LEARNING PROGRESS REVIEW

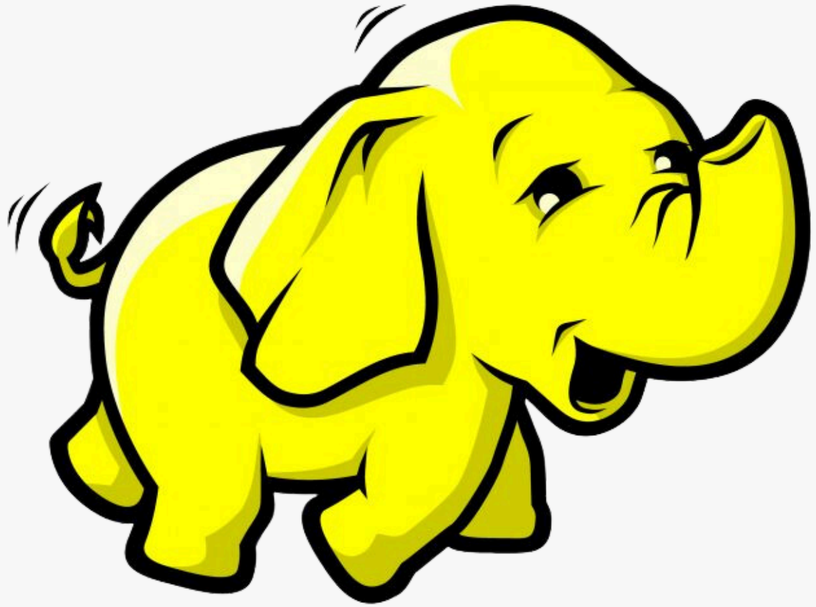
#5

OUR MINI PROJECT, HADOOP, AND SPARK

FAST TRACK DATA ENGINEER SCHOLARSHIP
BY
DIGITAL SKOLA

GARAP RENDANG TEAM

- Suhendar Aditya
- Chandra Trio Pamungkas
- Aryo Putra
- Kurniaman Andreas Zega
- Ahmad Fayyadh



HADOOP

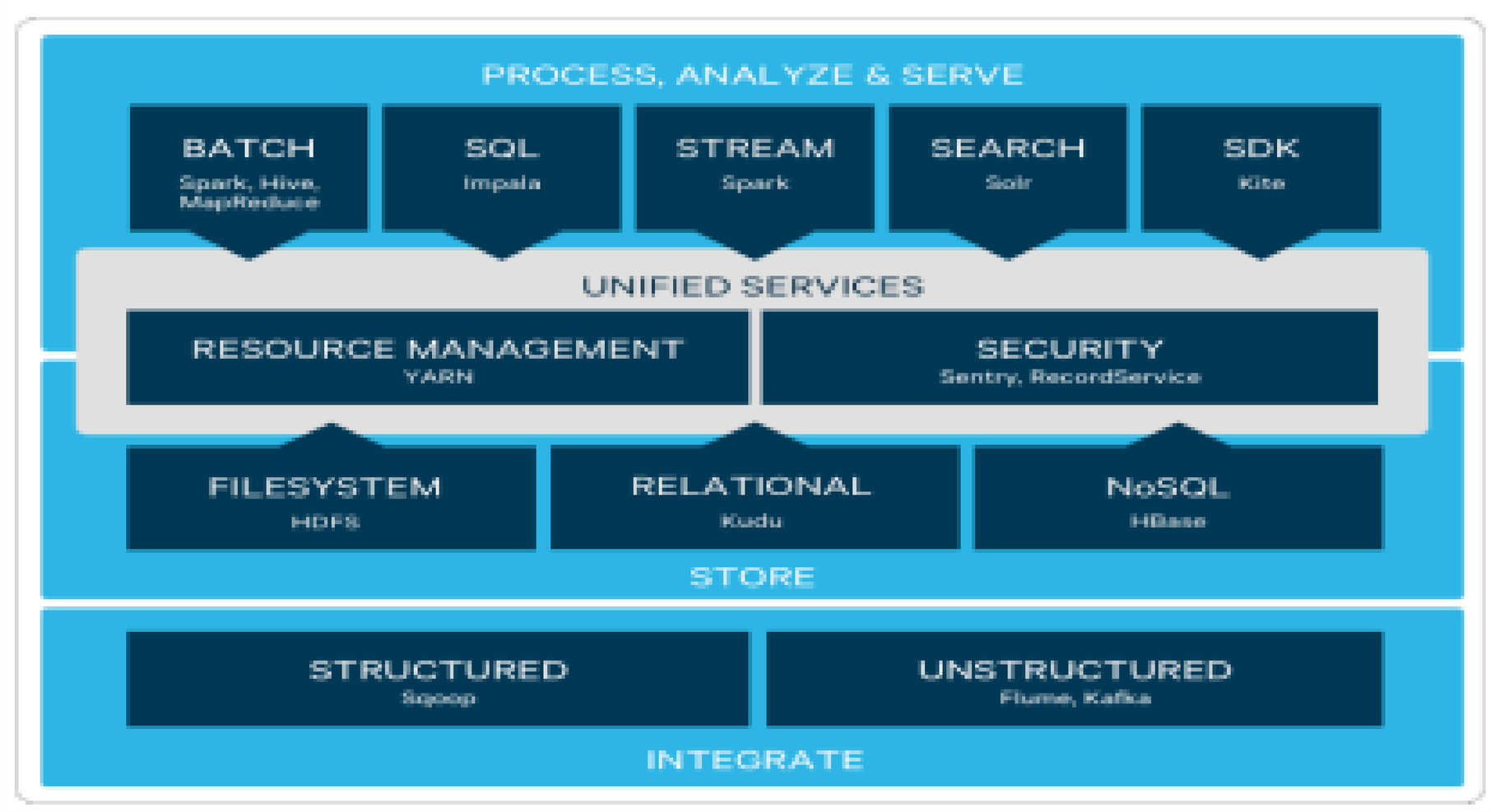
HADOOP adalah kerangka kerja sumber terbuka yang memungkinkan penyimpanan dan pemrosesan data dalam jumlah besar:

- Menggunakan penyimpanan terdistribusi dan pemrosesan paralel
- Memungkinkan pengelompokan beberapa komputer untuk menganalisis data secara paralel
- Dapat memproses data mulai dari ukuran gigabita hingga petabita
- Memiliki fitur-fitur seperti skalabilitas tinggi, toleransi kesalahan, dan kinerja tinggi

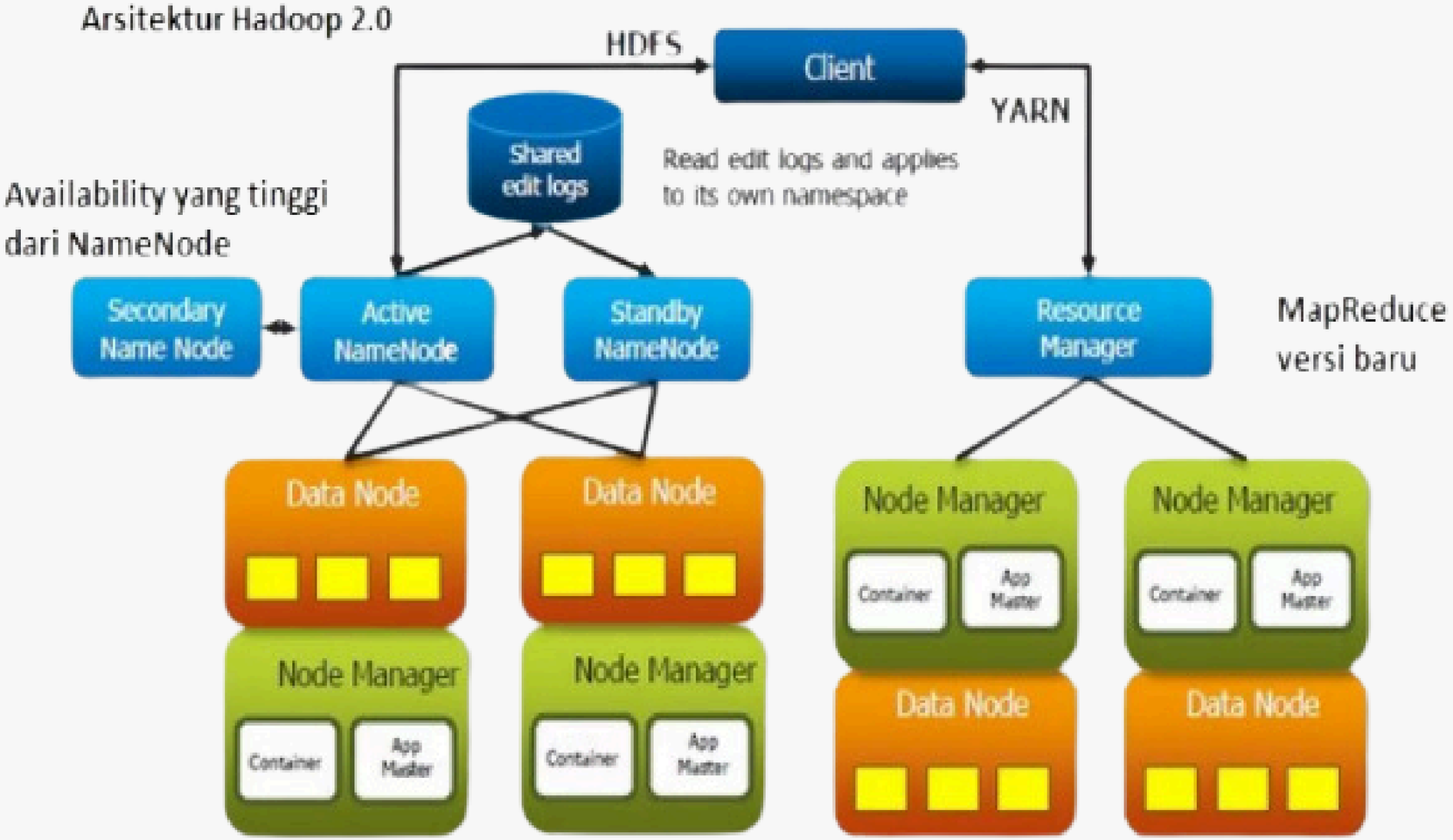
Hadoop menggunakan model pemrograman sederhana dan memiliki empat modul utama, yaitu:

1. **HDFS** (Hadoop Distributed File System) yaitu sebuah sistem file yang terdistribusi
2. **YARN** (Yet Another Resource Negotiator) merupakan resource manager yang bertanggung jawab mengelola resources dalam *cluster* dan *scheduling*
3. **MapReduce** yaitu sebuah model programming/Algoritma untuk pengolahan data skala besar dengan komputasi secara terdistribusi
4. **Hadoop Common** yaitu berisi *library/utility* untuk modul hadoop

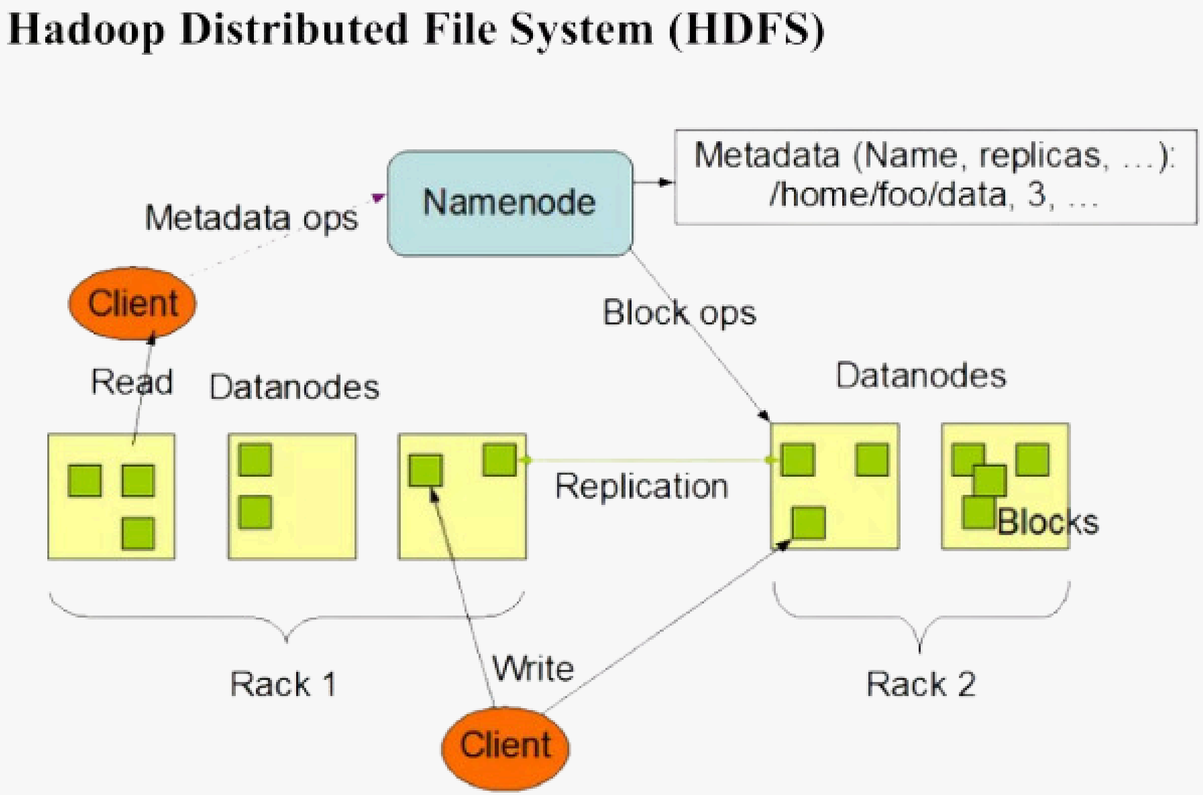
EKOSISTEM HADOOP



ARSITEKTUR HADOOP

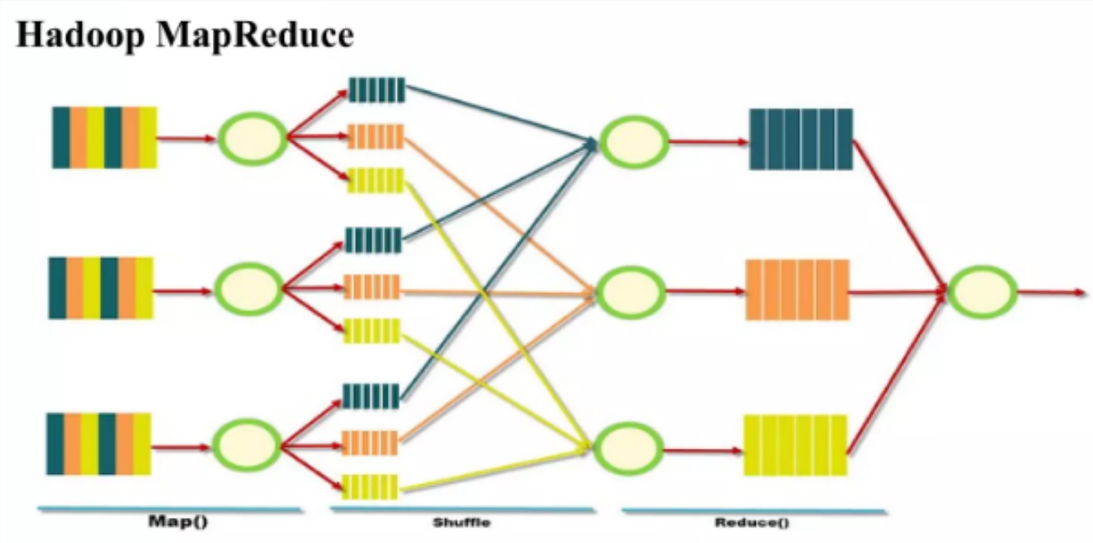


CARA KERJA HADOOP



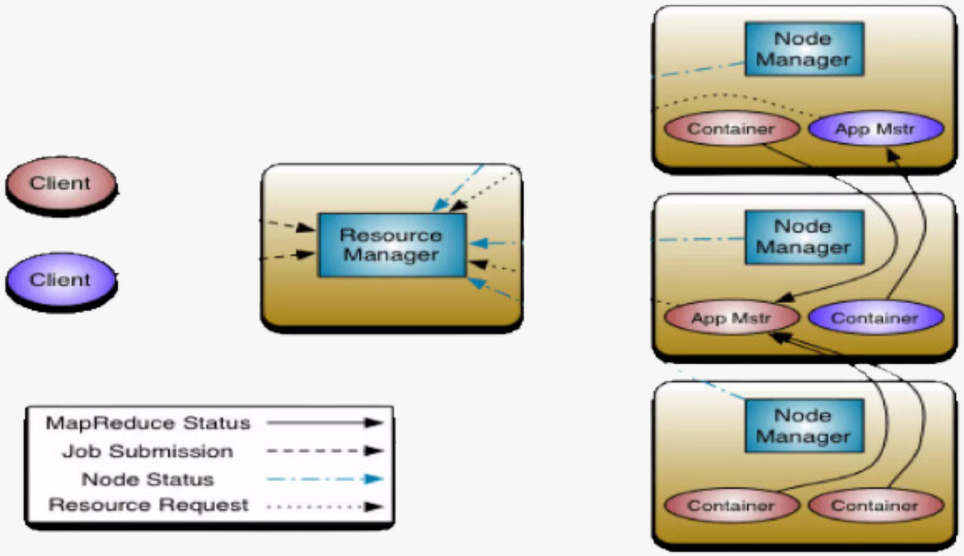
- Hadoop Distributed File System adalah sebuah sistem berkas terdistribusi dengan high-availability yang dapat menyimpan data pada mesin komoditas, digunakan untuk menyediakan bandwidth sangat tinggi yang di agregasi ke semua cluster (node).
- Berkas dibagi menjadi blok data dengan panjang yang baku dan didistribusikan secara redundan (berlebihan) pada simpul (node) yang berpartisipasi
- Sebuah kluster HDFS terdiri dari NameNode, yang mengelola metadata dari kluster, dan DataNode yang menyimpan data/file
- File dan direktori diwakili pada NameNode oleh inode. Inode menyimpan atribut seperti permission, modifikasi dan waktu akses, atau kuota namespace dan diskspace

CARA KERJA HADOOP



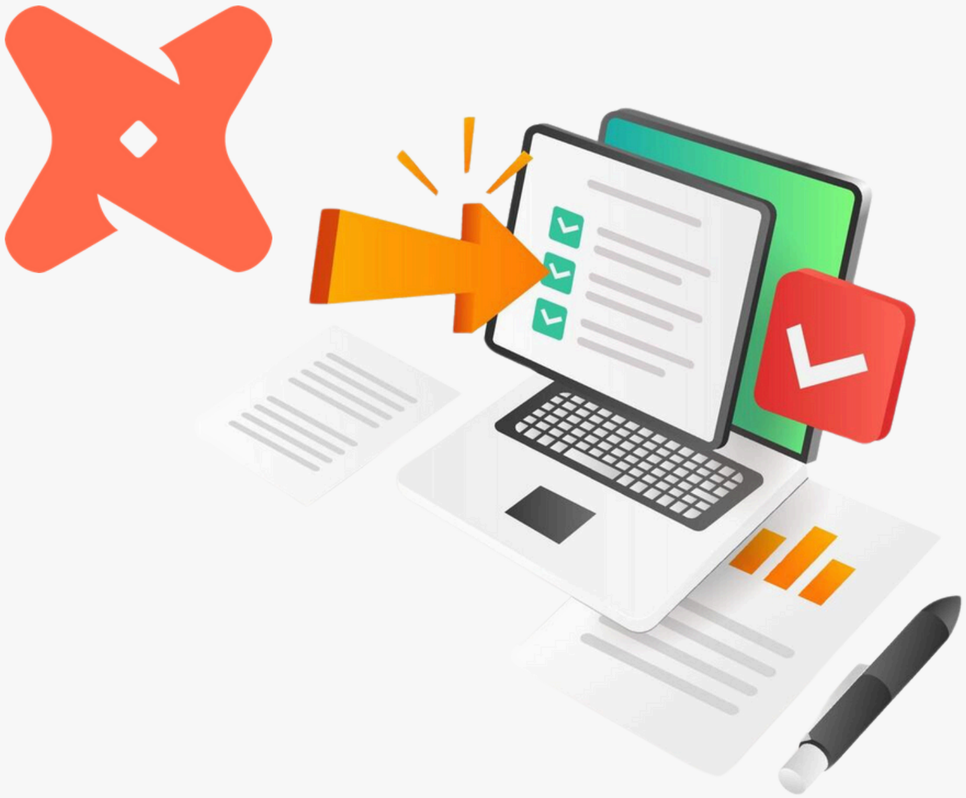
MapReduce bertugas membagi data yang besar ke dalam potongan lebih kecil dan mengatur mereka kedalam bentuk tupel untuk pemrosesan paralel. Tupel adalah kombinasi antara key dan value-nya, dapat disimbolkan dengan notasi : "(kl, vl)". Dengan pemrosesan bersifat paralel tersebut, tentunya akan meningkatkan kecepatan dan keandalan komputasi pada sistem klustering.

CARA KERJA HADOOP



Tujuan awal YARN adalah untuk memisahkan dua tanggung jawab utama dari Job Tracker atau Task Tracker menjadi beberapa entitas yang terpisah.

- **Global Resource Manage**r di node master, yang berfungsi mengatur semua resource yang digunakan aplikasi dalam sistem
- **Application Master** di setiap aplikasi, yang berfungsi untuk negosiasi resource dengan Resource Manager dan kemudian bekerja sama dengan Node Manager untuk mengeksekusi dan memonitor tasks
- **Node Manager** di Agen-Framework setiap node slave, yang bertanggung jawab terhadap Container, dengan memantau penggunaan resource/sumber daya dari container (cpu, memori, disk, jaringan) dan melaporkannya pada Resource Manager.
- **Container** di setiap aplikasi yang jalan di Node Manager, sebagai wadah penyimpanan data/file

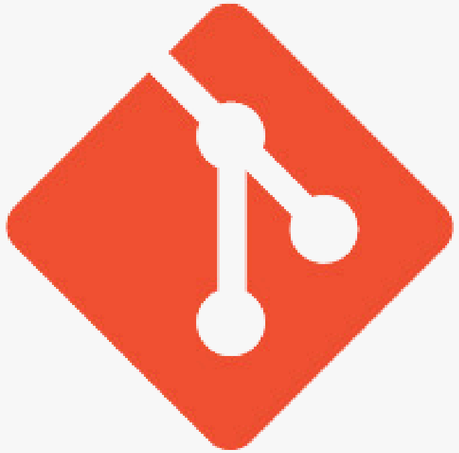


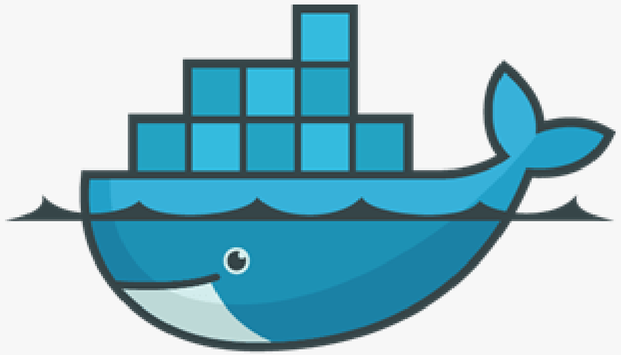
MINI PROJECT

DBT HANDS ON

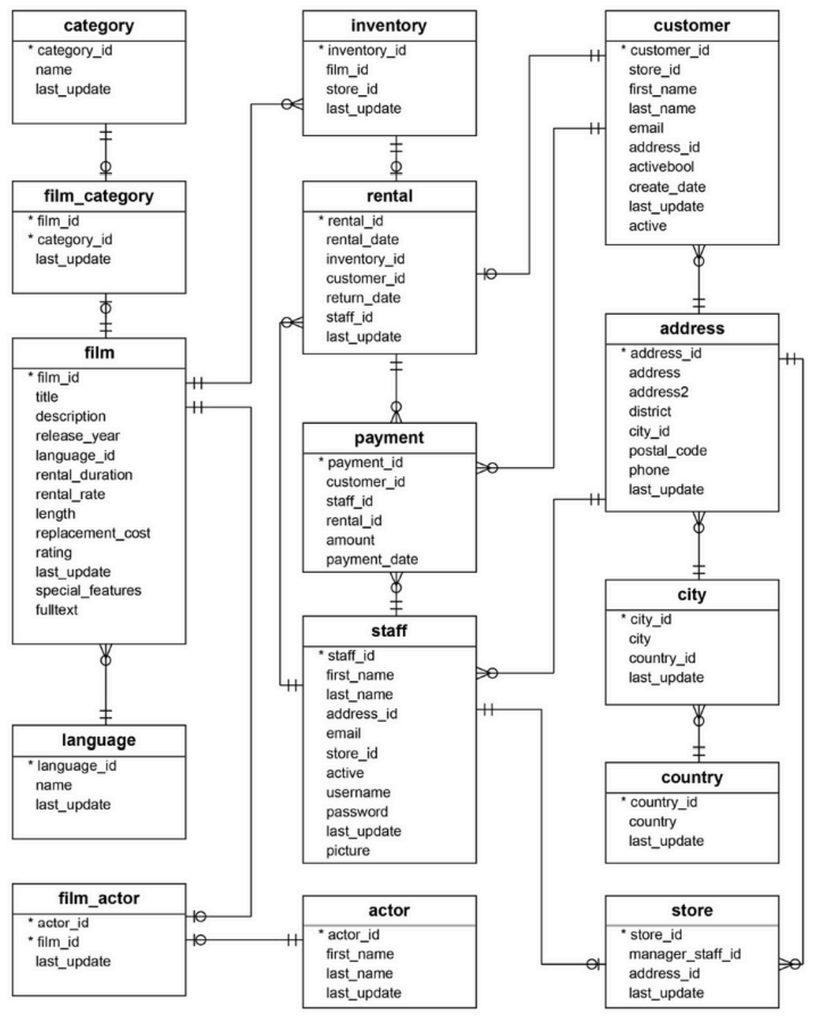
Tools


GARAP RENDANG
Grup Andalan Rakitan Pipeline
Record Engineer Ngolah Data
Anti Gagal





ERD of The Dataset



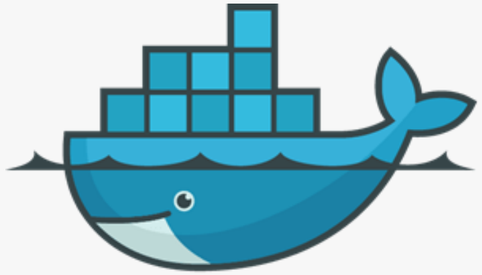
Fact Table:

payment

Dim Table:

- actor
- address
- customer
- film_actor
- film
- inventory
- payment
- rental
- staff

Run PostgreSQL



Pakai code ini jika ingin menjalankan lewat Docker:

```
docker exec -it <postgres container name> bash
create database data_warehouse
# quit database
\q

# Run in terminal
pg_restore -U postgres -d data_warehouse /dvdrental
```

Run PostgreSQL

Pakai code ini jika ingin menjalankan tidak dengan Docker:

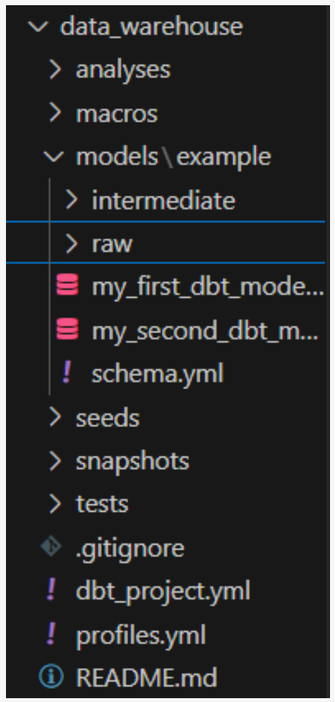
```
psql -U postgres
create database data_warehouse
# quit database
\q

# Run in terminal
pg_restore -U postgres -d data_warehouse /path/to/dvdrental
```

DBT Configurations

Konfigurasi DBT dengan dbt init, buat profiles.yaml, dan dbt debug untuk pengecekan koneksi

```yaml
data_warehouse:
  outputs:
    dev:
      dbname: data_warehouse
      host: localhost
      pass: postgres
      port: 5433
      schema: dbt_dev
      threads: 1
      type: postgres
      user: postgres
    prod:
      dbname: data_warehouse
      host: localhost
      pass: postgres
      port: 5433
      schema: dbt
      threads: 1
      type: postgres
      user: postgres
  target: dev
```

`dbt init`



`dbt debug`

```
13:38:38  Running with dbt=1.6.0
13:38:38  dbt version: 1.6.0
13:38:38  python version: 3.12.6
13:38:38  python path: C:\Users\user\Desktop\ftde-project2\venv\Scripts\python.exe
13:38:38  os info: Windows-11-10.0.26100-SP0
13:38:38  Using profiles dir at C:\Users\user\Desktop\ftde-project2\data_warehouse
13:38:38  Using profiles.yml file at C:\Users\user\Desktop\ftde-project2\data_warehouse\profiles.yml
13:38:38  Using dbt_project.yml file at C:\Users\user\Desktop\ftde-project2\data_warehouse\dbt_project.yml
13:38:38  adapter type: postgres
13:38:38  adapter version: 1.6.0
13:38:38  Configuration:
13:38:38    profiles.yml file [OK found and valid]
13:38:38    dbt_project.yml file [OK found and valid]
13:38:38  Required dependencies:
13:38:39    - git [OK found]

13:38:39  Connection:
13:38:39    host: localhost
13:38:39    port: 5433
13:38:39    user: postgres
13:38:39    database: data_warehouse
13:38:39    schema: dbt_dev
13:38:39    connect_timeout: 10
13:38:39    role: None
13:38:39    search_path: None
13:38:39    keepalives_idle: 0
13:38:39    sslmode: None
13:38:39    sslcert: None
13:38:39    sslkey: None
13:38:39    sslrootcert: None
13:38:39    application_name: dbt
13:38:39    retries: 1
13:38:39  Registered adapter: postgres=1.6.0
13:38:39    Connection test: [OK connection ok]
```

Data Modelling

1. Create database “data_warehouse”

```
create database data_warehouse;
```

2. Create schema:
   - Raw (raw data)
   - Intermediate (fact and dim table)
   - Gold (mart)

```
\c data_warehouse

create schema dbt_dev_raw;
create schema dbt_dev_intermediate;
create schema dbt_dev_mart;
```

3. Create sources.yml

```yaml
version: 2

sources:
  - name: public
    database: data_warehouse
    schema: public
    tables:
      - name: actor
      - name: address
      - name: customer
      - name: film_actor
      - name: film
      - name: inventory
      - name: payment
      - name: rental
      - name: staff
```

Input nama-nama table yang akan ditarik dari source kedalam sources.yml

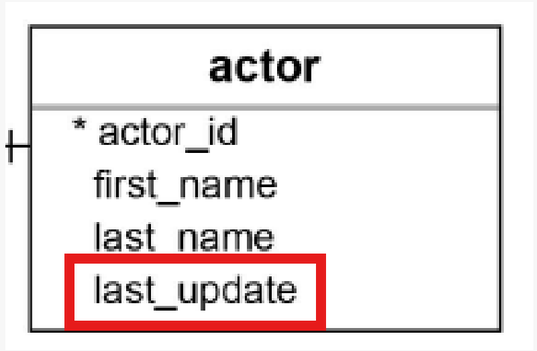
Data Modelling

4. Create raw_table.sql

data_warehouse > models > example > raw > raw_payment.sql

```sql
{{ config(materialized='table', schema='raw') }}

SELECT
    *
FROM {{ source('public', 'payment') }}
```

Selesai semua raw_table, lanjut ke intermediate_table dengan menyeleksi kolom yang dibutuhkan untuk menjawab pertanyaan di tahap mart_table



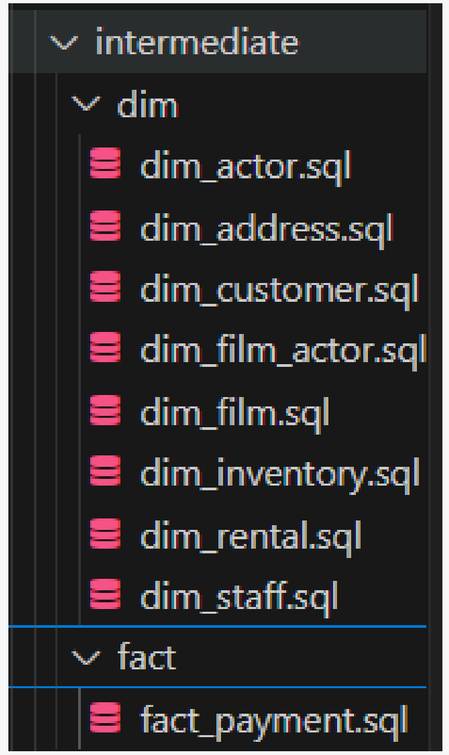
Terlihat kolom last_update tidak masuk karena tidak dibutuhkan

5. Create intermediate table queries

data_warehouse > models > example > intermediate > dim > dim_actor.sql

```sql
{{ config(materialized='table', schema='intermediate') }}

SELECT
    actor_id,
    first_name,
    last_name
FROM {{ ref('raw_actor') }}
```

Referensinya tidak lagi dari source, melainkan dari raw yang sudah ditarik sebelumnya



Data Modelling

6. Create mart_table untuk menjawab pertanyaan-pertanyaan dibawah ini:
   - How many monthly total revenue ?
   - What is the best selling film ?
   - Who is the actor who plays the most roles in films ?

data_warehouse > models > example > mart > mart_best_selling_film.sql

```sql
-- What is the best selling film ?

{{ config(materialized='table', schema='mart') }}

SELECT f.title, SUM(p.amount) AS total_revenue
FROM {{ ref('dim_film') }}  f
JOIN {{ ref('dim_inventory') }} i ON f.film_id = i.film_id
JOIN {{ ref('dim_rental') }}  r ON i.inventory_id = r.inventory_id
JOIN {{ ref('fact_payment') }} p ON r.rental_id = p.rental_id
GROUP BY f.title
ORDER BY total_revenue DESC
LIMIT 1
```

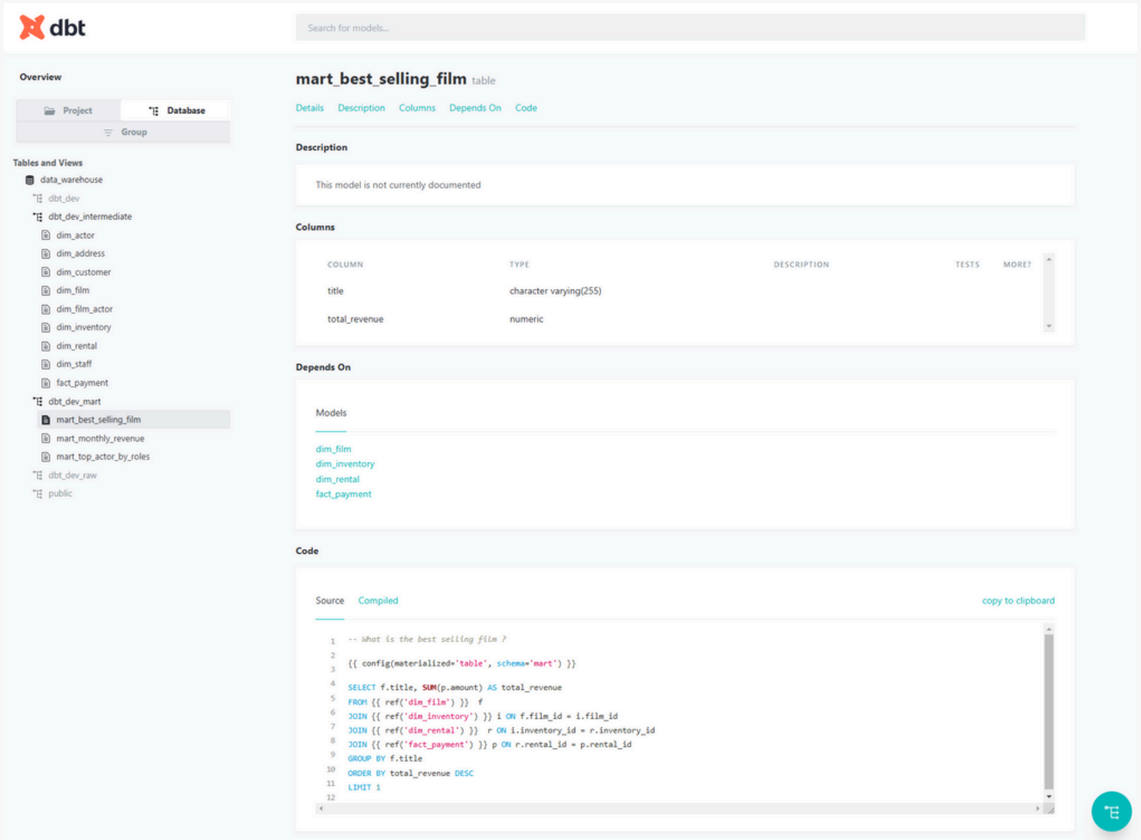
Contoh query untuk menjawab pertanyaan pertama

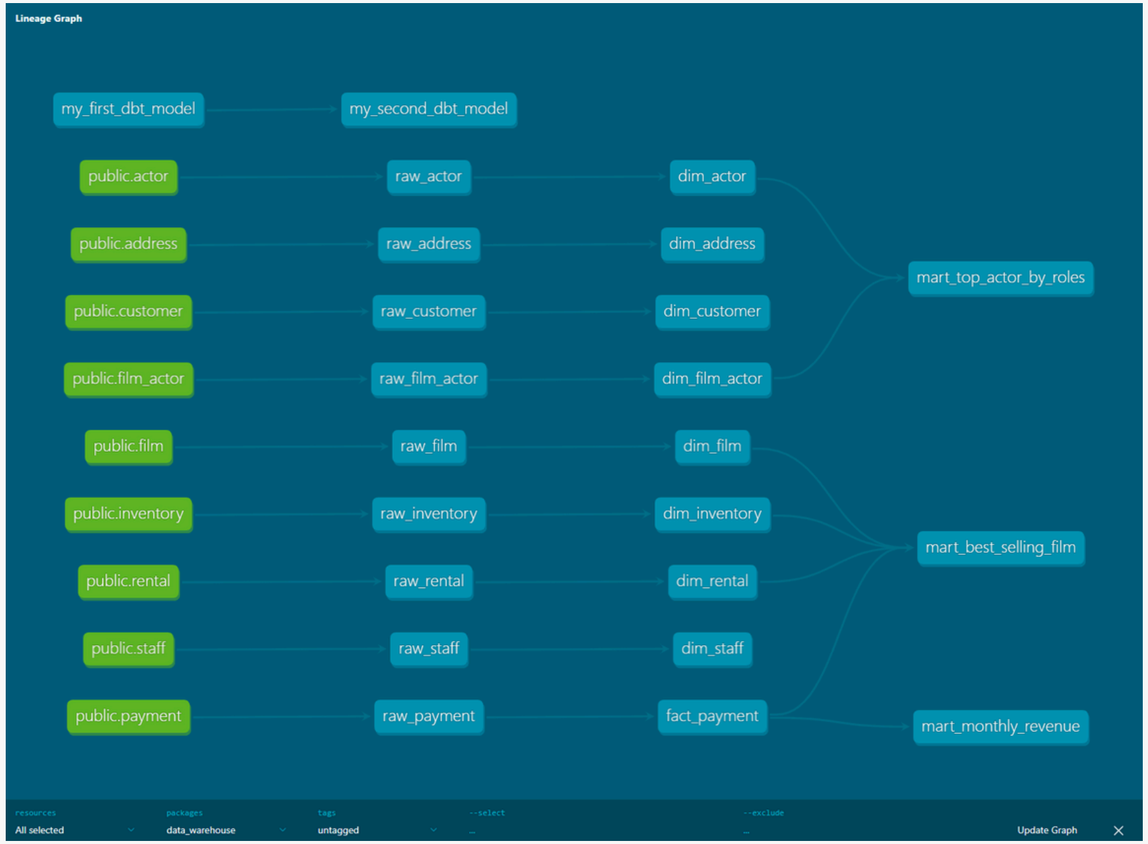
Output:

A-Z title	123 total_revenue
Telegraph Voyage	215.75

DBT UI

Selain dari hasil dan query kita juga dapat melihat hubungan/flow dengan DBT UI






APACHE
Spark

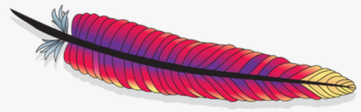
SPARK


GARAP RENDANG
Grup Andalan Rakitan Pipeline
Record Engineer Ngolah Data
Anti Gagal

Apache Spark adalah unified computing engine dan set of libraries yang dirancang untuk memproses data secara paralel pada computer clusters. Spark mendukung berbagai bahasa pemrograman seperti Python, Java, Scala, R, serta dapat digunakan untuk berbagai tugas seperti SQL, streaming, hingga pembelajaran mesin (machine learning.

ARSITEKTUR SPARK

- **Cluster:** Spark bekerja pada clusters komputer untuk menangani volume data besar yang tidak dapat ditangani oleh satu mesin saja. Spark menggunakan cluster manager (Standalone, YARN, Apache Mesos) untuk mengatur dan menjalankan tugas.
- **Driver Process & Executor Process**: Driver mengelola aplikasi Spark, sedangkan Executor menjalankan tugas yang diberikan.



API TERSTRUKTUR SPARK

Spark menyediakan API tingkat tinggi untuk memanipulasi data, seperti DataFrame, Dataset, dan SQL. API ini mendukung berbagai bahasa dan menggunakan pendekatan berbasis schema untuk efisiensi.

TRANSFORMASI DAN EVALUASI

- **Transformasi**: Operasi pada data yang bersifat lazy evaluation (dievaluasi hanya ketika diperlukan oleh action).
- **Action**: Operasi untuk memicu komputasi, seperti count() atau menulis data ke sumber eksternal.

TRANSFORMASI DAN EVALUASI

- **Transformasi**: Operasi pada data yang bersifat lazy evaluation (dievaluasi hanya ketika diperlukan oleh action).
- **Action**: Operasi untuk memicu komputasi, seperti count() atau menulis data ke sumber eksternal.
- **Dependencies:**
    - Narrow: Satu input partition menghasilkan satu output.
    - Wide: Banyak input partition menghasilkan banyak output, membutuhkan shuffle.

ANALITIK DATA DENGAN SPARK

- Spark dapat membaca data dari berbagai sumber (CSV, JSON, Parquet) dan mendukung manipulasi tipe data sederhana (boolean, string, number) hingga kompleks (structs, arrays, maps).
- **Null Handling:** Null lebih optimal dibanding nilai kosong, dengan opsi untuk menghapus (drop) atau mengisi (fill).

OPERASI LANJUTAN

- **Agregasi**: Digunakan untuk meringkas data numerik melalui fungsi seperti sum(), avg(), atau grouping berdasarkan kategori.
- **Joins**: Spark mendukung berbagai jenis join, seperti inner, outer, left, dan right joins untuk menggabungkan data dari beberapa sumber.

“Spark dirancang untuk mendukung kebutuhan analitik data berskala besar dengan pendekatan paralel dan terstruktur, menjadikannya alat yang efisien untuk pemrosesan **big data”**

THANK YOU

ANY FEEDBACKS?