task\_ids="branch\_initial\_or\_delta", key="random\_value")

    process\_name = "load\_to\_clickhouse"

    log\_status(process\_name, random\_value, "pending")

    try:

        client = ClickHouseHook(clickhouse\_conn\_id=CLICKHOUSE\_CONN\_ID)

        records = parse\_xml()

        # logging.warning(INSERT\_QUERY, records[0])

        for r in records:

            if r['ODQ\_CHANGEMODE'] == 'D':

                aufnr = r['AUFNR']

                DELETE\_QUERY =  f"ALTER TABLE `sap`.`afih` DELETE  WHERE AUFNR = '{aufnr}'"

                client.execute(DELETE\_QUERY)

            else:

                client.execute(INSERT\_QUERY, r)

        log\_status(process\_name, random\_value, "success")

    except Exception as e:

        log\_status(process\_name, random\_value, "failed", str(e))

        raise

with DAG(

    dag\_id=DAG\_ID,

    start\_date=datetime(2024, 1, 1),

    schedule\_interval= DAG\_INTERVAL,

    catchup=False,

    default\_args={

        'owner': 'airflow',

        # 'retries': 1,

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

    }

) as dag:

    branch = BranchPythonOperator(

        task\_id='branch\_initial\_or\_delta',

        python\_callable=choose\_fetch\_mode

    )

    initial\_task = PythonOperator(

        task\_id='fetch\_sap\_cdc\_initial',

        python\_callable=fetch\_sap\_cdc\_initial

    )

    store\_delta\_link = PythonOperator(

        task\_id='store\_initial\_delta\_link',

        python\_callable=store\_initial\_delta\_link

    )

    fetch\_delta\_task = PythonOperator(

        task\_id='fetch\_sap\_delta',

        python\_callable=fetch\_sap\_delta

    )

    parse\_task = PythonOperator(

        task\_id='parse\_and\_load',

        python\_callable=load\_to\_clickhouse

    )

    parse\_task\_init = PythonOperator(

        task\_id='parse\_and\_load\_init',

        python\_callable=load\_to\_clickhouse\_init

    )

    branch >> initial\_task >> store\_delta\_link >> parse\_task\_init

    branch >> fetch\_delta\_task >> parse\_task

## **Truncate DW**

import json

import pandas as pd

import random

import logging

from airflow import DAG

from datetime import datetime, timedelta

from airflow.operators.python import PythonOperator

from airflow.providers.mysql.hooks.mysql import MySqlHook

from airflow.providers.postgres.hooks.postgres import PostgresHook

from airflow\_clickhouse\_plugin.hooks.clickhouse import ClickHouseHook

SOURCE\_CONN\_ID = "clickhouse\_mitratel"

SOURCE\_DATABASE = "tms"

SOURCE\_TABLE = ""

TARGET\_CONN\_ID = "clickhouse\_dwh\_mitratel"

TARGET\_DATABASE = "dwh\_spmk"

TARGET\_TABLE = "fact\_spmk"

FILE\_PATH = "/tmp/debug\_DWH\_spmk\_factspmk.json"

DAG\_ID = "DWH\_spmk\_factspmk"

DAG\_INTERVAL = "32 17 \* \* \*"

CHUNK\_SIZE = 5000

LOG\_CONN\_ID = "airflow\_logs\_mitratel"

LOG\_TABLE = "airflow\_logs"

LOG\_TYPE = "truncate"

LOG\_KATEGORI = "Data WareHouse"

EXTRACT\_QUERY = """

SELECT DISTINCT

    a.parent AS wo\_number,

    a.project\_id,

    a.area,

    a.region,

    a.province,

    b.parent AS spmk\_number,

    b.is\_cancel AS is\_spmk\_cancel,

    bb.workflow\_state AS spmk\_status,

    c.name AS task\_number,

    c.subject AS task\_desc,

    c.type AS task\_type,

    c.status AS task\_status,

    c.bast\_document,

    d.name AS boq,

    d.workflow\_state AS boq\_status,

    d.mitra,

    d.scope\_of\_work,

    e.name AS bast\_name,

    e.nomor\_bast,

    e.workflow\_state AS bast\_status,

    e.pekerjaan,

    f.parent AS po\_number,

    ff.workflow\_state AS po\_status,

    g.gr\_sap\_status

    FROM (

        SELECT parent, project\_id, area, region, province FROM spmk.tabwoandpid

        UNION ALL

        SELECT parent, project\_id, area, region, province FROM spmk.tabnewwo

        UNION ALL

        SELECT parent, project\_id, area, region, province FROM spmk.tabwositelistcolo

    ) a

    LEFT JOIN spmk.tabspmkchild b

        ON a.project\_id = b.project\_id

    LEFT JOIN spmk.tabspmk bb

        ON b.parent = bb.spmk\_number

    LEFT JOIN spmk.tabtask c

        ON a.project\_id = c.project

    LEFT JOIN spmk.tabboqactual d

        ON a.project\_id = d.project\_id

        AND (

            (d.scope\_of\_work LIKE '%CME%' AND c.subject LIKE '%CME%')

            OR

            (d.scope\_of\_work LIKE '%Perkuatan%' AND c.subject LIKE '%Perkuatan%')

        )

    LEFT JOIN spmk.tabbast1 e

        ON a.project\_id = e.project\_id

        AND d.scope\_of\_work = e.pekerjaan

    LEFT JOIN spmk.tabpurchaseorderitem f

        ON a.project\_id = f.project

        AND e.purchase\_orderpo\_no = f.parent

    LEFT JOIN spmk.tabpurchaseorder ff

        ON ff.name = e.purchase\_orderpo\_no

    LEFT JOIN spmk.tabgoodreceipt g

        ON a.project\_id = g.project\_id

        AND e.purchase\_orderpo\_no = g.po\_number

        AND g.gr\_sap\_status = 'Success' AND g.gr\_sap\_number IS NOT NULL

    WHERE

        (g.gr\_sap\_status IS NULL OR TRIM(g.gr\_sap\_status) <> '')

        AND a.project\_id LIKE '24%'

        AND c.subject LIKE 'BAST%'

        AND (c.subject LIKE '%CME%' OR c.subject LIKE '%Perkuatan%')

"""

SOURCE\_COLUMNS=['wo\_number', 'project\_id', 'area', 'region',

                'province', 'spmk\_number', 'is\_spmk\_cancel',

                'spmk\_status', 'task\_number', 'task\_desc',

                'task\_type', 'task\_status', 'bast\_document',

                'boq', 'boq\_status', 'mitra', 'scope\_of\_work',

                'bast\_name', 'nomor\_bast', 'bast\_status', 'pekerjaan',

                'po\_number', 'po\_status', 'gr\_sap\_status', 'updated\_at']

default\_args = {

    'owner': 'airflow',

    'depends\_on\_past': False,

    'start\_date': datetime(2024, 2, 1, 0, 0),

    # 'retries': 1,

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

}

dag = DAG(

    dag\_id=DAG\_ID,

    schedule\_interval= DAG\_INTERVAL,

    default\_args=default\_args,

    catchup=False

)

def log\_status(process\_name, mark, status, error\_message=None):

    """Insert or update the log table."""

    pg\_hook = PostgresHook(postgres\_conn\_id=LOG\_CONN\_ID)

    conn = pg\_hook.get\_conn()

    cursor = conn.cursor()

    now = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

    dag\_name = f"{TARGET\_DATABASE}.{TARGET\_TABLE}"

    if status == "pending":

        cursor.execute(

            f"""

            INSERT INTO {LOG\_TABLE} (process\_name, dag\_name, type, status, start\_time, mark, kategori)

            VALUES (%s, %s, %s, %s, %s, %s, %s)

            """, (process\_name, dag\_name, LOG\_TYPE, status, now, mark, LOG\_KATEGORI)

        )

    else:

        cursor.execute(

            f"""

            UPDATE {LOG\_TABLE} SET status = %s, end\_time = %s, error\_message = %s

            WHERE process\_name = %s AND status = 'pending' AND dag\_name = %s AND mark = %s AND kategori = %s

            """, (status, now, error\_message, process\_name, dag\_name, mark, LOG\_KATEGORI)

        )

    conn.commit()

    cursor.close()

    conn.close()

def clean\_submit\_date(submit\_date\_str):

    if not submit\_date\_str:

        return None

    try:

        # Convert to datetime and remove milliseconds

        return submit\_date\_str.split('.')[0]

    except ValueError as e:

        logging.error(f"Error parsing submit\_date: {submit\_date\_str} - {e}")

        return None

def extract\_from\_clickhouse(\*\*kwargs):

    """Extract data from ClickHouse and store it as a DataFrame."""

    random\_value = random.randint(1000, 9999)  # Angka acak 4 digit

    timestamp = datetime.now().strftime("%Y%m%d%H%M%S")  # Format YYYYMMDDHHMMSS

    RANDOM\_VALUE = f"{random\_value}\_{timestamp}"

    process\_name = "extract\_clickhouse"

    log\_status(process\_name, RANDOM\_VALUE, "pending")

    try:

        clickhouse\_hook = ClickHouseHook(clickhouse\_conn\_id=SOURCE\_CONN\_ID)

        conn = clickhouse\_hook.get\_conn()

        data = conn.execute(EXTRACT\_QUERY)

        now = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

        data\_new = [(

            row[0], row[1], row[2], row[3], row[4], row[5],  row[6],

            row[7], row[8], row[9], row[10], row[11], row[12],  row[13],

            row[14], row[15], row[16], row[17], row[18], row[19],  row[20],

            row[21], row[22], row[23], now

        ) for row in data]

        df = pd.DataFrame(data\_new, columns=SOURCE\_COLUMNS)

        df.to\_json(FILE\_PATH, orient="records", default\_handler=str)

        kwargs['ti'].xcom\_push(key="file\_path", value=FILE\_PATH)

        kwargs['ti'].xcom\_push(key="random\_value", value=RANDOM\_VALUE)

        log\_status(process\_name, RANDOM\_VALUE, "success")

    except Exception as e:

        log\_status(process\_name, RANDOM\_VALUE, "failed", str(e))

        raise

def load\_to\_clickhouse(\*\*kwargs):

    """Load extracted data into ClickHouse."""

    random\_value = kwargs['ti'].xcom\_pull(task\_ids="extract\_clickhouse", key="random\_value")

    process\_name = "load\_clickhouse"

    log\_status(process\_name, random\_value, "pending")

    try:

        ti = kwargs['ti']

        file\_path = ti.xcom\_pull(task\_ids="extract\_clickhouse", key="file\_path")

        if not file\_path:

            raise ValueError("No file path received from ClickHouse extraction.")

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

            data\_dict = json.load(f)

        df = pd.DataFrame(data\_dict, dtype=str)

        df = df.applymap(lambda x: x.replace("\\", "") if isinstance(x, str) else x)

        df = df.applymap(lambda x: x.replace("'", "\\'") if isinstance(x, str) else x)

        hook = ClickHouseHook(clickhouse\_conn\_id=TARGET\_CONN\_ID)

        hook.execute(f"TRUNCATE TABLE `{TARGET\_DATABASE}`.`{TARGET\_TABLE}`")

        insert\_query = f"INSERT INTO `{TARGET\_DATABASE}`.`{TARGET\_TABLE}` VALUES"

        values = ', '.join(

            f"""({', '.join(f"'{x}'" for x in row)})"""

            for row in df.itertuples(index=False, name=None)

        )

        hook.execute(insert\_query + values)

        log\_status(process\_name, random\_value, "success")

    except Exception as e:

        log\_status(process\_name, random\_value, "failed", str(e))

        raise

extract\_clickhouse = PythonOperator(

    task\_id="extract\_clickhouse",

    python\_callable=extract\_from\_clickhouse,

    do\_xcom\_push=False,

    dag=dag,

)

load\_clickhouse = PythonOperator(

    task\_id="load\_clickhouse",

    python\_callable=load\_to\_clickhouse,

    do\_xcom\_push=False,

    provide\_context=True,

    dag=dag,

)

extract\_clickhouse >> load\_clickhouse

## **Truncate DM**

import json

import pandas as pd

import random

from airflow import DAG

from datetime import datetime, timedelta

from airflow.operators.python import PythonOperator

from airflow.providers.mysql.hooks.mysql import MySqlHook

from airflow.providers.postgres.hooks.postgres import PostgresHook

from airflow\_clickhouse\_plugin.hooks.clickhouse import ClickHouseHook

SOURCE\_CONN\_ID = "clickhouse\_dwh\_mitratel"

SOURCE\_DATABASE = "dwh\_tms"

SOURCE\_TABLE = "dim\_alluser"

TARGET\_CONN\_ID = "db\_datamart\_mitratel"

TARGET\_DATABASE = "datamart"

TARGET\_TABLE = "mart\_tms\_user"

FILE\_PATH = "/tmp/debug\_DM\_tms\_user.json"

DAG\_ID = "DM\_tms\_user"

DAG\_INTERVAL = "5-59/15 \* \* \* \*"

CHUNK\_SIZE = 5000

LOG\_CONN\_ID = "airflow\_logs\_mitratel"

LOG\_TABLE = "airflow\_logs"

LOG\_TYPE = "truncate"

LOG\_KATEGORI = "Data Mart"

EXTRACT\_QUERY = """

SELECT

    email, enabled, full\_name,

    mobile\_no, creation, last\_login,

    company\_name, module\_profile,

    role\_profile\_name, role\_detail, role\_unit,

    role\_unit AS role\_unit\_clean, position,

    territory, customer, customer\_group

FROM dwh\_tms.dim\_alluser FINAL

WHERE (email like '%@mitratel.co.id' OR email like '%@persadasokkatama.com')

"""

SOURCE\_COLUMNS=['email', 'enabled', 'full\_name',

                'mobile\_no', 'creation', 'last\_login',

                'company\_name', 'module\_profile',

                'role\_profile\_name', 'role\_detail', 'role\_unit',

                'role\_unit\_clean', 'position',

                'territory', 'customer', 'customer\_group']

default\_args = {

    'owner': 'airflow',

    'depends\_on\_past': False,

    'start\_date': datetime(2024, 2, 1, 0, 0),

    # 'retries': 1,

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

}

dag = DAG(

    dag\_id=DAG\_ID,

    schedule\_interval= DAG\_INTERVAL,

    default\_args=default\_args,

    catchup=False

)

def log\_status(process\_name, mark, status, error\_message=None):

    """Insert or update the log table."""

    pg\_hook = PostgresHook(postgres\_conn\_id=LOG\_CONN\_ID)

    conn = pg\_hook.get\_conn()

    cursor = conn.cursor()

    now = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

    dag\_name = f"{TARGET\_DATABASE}.{TARGET\_TABLE}"

    if status == "pending":

        cursor.execute(

            f"""

            INSERT INTO {LOG\_TABLE} (process\_name, dag\_name, type, status, start\_time, mark, kategori)

            VALUES (%s, %s, %s, %s, %s, %s, %s)

            """, (process\_name, dag\_name, LOG\_TYPE, status, now, mark, LOG\_KATEGORI)

        )

    else:

        cursor.execute(

            f"""

            UPDATE {LOG\_TABLE} SET status = %s, end\_time = %s, error\_message = %s

            WHERE process\_name = %s AND status = 'pending' AND dag\_name = %s AND mark = %s AND kategori = %s

            """, (status, now, error\_message, process\_name, dag\_name, mark, LOG\_KATEGORI)

        )

    conn.commit()

    cursor.close()

    conn.close()

def extract\_from\_clickhouse(\*\*kwargs):

    """Extract data from ClickHouse and store it as a DataFrame."""

    random\_value = random.randint(1000, 9999)  # Angka acak 4 digit

    timestamp = datetime.now().strftime("%Y%m%d%H%M%S")  # Format YYYYMMDDHHMMSS

    RANDOM\_VALUE = f"{random\_value}\_{timestamp}"

    process\_name = "extract\_clickhouse"

    log\_status(process\_name, RANDOM\_VALUE, "pending")

    try:

        clickhouse\_hook = ClickHouseHook(clickhouse\_conn\_id=SOURCE\_CONN\_ID)

        conn = clickhouse\_hook.get\_conn()

        data = conn.execute(EXTRACT\_QUERY)

        processed\_data = []

        for row in data:

            row = list(row)

            role\_unit\_clean = row[10].replace("- Mitratel", "").strip() if row[10] else None  # Index 10 = role\_unit

            row[11] = role\_unit\_clean  # Update role\_unit\_clean

            processed\_data.append(tuple(row))

        df = pd.DataFrame(processed\_data, columns=SOURCE\_COLUMNS).astype(str)

        df.to\_json(FILE\_PATH, orient="records", default\_handler=str)

        kwargs['ti'].xcom\_push(key="file\_path", value=FILE\_PATH)

        kwargs['ti'].xcom\_push(key="random\_value", value=RANDOM\_VALUE)

        log\_status(process\_name, RANDOM\_VALUE, "success")

    except Exception as e:

        log\_status(process\_name, RANDOM\_VALUE, "failed", str(e))

        raise

def load\_to\_postgres(\*\*kwargs):

    """Load extracted data into Postgres."""

    random\_value = kwargs['ti'].xcom\_pull(task\_ids="extract\_clickhouse", key="random\_value")

    process\_name = "load\_postgres"

    log\_status(process\_name, random\_value, "pending")

    try:

        ti = kwargs['ti']

        file\_path = ti.xcom\_pull(task\_ids="extract\_clickhouse", key="file\_path")

        if not file\_path:

            raise ValueError("No file path received from ClickHouse extraction.")

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

            data\_dict = json.load(f)

        df = pd.DataFrame(data\_dict, dtype=str)

        df = df.applymap(lambda x: x.replace("\\", "") if isinstance(x, str) else x)

        df = df.applymap(lambda x: x.replace("'", "\\'") if isinstance(x, str) else x)

        df = df.replace({'': None, 'None': None})

        hook = PostgresHook(postgres\_conn\_id=TARGET\_CONN\_ID)

        conn = hook.get\_conn()

        cursor = conn.cursor()

        insert\_query = """

        INSERT INTO public.mart\_tms\_user (

            email, enabled, full\_name, mobile\_no, creation, last\_login, company\_name, module\_profile,

            role\_profile\_name, role\_detail, role\_unit, role\_unit\_clean, position, territory, customer, customer\_group

        )

        VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)

        ON CONFLICT (email) DO UPDATE

        SET

            enabled = EXCLUDED.enabled,

            full\_name = EXCLUDED.full\_name,

            mobile\_no = EXCLUDED.mobile\_no,

            creation = EXCLUDED.creation,

            last\_login = EXCLUDED.last\_login,

            company\_name = EXCLUDED.company\_name,

            module\_profile = EXCLUDED.module\_profile,

            role\_profile\_name = EXCLUDED.role\_profile\_name,

            role\_detail = EXCLUDED.role\_detail,

            role\_unit = EXCLUDED.role\_unit,

            role\_unit\_clean = EXCLUDED.role\_unit\_clean,

            position = EXCLUDED.position,

            territory = EXCLUDED.territory,

            customer = EXCLUDED.customer,

            customer\_group = EXCLUDED.customer\_group,

            updated\_at = NOW() + INTERVAL '7 hours'

        """

        for \_\_, row in df.iterrows():

            cursor.execute(insert\_query, (row['email'], row['enabled'], row['full\_name'],

                                               row['mobile\_no'], row['creation'],

                                               row['last\_login'], row['company\_name'],

                                               row['module\_profile'], row['role\_profile\_name'], row['role\_detail'],

                                               row['role\_unit'], row['role\_unit\_clean'],

                                               row['position'], row['territory'],

                                               row['customer'], row['customer\_group']

                                               ))

        conn.commit()

        log\_status(process\_name, random\_value, "success")

    except Exception as e:

        log\_status(process\_name, random\_value, "failed", str(e))

        raise

extract\_clickhouse = PythonOperator(

    task\_id="extract\_clickhouse",

    python\_callable=extract\_from\_clickhouse,

    do\_xcom\_push=False,

    dag=dag,

)

load\_postgres = PythonOperator(

    task\_id="load\_postgres",

    python\_callable=load\_to\_postgres,

    do\_xcom\_push=False,

    provide\_context=True,

    dag=dag,

)

extract\_clickhouse >> load\_postgres