PRAKTIKUM REKAYASA PERANGKAT LUNAK 2

LAPORAN AKHIR

"PENGEMBANGAN AUTOMATED DATA PIPELINE DATA CUACA MENGGUNAKAN WEATHERSTACK API BERBASIS APACHE AIRFLOW, POSTGRESQL, DBT, APACHE SUPERSET, DAN DOCKER"

Ahmad Maulana	**50422124**
Alya Gustasya	**50422186**

Kelas : 4IA16

Jurusan : Informatika

UNIVERSITAS GUNADARMA

FAKULTAS TEKNOLOGI INDUSTRI

2025

DAFTAR ISI

BAB I PENDAHULUAN

1.1 Latar Belakang

Perkembangan teknologi informasi yang semakin pesat mendorong organisasi dan institusi untuk mengelola data secara lebih efektif dan terstruktur. Data tidak lagi diproses secara manual, melainkan melalui sistem otomatis yang mampu menangani data dalam jumlah besar secara konsisten dan berkelanjutan. Salah satu jenis data yang banyak dimanfaatkan dalam berbagai sektor adalah data cuaca.

Data cuaca memiliki peranan penting dalam bidang transportasi, pertanian, logistik, mitigasi bencana, serta perencanaan aktivitas sehari-hari. Umumnya, data cuaca disediakan oleh pihak ketiga dalam bentuk Application Programming Interface (API), seperti Weatherstack. Namun, data yang tersedia melalui API tersebut masih bersifat mentah dan memerlukan proses lanjutan agar dapat digunakan untuk analisis dan pengambilan keputusan.

Permasalahan yang sering ditemui adalah proses pengambilan data cuaca yang masih dilakukan secara manual atau tidak terjadwal, sehingga data menjadi tidak konsisten dan sulit dianalisis. Selain itu, tanpa adanya sistem orkestrasi yang baik, proses pengambilan, penyimpanan, transformasi, dan visualisasi data menjadi kurang efisien.

Oleh karena itu, diperlukan sebuah sistem **Automated Data Pipeline** yang mampu mengelola data cuaca secara otomatis mulai dari proses pengambilan data, penyimpanan ke basis data, transformasi data analitik, hingga penyajian data dalam bentuk visualisasi dashboard. Sistem ini diharapkan dapat menjadi solusi pengolahan data cuaca yang terstruktur, terjadwal, dan mudah dikembangkan.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah diuraikan, maka rumusan masalah dalam proyek ini adalah sebagai berikut:

1. Bagaimana membangun automated data pipeline untuk pengambilan data cuaca dari Weatherstack API?
2. Bagaimana mengorkestrasi proses pengambilan dan pengolahan data menggunakan Apache Airflow?
3. Bagaimana menyimpan dan mentransformasikan data cuaca agar siap digunakan untuk analisis?

4. Bagaimana menyajikan data cuaca dalam bentuk dashboard visual yang informatif?

1.3 Tujuan Penelitian

Tujuan dari pengembangan sistem ini adalah:

1. Membangun sistem automated data pipeline berbasis API cuaca.
2. Mengimplementasikan Apache Airflow sebagai sistem orkestrasi dan penjadwalan workflow.
3. Menggunakan PostgreSQL sebagai media penyimpanan data cuaca.
4. Melakukan transformasi data menggunakan dbt untuk kebutuhan analitik.
5. Menyajikan data cuaca dalam bentuk dashboard menggunakan Apache Superset.

1.4 Manfaat Penelitian

Manfaat yang diharapkan dari proyek ini adalah sebagai berikut:

- Memberikan pemahaman mengenai implementasi data engineering end-to-end.
- Menjadi studi kasus penerapan automated data pipeline.
- Menjadi media pembelajaran pada mata kuliah Rekayasa Perangkat Lunak dan Data Engineering.
- Menjadi portfolio proyek berbasis industri.

1.5 Batasan Masalah

Agar pembahasan lebih terarah, maka batasan masalah dalam proyek ini adalah:

1. Data yang digunakan hanya berasal dari Weatherstack API.
2. Sistem dijalankan pada lingkungan lokal menggunakan Docker.
3. Fokus pengembangan pada proses pengambilan, pengolahan, dan visualisasi data.
4. Tidak dilakukan analisis prediktif atau machine learning.

BAB II METODOLOGI PENGEMBANGAN SISTEM

2.1 Metode Pengembangan Perangkat Lunak

Metode pengembangan perangkat lunak yang digunakan dalam proyek ini adalah metode **Waterfall**. Metode ini dipilih karena tahapan pengembangan sistem dilakukan secara berurutan dan kebutuhan sistem telah didefinisikan sejak awal.

Tahapan metode Waterfall meliputi:

1. Analisis kebutuhan
2. Perancangan sistem
3. Implementasi
4. Pengujian
5. Deployment dan dokumentasi

2.2 Analisis Kebutuhan Sistem

Kebutuhan sistem pada proyek ini terdiri dari kebutuhan fungsional dan non-fungsional.

2.2.1 Kebutuhan Fungsional

- Sistem dapat mengambil data cuaca dari Weatherstack API
- Sistem dapat menyimpan data cuaca ke PostgreSQL
- Sistem dapat menjalankan pipeline secara otomatis dan terjadwal
- Sistem dapat menampilkan data dalam bentuk dashboard

2.2.2 Kebutuhan Non-Fungsional

- Sistem berjalan pada lingkungan Docker
- Sistem mudah dikembangkan dan dipelihara
- Sistem berjalan secara stabil

2.3 Arsitektur Sistem

Arsitektur sistem menggunakan konsep Extract, Load, Transform, dan Visualize (ELT). Apache Airflow digunakan sebagai orchestrator utama yang mengatur alur kerja pipeline data.

2.4 Tools dan Teknologi

Tools dan teknologi yang digunakan dalam proyek ini antara lain:

- Python
- Weatherstack API
- PostgreSQL
- Apache Airflow
- dbt
- Apache Superset
- Docker dan Docker Compose

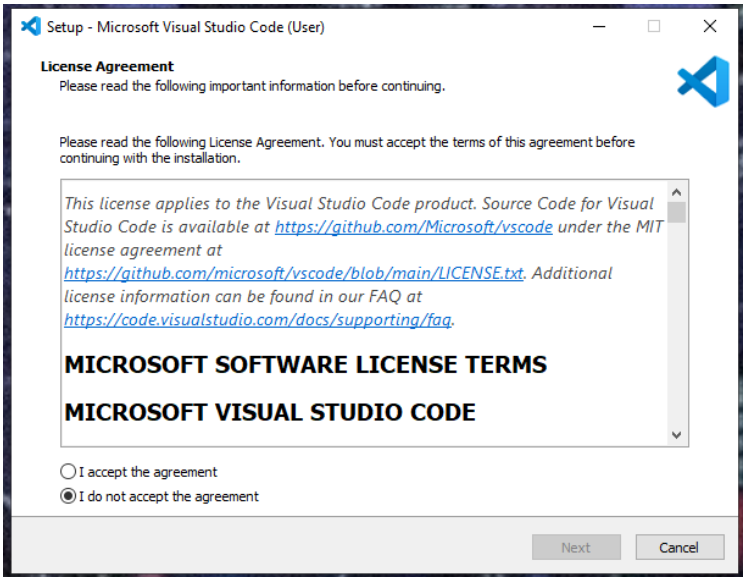
BAB III IMPLEMENTASI DAN PEMBAHASAN

3.1 Persiapan Lingkungan Pengembangan

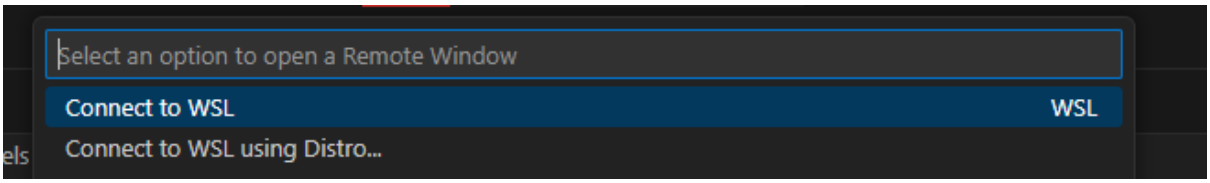
Pada tahap awal pengembangan sistem, dilakukan persiapan lingkungan pengembangan untuk memastikan seluruh tools dan teknologi yang digunakan dapat berjalan dengan baik dan saling terintegrasi. Lingkungan pengembangan disiapkan pada sistem operasi Windows dengan memanfaatkan Windows Subsystem for Linux (WSL) sebagai lingkungan kerja berbasis Linux.

Adapun tahapan persiapan lingkungan pengembangan meliputi:

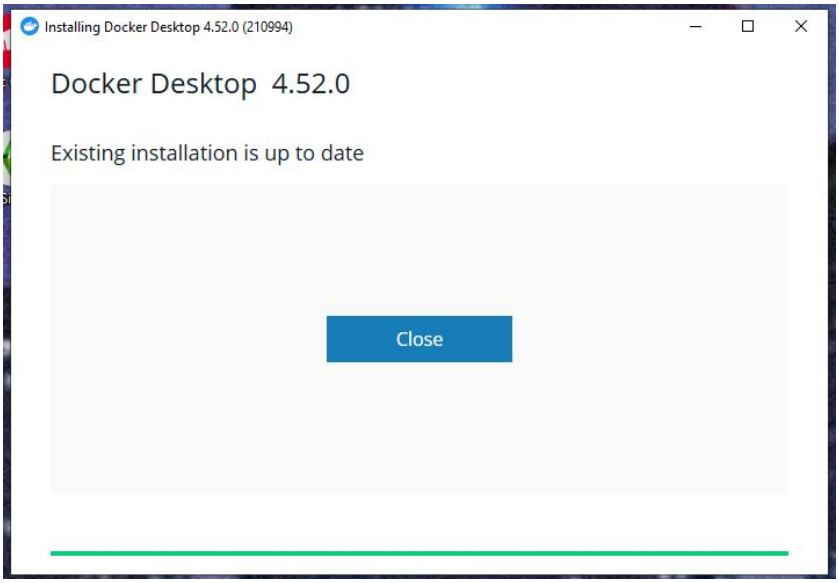
- **Instalasi Visual Studio Code sebagai text editor utama untuk pengembangan script Python, konfigurasi Docker, Airflow, dan dbt.**

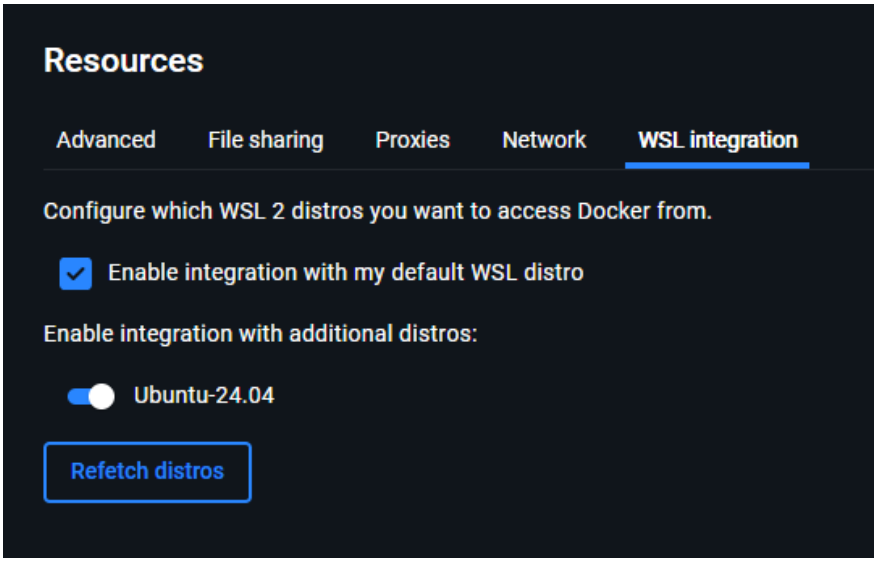


- **Setup Windows Subsystem for Linux (WSL) untuk menyediakan lingkungan Linux yang stabil dan kompatibel dengan tools data engineering.**



- **Instalasi dan konfigurasi Docker serta Docker Compose untuk menjalankan seluruh service dalam bentuk container.**





Tahap ini bertujuan untuk memastikan bahwa proses pengembangan dapat dilakukan secara terisolasi, konsisten, dan mudah direplikasi di lingkungan lain.

3.2 Integrasi Weatherstack API

Pada tahap ini dilakukan integrasi dengan Weatherstack API sebagai sumber utama data cuaca. Weatherstack merupakan layanan API berbasis HTTP yang menyediakan data cuaca secara real-time berdasarkan lokasi tertentu. Integrasi ini bertujuan untuk memperoleh data cuaca mentah yang selanjutnya akan disimpan ke database PostgreSQL dan diproses lebih lanjut.

3.2.1 Pendaftaran dan Perolehan API Key

Langkah pertama dalam integrasi Weatherstack API adalah melakukan pendaftaran pada platform Weatherstack. Setelah proses pendaftaran berhasil, pengguna akan memperoleh API Key yang digunakan sebagai autentikasi dalam setiap permintaan (request) ke API Weatherstack.

API Key ini bersifat unik dan wajib disertakan pada parameter request agar API dapat diakses.

3.2.2 Pembuatan Script Python untuk Request Data Cuaca

Setelah memperoleh API Key, dibuat sebuah script Python untuk melakukan HTTP request ke Weatherstack API. Script ini bertugas mengambil data cuaca berdasarkan lokasi tertentu dan mengembalikannya dalam format JSON.

Berikut adalah potongan kode dari file api_request.py:

```
import requests


api_key = "c0e5b9bbfc8069b5346932f637dfbed9"
api_url = f"http://api.weatherstack.com/current?access_key={api_key}&query=New York"


def fetch_data():
    print("Fetching weather data from Weatherstack API...")
    try:
        response = requests.get(api_url)
        response.raise_for_status()
        print("API response received successfully")
```

```
        return response.json()


    except requests.exceptions.RequestException as e:
        print(f"An error accured: {e}")
        raise
```

Pada kode di atas:

- **Library requests digunakan untuk melakukan HTTP request.**
- **API Key disertakan pada parameter access_key.**
- **Lokasi cuaca ditentukan menggunakan parameter query.**
- **Data yang diterima dari API dikembalikan dalam format JSON.**

3.2.3 Pengujian Response API

Untuk memastikan bahwa struktur data yang diterima sesuai dengan dokumentasi Weatherstack, dilakukan pengujian response API menggunakan data tiruan (mock data). Pengujian ini berguna untuk memvalidasi struktur JSON tanpa harus selalu melakukan request ke API secara langsung.

Berikut adalah contoh fungsi pengujian pada file api_request.py:

```
def mock_fetch_data():
    return {
        'request': {
            'type': 'City',
            'query': 'New York, United States of America',
            'language': 'en',
            'unit': 'm'
        },
        'location': {
            'name': 'New York',
            'country': 'United States of America',
            'region': 'New York',
            'lat': '40.714',
```

```
        'lon': '-74.006',
        'timezone_id': 'America/New_York',
        'localtime': '2025-12-22 23:16',
        'localtime_epoch': 1766445360,
        'utc_offset': '-5.0'
    },
    'current': {
        'observation_time': '04:16 AM',
        'temperature': 3,
        'weather_code': 113,
        'weather_icons': [
'https://cdn.worldweatheronline.com/images/wsymbols01_png_64/wsymbol_0008_clear_sky_night.png'
        ],
        'weather_descriptions': ['Clear'],
        'wind_speed': 14,
        'wind_degree': 244,
        'wind_dir': 'WSW',
        'pressure': 1030,
        'precip': 0,
        'humidity': 44,
        'cloudcover': 25,
        'feelslike': -1,
        'uv_index': 0,
        'visibility': 16,
        'is_day': 'no'
    }
}
```

Mock data ini digunakan untuk memastikan bahwa struktur data mencakup informasi lokasi dan kondisi cuaca yang dibutuhkan sebelum disimpan ke database.

3.2.4 Validasi Data Cuaca

Setelah data diterima dari API, dilakukan proses validasi untuk memastikan bahwa data tidak mengandung nilai kosong (null) atau error yang dapat mengganggu proses penyimpanan dan transformasi data selanjutnya.

Data cuaca yang berhasil diambil meliputi:

- **Informasi lokasi (kota, negara, koordinat)**
- **Waktu pengambilan data**
- **Suhu udara**
- **Kelembaban**
- **Kecepatan angin**
- **Kondisi cuaca**

Data yang telah divalidasi ini selanjutnya digunakan sebagai data mentah (raw data) yang akan disimpan ke database PostgreSQL dan diproses menggunakan dbt pada tahap berikutnya.

3.3 Setup Database PostgreSQL

Database PostgreSQL digunakan sebagai media penyimpanan data cuaca hasil pengambilan dari Weatherstack API. Database ini dijalankan menggunakan Docker Compose agar mudah dikonfigurasi, portable, dan terisolasi dari sistem host.

3.3.1 Pembuatan Service PostgreSQL pada Docker Compose

Pembuatan database PostgreSQL dilakukan dengan mendefinisikan sebuah service pada file docker-compose.yml. Service ini menggunakan image resmi PostgreSQL versi 16-alpine.

Potongan kode:

```
services:
  db:
```

```
container_name: postgres_container
image: postgres:16-alpine
ports:
  - 5000:5432
```

Konfigurasi tersebut menjalankan PostgreSQL di dalam container dengan port database 5432 yang dipetakan ke port 5000 pada host.

3.3.2 Konfigurasi Database (Nama Database, Username, Password)

Konfigurasi database dilakukan melalui environment variable yang didefinisikan di dalam Docker Compose. Hal ini memungkinkan proses setup database dilakukan secara otomatis saat container pertama kali dijalankan.

Potongan kode:

```
environment:
  POSTGRES_DB: db
  POSTGRES_USER: db_user
  POSTGRES_PASSWORD: db_password
```

Konfigurasi tersebut menghasilkan:

- **Nama database: db**
- **Username: db_user**
- **Password: db_password**

3.3.3 Inisialisasi dan Penyimpanan Data Database

Untuk menjaga persistensi data, volume digunakan agar data PostgreSQL tetap tersimpan meskipun container dihentikan atau dijalankan ulang. Selain itu, file SQL inisialisasi digunakan untuk pembuatan schema awal.

Potongan kode:

```
volumes:
  - ./postgres/data:/var/lib/postgresql/data
  - ./postgres/airflow_init.sql:/docker-entrypoint-initdb.d/airflow_init.sql
```

Penggunaan volume memastikan bahwa data cuaca yang tersimpan tidak hilang ketika container dihapus atau direstart.

3.3.4 Perancangan Struktur Tabel Penyimpanan Data Cuaca

Struktur tabel data cuaca dirancang untuk menyimpan data mentah hasil respon API Weatherstack. Data tersebut mencakup informasi lokasi, suhu, kelembaban, kecepatan angin, dan waktu observasi.

Pembuatan tabel dilakukan melalui proses ingestion menggunakan script Python insert_records.py, yang secara otomatis membuat dan mengisi tabel raw_weather_data pada schema dev.

Contoh struktur tabel:

dev.raw_weather_data

Tabel ini berfungsi sebagai raw layer atau data mentah sebelum dilakukan transformasi menggunakan dbt.

3.3.5 Pengujian Koneksi Database

Pengujian koneksi database dilakukan dengan mengakses PostgreSQL secara langsung dari dalam container menggunakan perintah berikut:

docker exec -it postgres_container psql -U db_user -d db

Selanjutnya dilakukan pengecekan schema dan tabel dengan perintah:

\dn

\dt dev.*

Hasil pengujian menunjukkan bahwa database PostgreSQL dapat diakses dengan baik dan siap digunakan oleh service lain seperti Airflow, Python ingestion, dan dbt.

3.3.6 Hasil Setup Database

Berdasarkan hasil konfigurasi dan pengujian, dapat disimpulkan bahwa:

- **Database PostgreSQL berhasil dijalankan menggunakan Docker.**
- **Struktur database siap menerima data cuaca dari Weatherstack API.**
- **Database berfungsi sebagai tempat penyimpanan data mentah (raw data) sebelum dilakukan proses transformasi menggunakan dbt.**

3.4 Proses Ingest Data ke PostgreSQL

Setelah database PostgreSQL berhasil disiapkan, tahap selanjutnya adalah melakukan ingestion data cuaca ke dalam database. Proses ingestion ini bertujuan untuk menyimpan data mentah hasil pengambilan dari Weatherstack API ke dalam PostgreSQL sebelum dilakukan proses transformasi data menggunakan dbt.

Proses ingestion dikembangkan menggunakan bahasa pemrograman Python dengan bantuan library psycopg2 sebagai penghubung antara aplikasi dan database PostgreSQL.

3.4.1 Koneksi ke Database PostgreSQL

Langkah pertama pada proses ingestion adalah membangun koneksi antara script Python dan database PostgreSQL. Koneksi ini dilakukan menggunakan parameter host, port, nama database, username, dan password yang telah dikonfigurasi pada Docker Compose.

Berikut adalah potongan kode untuk koneksi database pada file insert_records.py:

```
import psycopg2


def connect_to_db():
    print("Connecting to the postgresSQL database...")
    try:
        conn = psycopg2.connect(
            host="db",
            port=5432,
            dbname="db",
            user="db_user",
            password="db_password"
        )
        return conn
    except psycopg2.Error as e:
        print(f"database connection failed: {e}")
```

```
        raise
```

Koneksi ini dilakukan dari dalam container Docker, sehingga hostname database menggunakan nama service db sesuai dengan konfigurasi Docker Compose.

3.4.2 Pembuatan Struktur Tabel

Sebelum melakukan penyimpanan data, sistem memastikan bahwa schema dan tabel tujuan telah tersedia di dalam database. Apabila tabel belum ada, maka tabel akan dibuat secara otomatis.

```
def create_table(conn):
    print("Creating table if not exist...")
    try:
        cursor = conn.cursor()
        cursor.execute("""
            CREATE SCHEMA IF NOT EXISTS dev;
            CREATE TABLE IF NOT EXISTS dev.raw_weather_data (
                id SERIAL PRIMARY KEY,
                city TEXT,
                temperature FLOAT,
                weather_descriptions TEXT,
                wind_speed FLOAT,
                time TIMESTAMP,
                inserted_at TIMESTAMP DEFAULT NOW(),
                utc_offset TEXT
            );
        """)
        conn.commit()
        print("Table was created.")
    except psycopg2.Error as e:
        print(f"Failed to create table: {e}")
        raise
```

Struktur tabel ini dirancang untuk menyimpan data cuaca mentah (raw data) yang diperoleh langsung dari API tanpa transformasi lanjutan.

3.4.3 Proses Insert Data Cuaca

Setelah tabel tersedia, data cuaca yang diperoleh dari Weatherstack API disimpan ke dalam database menggunakan perintah SQL INSERT.

```
def insert_records(conn, data):
    print("Inserting weather data into the database...")
    try:
        weather = data['current']
        location = data['location']
        cursor = conn.cursor()
        cursor.execute("""
            INSERT INTO dev.raw_weather_data(
                city,
                temperature,
                weather_descriptions,
                wind_speed,
                time,
                inserted_at,
                utc_offset
            ) VALUES (%s, %s, %s, %s, %s, NOW(), %s)
        """, (
            location['name'],
            weather['temperature'],
            weather['weather_descriptions'][0],
            weather['wind_speed'],
            location['localtime'],
            location['utc_offset']
        ))
```

```
        conn.commit()
        print("Data successfully inserted")
    except psycopg2.Error as e:
        print(f"Error inserting data into the database: {e}")
        raise
```

Data yang disimpan meliputi informasi lokasi, suhu, kondisi cuaca, kecepatan angin, serta waktu pengambilan data.

3.4.4 Orkestrasi Proses Ingestion

Seluruh proses ingestion dijalankan secara terstruktur melalui fungsi utama main(), yang mencakup pengambilan data dari API, pembuatan tabel, dan proses penyimpanan data ke database.

```
from api_request import fetch_data


def main():
    try:
        data = fetch_data()
        conn = connect_to_db()
        create_table(conn)
        insert_records(conn, data)
    except Exception as e:
        print(f"An error occurred during execution: {e}")
    finally:
        if 'conn' in locals():
            conn.close()
            print("Database connection closed")
```

Fungsi ini memastikan bahwa setiap tahap berjalan secara berurutan dan koneksi database ditutup dengan aman setelah proses selesai.

3.4.5 Pengujian dan Validasi Data

Pengujian dilakukan dengan menjalankan script ingestion dan memeriksa isi tabel pada PostgreSQL menggunakan perintah SQL. Hasil pengujian menunjukkan bahwa data cuaca berhasil tersimpan ke dalam tabel dev.raw_weather_data secara konsisten dan sesuai dengan struktur yang telah dirancang.

Dengan demikian, tahap ingestion data ke PostgreSQL telah berhasil dilakukan dan menghasilkan data mentah yang siap untuk diproses lebih lanjut pada tahap transformasi menggunakan dbt.

3.5 Orkestrasi Workflow Menggunakan Apache Airflow

Apache Airflow digunakan sebagai sistem orkestrasi workflow untuk mengatur alur eksekusi, penjadwalan, dan monitoring proses ingestion data cuaca dari Weatherstack API ke database PostgreSQL. Dengan menggunakan Airflow, proses pengambilan dan penyimpanan data dapat dijalankan secara otomatis dan terjadwal tanpa perlu intervensi manual.

Airflow dijalankan menggunakan Docker Compose agar mudah dikonfigurasi dan terintegrasi dengan service lain seperti PostgreSQL.

3.5.1 Menjalankan Apache Airflow Menggunakan Docker Compose

Service Apache Airflow dikonfigurasi pada file docker-compose.yml dengan menggunakan image resmi Airflow. Konfigurasi ini mencakup pengaturan port, koneksi database metadata, serta mounting folder DAG.

```
af:
  container_name: airflow_container
  image: apache/airflow:3.1.5-python3.13
  ports:
    - 8000:8080
  environment:
    AIRFLOW__DATABASE__SQL_ALCHEMY_CONN:
postgresql+psycopg2://airflow:airflow@db:5432/airflow_db
  volumes:
    - ./airflow/dags:/opt/airflow/dags
    - ./api-request:/opt/airflow/api-request
```

```
depends_on:
  - db
command: >
  bash -c "airflow db migrate && airflow standalone"
```

Konfigurasi ini memungkinkan Airflow berjalan dalam satu container dengan webserver, scheduler, dan database metadata yang terhubung ke PostgreSQL.

3.5.2 Konfigurasi Environment Apache Airflow

Airflow dikonfigurasi agar dapat:

- **Mengakses script ingestion Python.**
- **Berkomunikasi dengan database PostgreSQL.**
- **Menjalankan workflow berbasis waktu (schedule).**

Folder api-request dimount ke dalam container Airflow agar script Python ingestion dapat dipanggil langsung oleh Airflow.

3.5.3 Pembuatan DAG Ingestion Data Cuaca

Workflow ingestion didefinisikan dalam sebuah DAG (Directed Acyclic Graph) yang merepresentasikan urutan task yang harus dijalankan.

DAG dibuat di dalam folder airflow/dags dan berfungsi untuk:

1. **Menjalankan proses pengambilan data cuaca dari Weatherstack API.**
2. **Menyimpan data ke database PostgreSQL.**

Contoh implementasi DAG ingestion data cuaca:

```
from airflow import DAG
from airflow.operators.bash import BashOperator
from datetime import datetime

default_args = {
   'owner': 'airflow',
   'start_date': datetime(2025, 12, 1),
   'retries': 1,
```

```
}


with DAG(
    dag_id='weather_data_ingestion',
    default_args=default_args,
    schedule_interval='@daily',
    catchup=False
) as dag:


    ingest_weather_data = BashOperator(
        task_id='ingest_weather_data',
        bash_command='python /opt/airflow/api-request/insert_records.py'
    )


    ingest_weather_data
```

DAG ini memastikan bahwa script ingestion dijalankan sebagai satu kesatuan workflow yang terkontrol.

3.5.4 Penjadwalan Eksekusi Pipeline

Airflow memungkinkan penjadwalan pipeline secara otomatis menggunakan parameter schedule_interval. Pada penelitian ini digunakan jadwal harian (@daily), sehingga data cuaca akan diambil dan disimpan secara rutin setiap hari.

Pengaturan catchup=False digunakan agar Airflow tidak menjalankan pipeline untuk waktu lampau yang terlewat.

3.5.5 Monitoring dan Kontrol Workflow

Melalui Airflow Web UI, pengguna dapat:

- **Memantau status eksekusi DAG (success, failed, running).**
- **Melihat log eksekusi task secara detail.**
- **Melakukan trigger DAG secara manual apabila diperlukan.**

Dengan adanya Airflow, proses ingestion data cuaca menjadi:

- **Terstruktur**
- **Terjadwal**
- **Mudah dipantau**
- **Lebih andal dibandingkan eksekusi manual**

3.6 Transformasi Data Menggunakan dbt

Setelah data cuaca berhasil disimpan ke dalam database PostgreSQL melalui proses ingestion, tahap selanjutnya adalah melakukan transformasi data agar data mentah (raw data) dapat digunakan untuk keperluan analisis dan visualisasi. Proses transformasi ini dilakukan menggunakan dbt (data build tool).

dbt digunakan untuk mengelola transformasi data langsung di dalam database dengan pendekatan *ELT (Extract, Load, Transform)*, di mana proses transformasi dilakukan setelah data dimuat ke database.

3.6.1 Setup dbt Menggunakan Docker Compose

dbt dijalankan menggunakan Docker agar konsisten dengan lingkungan pengembangan lainnya serta memudahkan integrasi dengan PostgreSQL. Service dbt dikonfigurasi dalam file docker-compose.yml sebagai berikut:

```
dbt:
  container_name: dbt_container
  image: ghcr.io/dbt-labs/dbt-postgres:1.9.latest
  volumes:
    - ./dbt/my_project:/usr/app
    - ./dbt/profiles:/root/.dbt
  working_dir: /usr/app
  depends_on:
    - db
  networks:
    - my-network
```

Konfigurasi ini memungkinkan dbt:

- **Mengakses project dbt (dbt_project.yml)**
- **Menggunakan file profiles.yml untuk koneksi ke PostgreSQL**
- **Menjalankan transformasi langsung pada database yang sama dengan data mentah**

3.6.2 Konfigurasi Koneksi Database dbt

Koneksi dbt ke PostgreSQL didefinisikan pada file profiles.yml. File ini berisi konfigurasi database target yang akan digunakan oleh dbt saat menjalankan transformasi.

Contoh konfigurasi profiles.yml:

```
my_project:
  target: dev
  outputs:
    dev:
      type: postgres
      host: db
      user: db_user
      password: db_password
      port: 5432
      dbname: db
      schema: dev
```

Konfigurasi tersebut mengarahkan dbt untuk membaca dan menulis data pada schema dev di database PostgreSQL.

3.6.3 Pembuatan Source dbt (Raw Data)

Data mentah hasil ingestion disimpan pada tabel dev.raw_weather_data. Tabel ini didefinisikan sebagai source di dbt agar dapat digunakan pada proses transformasi.

File sources.yml dibuat pada folder models:

```
version: 2
```

```
sources:
  - name: weather
    schema: dev
    tables:
      - name: raw_weather_data
```

Pendefinisian source ini memungkinkan dbt melakukan validasi dan dokumentasi terhadap data mentah.

3.6.4 Pembuatan Model Transformasi Data

Model dbt dibuat untuk melakukan transformasi data mentah menjadi data yang lebih terstruktur. Pada penelitian ini dibuat model *staging* untuk membersihkan dan memilih kolom yang relevan dari tabel raw.

Contoh model transformasi staging.sql:

```
select
    city,
    temperature,
    weather_descriptions,
    wind_speed,
    time,
    inserted_at
from {{ source('weather', 'raw_weather_data') }}
```

Model ini menghasilkan tabel staging yang berisi data cuaca terstruktur dan siap dianalisis.

3.6.5 Pengujian dan Eksekusi Transformasi Data

Transformasi data dijalankan menggunakan perintah dbt berikut:

docker compose run dbt

Perintah tersebut akan:

- **Membaca data mentah dari PostgreSQL**
- **Menjalankan query transformasi**

- **Membuat tabel hasil transformasi pada schema dev**

Keberhasilan proses ditandai dengan status PASS pada output dbt dan terbentuknya tabel hasil transformasi di database.

3.6.6 Hasil Transformasi Data

Hasil dari proses transformasi menunjukkan bahwa:

- **Data mentah berhasil diubah menjadi data terstruktur**
- **Struktur tabel menjadi lebih konsisten dan mudah digunakan**
- **Data siap digunakan untuk kebutuhan visualisasi dan analisis lanjutan**

3.7 Visualisasi Data Menggunakan Apache Superset

Setelah data cuaca berhasil melalui proses ingestion dan transformasi, tahap selanjutnya adalah melakukan visualisasi data. Visualisasi bertujuan untuk menyajikan data dalam bentuk grafik dan dashboard agar lebih mudah dipahami dan dianalisis. Pada penelitian ini digunakan Apache Superset sebagai alat visualisasi data.

Apache Superset merupakan platform *business intelligence* berbasis web yang mampu menampilkan dashboard interaktif dan terhubung langsung dengan berbagai database, termasuk PostgreSQL.

3.7.1 Setup Apache Superset Menggunakan Docker Compose

Apache Superset dijalankan menggunakan Docker Compose agar dapat terintegrasi dengan service PostgreSQL dan service lainnya dalam satu lingkungan jaringan.

Contoh konfigurasi service Superset pada file docker-compose.yml:

```
superset:
  container_name: superset_container
  image: apache/superset:latest
  ports:
    - 8088:8088
  environment:
    SUPERSET_SECRET_KEY: superset_secret_key
  depends_on:
```

```
  - db
networks:
  - my-network
```

Konfigurasi ini memungkinkan Superset:

- **Berjalan sebagai aplikasi web**
- **Terhubung langsung ke database PostgreSQL**
- **Mengakses data hasil transformasi**

3.7.2 Konfigurasi Koneksi Superset ke PostgreSQL

Setelah Superset berjalan, dilakukan konfigurasi koneksi database melalui menu Database Connections pada Superset.

Parameter koneksi yang digunakan meliputi:

- **Database Type: PostgreSQL**
- **Host: db**
- **Port: 5432**
- **Database Name: db**
- **Username: db_user**
- **Password: db_password**

Koneksi ini memungkinkan Superset membaca data langsung dari schema dev pada PostgreSQL.

3.7.3 Pembuatan Dataset pada Apache Superset

Dataset dibuat berdasarkan tabel hasil transformasi dbt, misalnya tabel dev.staging. Dataset ini menjadi sumber utama dalam pembuatan visualisasi.

Langkah pembuatan dataset meliputi:

- **Memilih database PostgreSQL yang telah dikonfigurasi**
- **Memilih schema dev**
- **Memilih tabel hasil transformasi**
- **Menyimpan dataset untuk digunakan pada chart dan dashboard**

3.7.4 Pembuatan Chart Visualisasi Data Cuaca

Setelah dataset tersedia, dilakukan pembuatan berbagai jenis chart untuk menampilkan informasi cuaca, antara lain:

- **Grafik suhu berdasarkan waktu**
- **Distribusi kondisi cuaca**
- **Kecepatan angin per lokasi**
- **Tren perubahan suhu**

Superset menyediakan berbagai jenis visualisasi seperti *line chart*, *bar chart*, dan *table chart* yang dapat dikonfigurasi secara interaktif.

3.7.5 Pembuatan Dashboard Visualisasi

Chart yang telah dibuat kemudian digabungkan ke dalam satu dashboard. Dashboard ini berfungsi sebagai tampilan utama untuk memantau data cuaca.

Dashboard yang dihasilkan bersifat:

- **Interaktif, pengguna dapat melakukan filter data**
- **Real-time, menampilkan data terbaru dari database**
- **User-friendly, mudah digunakan untuk analisis**

3.8 Pengujian Sistem

Pengujian sistem dilakukan untuk memastikan seluruh komponen dalam Automated Data Pipeline berjalan dengan baik dan saling terintegrasi sesuai dengan kebutuhan sistem. Pengujian tidak hanya dilakukan pada masing-masing komponen secara terpisah, tetapi juga dilakukan pengujian end-to-end untuk memastikan alur data berjalan secara utuh dari pengambilan data hingga visualisasi.

3.8.1 Pengujian Pengambilan Data dari API

Pengujian ini bertujuan untuk memastikan sistem mampu mengambil data cuaca dari Weatherstack API dengan benar.

Pengujian dilakukan dengan:

- **Menjalankan script Python untuk melakukan request ke Weatherstack API**
- **Memastikan response API diterima tanpa error**

- **Memvalidasi struktur data JSON yang diterima**

Hasil pengujian menunjukkan bahwa sistem berhasil menerima data cuaca dari API secara konsisten dan sesuai dengan format yang diharapkan.

3.8.2 Pengujian Penyimpanan Data ke PostgreSQL

Pengujian penyimpanan data dilakukan untuk memastikan data hasil pengambilan API dapat disimpan ke dalam database PostgreSQL.

Tahapan pengujian meliputi:

- **Pengujian koneksi database menggunakan library psycopg2**
- **Pengujian eksekusi query SQL untuk proses insert data**
- **Pemeriksaan data yang tersimpan pada tabel database**

Hasil pengujian menunjukkan bahwa data cuaca berhasil disimpan ke dalam tabel PostgreSQL tanpa kehilangan data dan dapat diakses kembali untuk proses selanjutnya.

3.8.3 Pengujian Eksekusi DAG Apache Airflow

Pengujian ini bertujuan untuk memastikan workflow ingestion data dapat dijalankan secara otomatis melalui Apache Airflow.

Pengujian dilakukan dengan:

- **Menjalankan DAG pada Airflow secara manual**
- **Mengamati status task pada Airflow Web UI**
- **Memastikan seluruh task berjalan dengan status *success***

Hasil pengujian menunjukkan bahwa DAG Airflow berhasil mengeksekusi proses ingestion data sesuai dengan alur yang telah dirancang.

3.8.4 Pengujian Transformasi Data Menggunakan dbt

Pengujian transformasi data dilakukan untuk memastikan model dbt dapat mengolah data mentah menjadi data analitik.

Tahapan pengujian meliputi:

- **Menjalankan perintah dbt run**
- **Menjalankan pengujian menggunakan dbt test**

- **Memverifikasi hasil transformasi pada tabel target**

Hasil pengujian menunjukkan bahwa proses transformasi data berjalan dengan baik dan menghasilkan data yang lebih terstruktur dan siap digunakan untuk visualisasi.

3.8.5 Pengujian Dashboard Apache Superset

Pengujian dashboard dilakukan untuk memastikan visualisasi data dapat ditampilkan dengan baik dan menampilkan data yang benar.

Pengujian meliputi:

- **Pengujian koneksi Superset ke PostgreSQL**
- **Pengujian pemuatan dataset**
- **Pengujian interaksi chart dan dashboard**

Hasil pengujian menunjukkan bahwa dashboard Superset mampu menampilkan data cuaca secara interaktif dan real-time tanpa kendala.

BAB IV PENUTUP

4.1 Kesimpulan

Berdasarkan hasil pengembangan dan pengujian, dapat disimpulkan bahwa sistem Automated Data Pipeline data cuaca berhasil dibangun dan berjalan sesuai dengan tujuan. Sistem mampu mengotomatisasi proses pengambilan data, penyimpanan, transformasi, dan visualisasi data cuaca.

4.2 Saran

Saran untuk pengembangan selanjutnya antara lain:

- Menambahkan sumber data lain
- Mengimplementasikan sistem ke cloud
- Menambahkan analitik lanjutan