# ETL & Data Pipeline Modern

Dengan PostgreSQL, Python, Streamlit, dan Orchestration Tools

Fokus: Extract → Transform → Load + Scheduling + Monitoring + Automation

# Tujuan Pelatihan

### Konsep Modern

Memahami konsep ETL dan Data Pipeline modern secara mendalam.

### Implementasi PostgreSQL

Praktik implementasi ETL sederhana langsung di database PostgreSQL.

### Python Integration

ETL berbasis Python untuk integrasi API & batch job yang efisien.

### Incremental Loading

Memahami teknik streaming dan incremental loading data.

### Frontend Monitoring

Integrasi front-end ETL monitoring menggunakan Streamlit.

### Orchestration & Deploy

Pengenalan Airflow, Prefect, Luigi, dbt, serta best practice deployment.

# ETL: Konsep Inti

## Extract

Mengambil data dari berbagai sumber seperti API, CSV, SFTP, atau database eksternal lainnya untuk diproses lebih lanjut.

## Transform

Normalisasi, enrichment, agregasi data. Menggunakan SQL di PostgreSQL atau Python untuk transformasi logika berat.

## Load

Memasukkan data ke tabel target. Mencakup teknik Insert, Upsert/Merge, Partitioning, dan Incremental loading.

*"ETL adalah jantung operasional pemadanan data & integrasi antar sistem di pemerintahan."*

# Struktur Layer Data Pipeline

## Raw/Staging

Data mentah apa adanya tanpa modifikasi, tempat pendaratan pertama data.

## Transform

Query SQL, transformasi Python, tabel intermediate, dan materialized views.

## Serving Layer

Data final bersih yang siap digunakan untuk dashboard, API, dan analitik.

## Metadata

Menyimpan audit log, history job, dan status monitoring pipeline.

# ETL Pattern di PostgreSQL

## Teknik Umum

- **COPY:** Metode tercepat untuk bulk load data besar.
- **UPSERT:** `INSERT...ON CONFLICT DO UPDATE` untuk menangani duplikasi.
- **Partitioning:** Manajemen tabel besar untuk performa query.
- **Materialized View:** View fisik yang direfresh secara berkala.
- **FDW:** Foreign Data Wrapper untuk akses DB eksternal langsung.

PostgreSQL dapat menjadi "ELT engine" lengkap tanpa sistem tambahan.

# Extract: Python → Postgres


## Contoh Pipeline

1. Request data dari API (misal: DikTi, Dukcapil).
2. Simpan sementara ke format JSON atau CSV.
3. Load ke tabel staging di PostgreSQL.
4. Menambahkan timestamp otomatis.
5. Job dijadwalkan per jam atau per hari.

```
import requests, pandas, psycopg2 # Tools utama: # requests (API), pandas (Data), # psycopg2/SQLAlchemy (DB)
```

# Transform: SQL-Based

## Teknik Transformasi

Memanfaatkan kekuatan engine database untuk pemrosesan data:

- Aggregation & Grouping
- Join antar sumber data
- Window functions untuk analisis kompleks
- Konversi JSON ke Relational
- Geospatial processing (PostGIS)
- Incremental transform (WHERE updated_at > last_run)

## Contoh Implementasi

Menggunakan Materialized View untuk hasil query kompleks:

```
CREATE MATERIALIZED VIEW mv_data_final AS
SELECT s.id, s.nama, r.kategori, COUNT(*) as
total_transaksi FROM staging_transaksi s JOIN
ref_institusi r ON s.kode_inst = r.kode GROUP BY 1, 2,
3 WITH DATA;
```

# Transform: Python-Based

## Mengapa Python?

**Heavy Logic:** Cocok untuk algoritma kompleks yang sulit di SQL.

**Rich Libraries:** Pandas, Numpy, Polars tersedia.

**Integrasi:** Mudah menghubungkan API dengan Database.

**Paralelisme:** Bisa multithreading untuk performa.

## Pattern Umum

```
      # 1. Load dari Postgres df =
pd.read_sql("SELECT * FROM staging", conn) # 2.
Transform Logic df['clean_name'] =
df['name'].apply(clean_func) df['status'] =
calculate_complex_status(df) # 3. Write Back
(Batch/Merge) df.to_sql('final_table', engine,
if_exists='append', index=False) # 4. Log job status
log_success("Job finished")
```

# Load: Teknik Pengisian ke PostgreSQL

**COPY FROM STDIN:**

Teknik paling cepat untuk memasukkan data, menghindari overhead INSERT baris per baris.

**Bulk & Batched INSERT:**

Mengirim data dalam batch (misal 500-5000 baris) untuk efisiensi jaringan dan transaksi.

**UPSERT (Merge):**

Memperbarui data yang sudah ada dan memasukkan data baru dalam satu perintah atomik.

**Incremental Load:**

Hanya memuat data yang berubah menggunakan "watermark" timestamp.

```
-- Contoh Incremental Query SELECT * FROM api_data WHERE updated_at > last_processed_timestamp;
```

# ETL Monitoring dengan Streamlit

## Fitur Dashboard

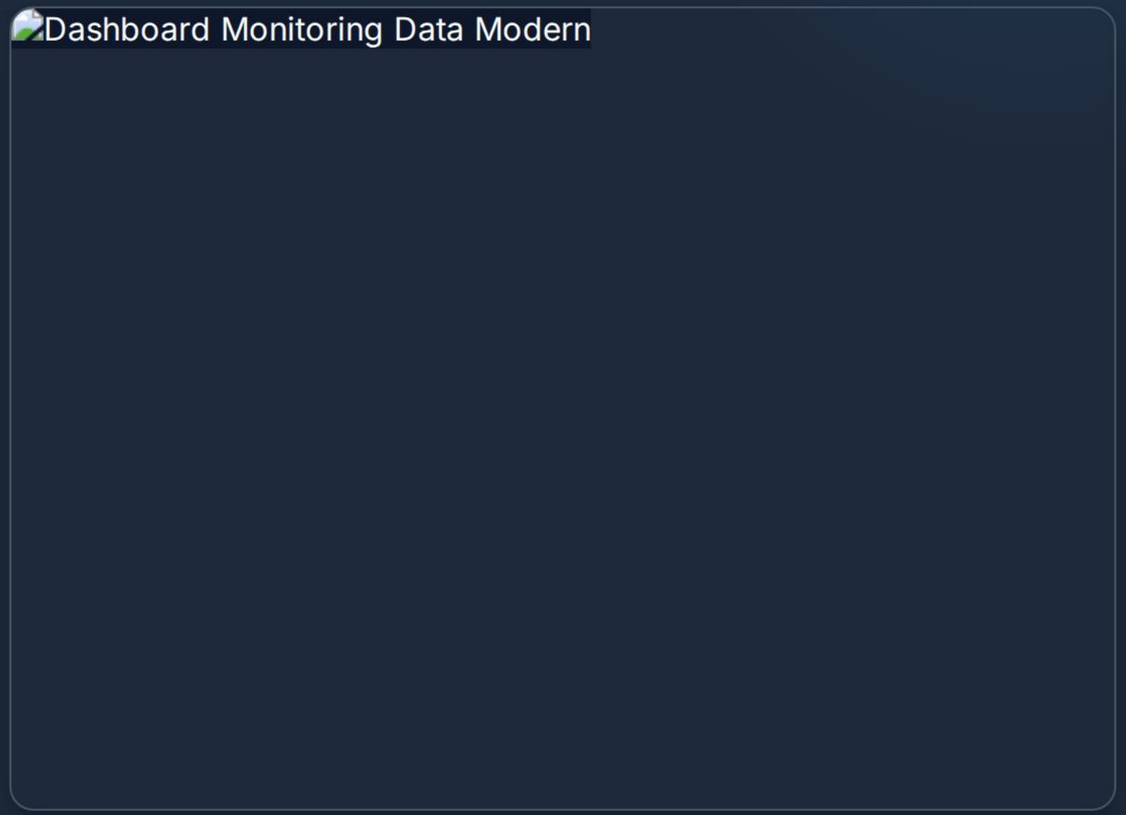
Streamlit memudahkan pembuatan UI interaktif untuk Data Engineer:

`st.button()` Trigger manual job ETL.

`st.dataframe()` Review data staging & transform.

`st.status()` Monitoring langkah-langkah pipeline.

`st.progress()` Visualisasi progress job berjalan.



# Orchestration Tools

## Apache Airflow

Enterprise-grade, Scheduler + DAG, Monitoring UI lengkap. Standar industri untuk ETL skala besar.

## Prefect

Modern, mudah untuk tim kecil, support hybrid local/cloud execution, logging otomatis.

## Luigi

Python native orchestration, sangat cocok untuk task dependency chain skala kecil-menengah.

## dbt

Data Build Tool. Fokus pada transformasi SQL, lineage graph, dan dokumentasi otomatis.

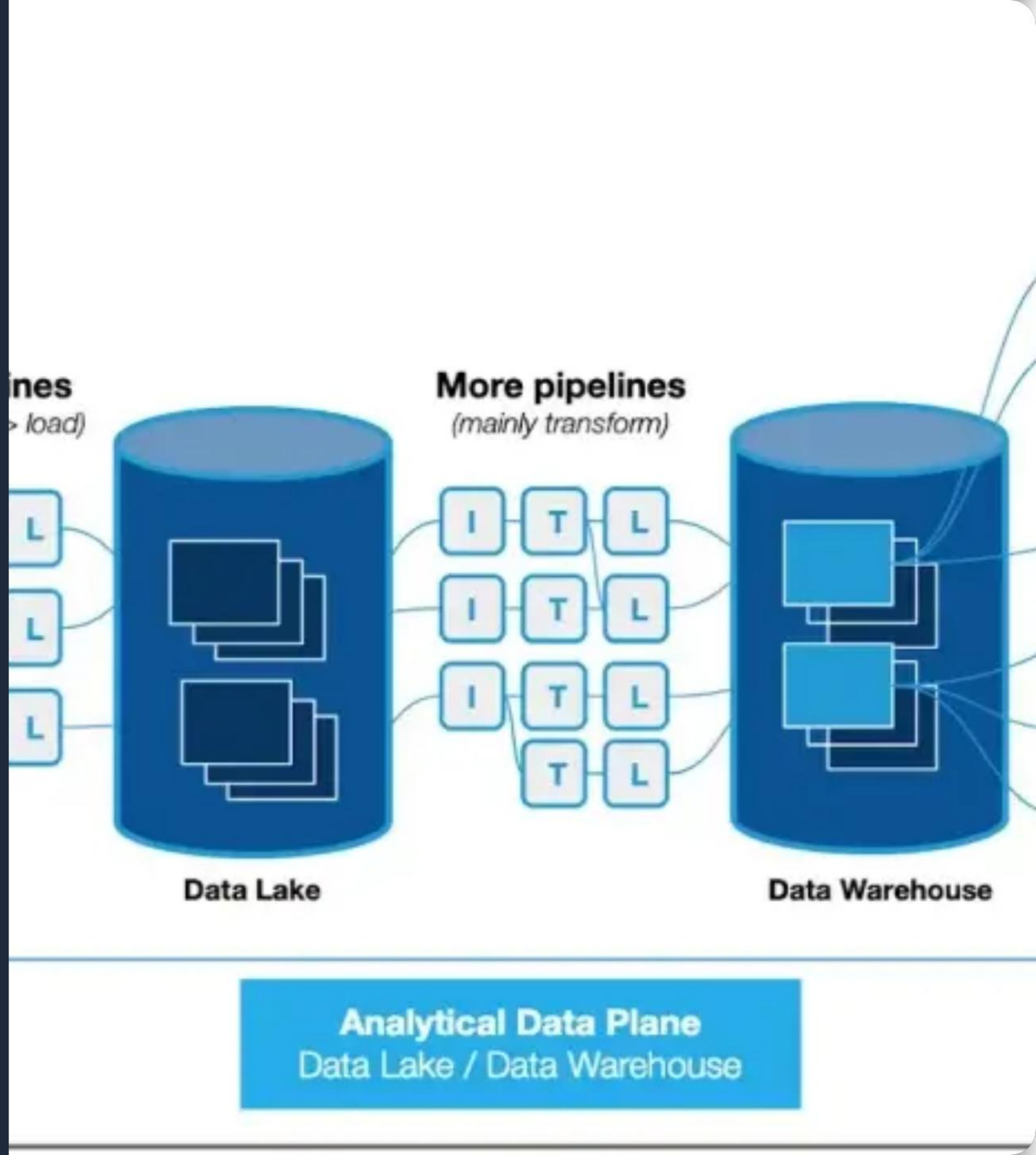
# Arsitektur End-to-End

## Alur Data Instansi Pemerintah

- **Extract:** API DikTi/Dukcapil, SFTP Bulk.
- **Load Staging:** Raw Table via COPY.
- **Transform:** SQL Parsing JSON, Python Logic, Join Reference.
- **Serving:** Materialized Views untuk Analitik.

**Orchestration:** Menggunakan Airflow/Prefect untuk jadwal, notifikasi, dan retry logic.

**Monitoring:** Dashboard Streamlit untuk audit history.



# Best Practice ETL

### Idempotent Job

Job harus bisa dijalankan ulang berkali-kali tanpa merusak atau menduplikasi data.

### Watermark Timestamp

Gunakan timestamp terakhir (last_updated) untuk efisiensi incremental load.

### Layer Separation

Pisahkan secara tegas: Raw → Staging → Transform → Final Serving.

### Metadata Audit

Selalu simpan metadata job (start_time, end_time, status) untuk keperluan audit.

### Performance

Manfaatkan cache memori (Redis/Python Dict) dan concurrent refresh materialized view.

### Documentation

Dokumentasikan setiap model transformasi data (dbt sangat direkomendasikan).

# Image Sources

https://media2.dev.to/dynamic/image/width=1000,height=420,fit=cover,gravity=auto,format=auto/https%3A%2F%2Fdev-to-uploads.s3.amazonaws.com%2Fuploads%2Farticles%2Flwpc2cy3a32rtdqss1ok.jpg

Source: dev.to

---

https://daxg39y63pxwu.cloudfront.net/images/blog/python-for-data-engineering/image_634543498616531296572261.png

Source: www.projectpro.io

---

https://images.stockcake.com/public/c/9/a/c9a7a5e8-a217-445c-bdf0-e740903f156c_large/modern-data-dashboard-stockcake.jpg

Source: stockcake.com

---

https://static.wixstatic.com/media/904900_03cec6a515434918ad8db97814d98a5c~mv2.png/v1/fill/w_1014,h_518,al_c,q_90,enc_avif,quality_auto/904900_03cec6a515434918ad8db97814d98a5c~mv2.png

Source: www.timeplus.com