Pertemuan 1 — Data Engineering 101

Dari *raw data* ke keputusan: *lifecycle*, *hot* dan *cold*, *security*, dan *governance*

Hendri Karisma

Semester Ganjil 2026/2027

Program Studi Teknik Informatika, STMIK Tazkia

Perkenalan



Hendri Karisma

VP Engineering, jejakin.com
Produk yang ditangani:

- Emission and ESG analytics and calculation
- MRV (*Measurement, Reporting, Verification*) untuk ESG dan proyek karbon
- *Marketplace* proyek ESG, karbon, air, dan REC

Lecturer, STMIK Tazkia
Mengajar: Machine Learning, Artificial Intelligence, Data Engineering, Distributed System, Research Methodology.

Research area
Secure and Sustainable Computation, Artificial Intelligence, Quantum Computing.

Situs hendrikarisma.my.id
GitHub github.com/situkangsayur
LinkedIn linkedin.com/in/hendri-karisma-736a0a39
X @infoHendri Instagram @karism4_

Alur pertemuan hari ini

1. Dari data ke keputusan: organisasi *data-driven*
2. Linimasa: dari *database* relasional ke *lakehouse*
3. Apa itu *data engineering*, dan siapa mengerjakan apa
4. *Data engineering lifecycle* dan *undercurrents*
5. *Big data*: kapan data disebut besar
6. *Hot*, *warm*, dan *cold*: data tidak sama nilainya
7. Keamanan dan GRC: data adalah amanah
8. Studi kasus semester: bank syariah digital
9. Python, Scala, Rust, dan perangkat: kapan memakai apa
10. Kurikulum acuan dan pembagian dengan Sistem Terdistribusi
11. Rencana semester, penilaian, dan aturan main

Sasaran pertemuan ini

Anda dapat menjelaskan pekerjaan seorang *data engineer* dengan satu kalimat, menggambar *lifecycle* data dari *source* sampai keputusan, dan membedakan data yang harus diproses **sekarang** dari data yang **boleh menunggu**.

Dari data ke keputusan

Satu hari di sebuah bank syariah digital

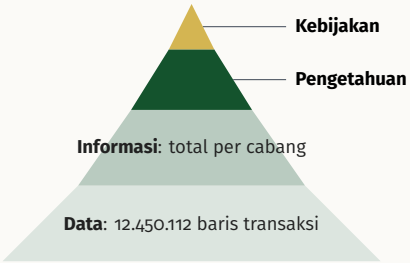
Setiap aktivitas nasabah meninggalkan jejak data, dan setiap jejak menjawab pertanyaan yang berbeda, dengan tenggat yang berbeda pula.

Pertanyaan	Siapa yang bertanya	Tenggat jawaban
Transaksi ini penipuan atau bukan?	Sistem anti-*fraud*	**< 1 detik**
Berapa saldo rekening saya?	Nasabah di aplikasi	milidetik
Berapa pembiayaan murabahah yang cair hari ini?	Kepala cabang	menit sampai jam
Produk mana yang paling diminati bulan ini?	Tim produk dan pemasaran	harian
Laporan bulanan kepada regulator	Divisi kepatuhan	bulanan
Bukti transaksi 8 tahun lalu untuk audit	Auditor, aparat hukum	hari

Ringkasan

Data yang sama mengalir ke banyak keputusan. Tugas *data engineering* adalah membuat data itu **tersedia, benar, aman, dan tepat waktu** untuk tiap keputusan, dengan *cost* yang masuk akal.

Piramida DIKW: data saja belum berarti apa-apa



- **Data**: fakta mentah, belum diberi konteks.
- **Informasi**: data yang sudah diringkas dan diberi konteks (“transaksi cabang Bogor turun 18% pekan ini”).
- **Pengetahuan**: pola dan sebabnya (“penurunan terjadi sejak ATM mitra diganti”).
- **Kebijakan** (*wisdom*): tindakan yang diambil dan diukur hasilnya.

Data-driven vs HiPPO

Organisasi *data-driven* memutuskan berdasarkan bukti yang terukur, bukan berdasarkan *Highest Paid Person’s Opinion*. Syaratnya: datanya ada, dapat dipercaya, dan sampai tepat waktu.

Mengapa banyak inisiatif *data-driven* gagal

Gejalanya

- Dua *dashboard* menampilkan angka “nasabah aktif” yang berbeda
- Laporan harian baru siap pukul 14.00, padahal rapat pukul 09.00
- *Data scientist* menghabiskan sebagian besar waktunya membersihkan data
- Model *fraud* bagus saat diuji, buruk di produksi
- Data nasabah bocor dari *bucket* yang tidak sengaja dibuka publik

Akar masalahnya

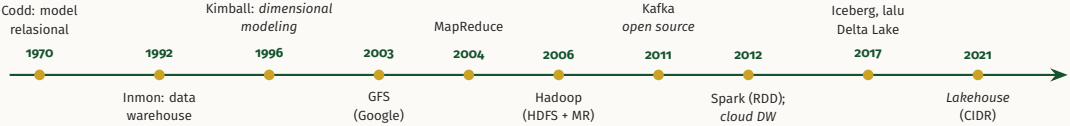
- Definisi *metric* tidak disepakati → *semantic layer*, *data governance*
- *Pipeline* lambat dan rapuh → *orchestration*, *idempotency*
- *Data quality* tidak diuji → *data quality*, *data contract*
- *Training data* dan data produksi berbeda jalur → arsitektur *hot*/*cold*
- Akses tidak dikendalikan → *security* dan GRC

Ringkasan

Hampir semua masalah di kiri adalah masalah *engineering* yang muncul sebelum analisis dimulai. Masalah itulah yang dipelajari di mata kuliah ini.

Linimasa singkat

Dari *database* relasional ke *lakehouse*



Era *warehouse*

Data dari banyak sistem disalin ke satu *database* analitik yang terstruktur. Mahal, rapi, lambat berubah.

Era *big data*

Data terlalu besar dan beragam untuk satu mesin: simpan di banyak mesin murah, proses di tempat datanya.

Era *cloud* dan *streaming*

Compute dan *storage* dipisah, data mengalir terus, dan *lakehouse* menyatukan kerapian *warehouse* dengan murahnya *lake*.

Apa itu *data engineering*

Definisi yang kita pakai

Reis dan Housley (2022), disarikan

Data engineering adalah membangun, menjalankan, dan merawat *systems and processes* yang mengambil *raw data* lalu menghasilkan data yang *high-quality* dan konsisten untuk *downstream use cases* seperti *analytics* dan *machine learning*.

Tiga kata kunci:

- ***Systems and processes***: berjalan otomatis dan berulang, bukan skrip sekali jalan
- ***Raw*** $\rightarrow$ ***high-quality data***: *cleaning*, *integration*, *modeling*
- ***Downstream use cases***: yang memakai hasilnya adalah orang (*analyst*, manajemen) dan model ML yang mengambil keputusan

Yang bukan *data engineering*

Menulis satu *query* untuk laporan sekali pakai. Melatih model. Membuat *dashboard*. Ketiganya **memakai** hasil *data engineering*; tanpa fondasi itu, ketiganya rapuh.

Siapa mengerjakan apa

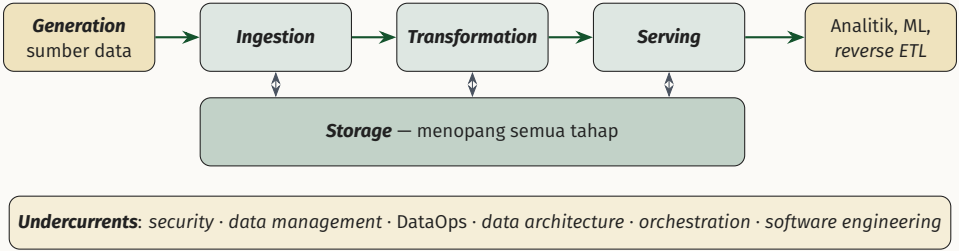
Peran	Fokus	*Output* khas
Data engineer	Membangun dan mengoperasikan *pipeline* dan platform data	*Pipeline* andal, *lakehouse*, *stream event*
Analytics engineer	Memodelkan data siap pakai bisnis, definisi *metric*	Model dbt, *semantic layer*
Data analyst	Menjawab pertanyaan bisnis dengan data	Analisis, *dashboard*, rekomendasi
Data scientist	Membangun model statistik dan prediktif	Model, eksperimen
ML engineer	Membawa model ke produksi dan menjaganya	Layanan model, *feature store*
Data steward / DPO	*Policy*, *data quality*, *privacy*, *compliance*	Katalog, kebijakan, bukti audit

Ringkasan

Data engineer ada di *upstream*. Kesalahan di *upstream* mengalir ke semua peran *downstream*, itu sebabnya pekerjaan ini menuntut ketelitian yang tinggi.

Data engineering lifecycle dan *undercurrents*

Data engineering lifecycle



Sumber: Reis dan Housley, *Fundamentals of Data Engineering*, Bab 2.

Lima tahap, satu contoh

Tahap	Pada bank syariah digital	Dibahas pada
Generation	*Database* inti perbankan, aplikasi *mobile*, log ATM, API mitra	P2 (*source* dan *data modeling*)
Storage	*Object storage*, *lakehouse*, *database* cepat untuk saldo	P3, P4, P7
Ingestion	Ekspor harian (*batch*) dan CDC tiap transaksi (*streaming*)	P5, P11
Transformation	Membersihkan, menyatukan, menghitung agregat, mendeteksi pola	P5, P6, P9, P10
Serving	*Dashboard* manajemen, laporan regulator, *feature* model *fraud*	P12, P13
Undercurrents	*Encryption*, *access control*, *catalog*, *lineage*, *compliance* UU PDP	P14, P15

Ringkasan

Undercurrents dirancang sejak awal. *Security* dan *governance* yang ditempel belakangan *cost*-nya jauh lebih mahal daripada yang dirancang sejak awal.

Big data: kapan data disebut besar

Lima V, dan definisi yang lebih jujur

- ***Volume***: terabyte sampai petabyte
- ***Velocity***: ribuan sampai jutaan *event* per detik
- ***Variety***: tabel, JSON, log, citra, audio, teks
- ***Veracity***: data kotor, ganda, terlambat, saling bertentangan
- ***Value***: hanya bernilai bila sampai ke keputusan

Definisi praktis

Data disebut besar ketika **cara biasa berhenti bekerja**: tidak muat di memori satu mesin, terlalu lambat diproses satu mesin, atau datang lebih cepat daripada kemampuan satu mesin mencatatnya.

Hitungan kasar

Membaca 1 TB dari satu disk dengan laju $\approx$ 200 MB/s:

$$\frac{10^{12}}{2 \times 10^{8}} = 5.000 \text{ detik} \approx 83 \text{ menit}$$

Dibaca paralel dari 100 disk: $\approx$ 50 detik.

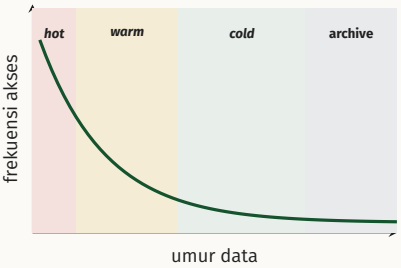
Satu juta nasabah $\times$ 5 transaksi per hari $\times$ 1 KB $\approx$ 5 GB per hari $\approx$ 1,8 TB per tahun, belum termasuk log aplikasi yang bisa sepuluh kali lebih besar.

Jangan terburu-buru ke *cluster*

Data puluhan GB sering lebih cepat diolah DuckDB di satu laptop daripada di *cluster*. Pilih alat sesuai ukuran.

Hot, *warm*, dan *cold*

Data tidak sama nilainya sepanjang umurnya



- Transaksi **detik ini** dibaca berkali-kali: sistem *fraud*, saldo, notifikasi.
- Transaksi **bulan ini** dibaca untuk *dashboard* dan rekonsiliasi.
- Transaksi **tahun lalu** jarang disentuh, kecuali untuk analitik tahunan dan melatih model.
- Transaksi **bertahun-tahun lalu** hampir tidak pernah dibaca, tetapi **wajib disimpan** untuk audit dan hukum.

Ringkasan

Menyimpan semuanya di *storage* tercepat itu boros; menyimpan semuanya di *storage* termurah membuat keputusan terlambat.

Empat *tier* dan *cost*-nya

Tier	Media khas	*Latency* baca	*Storage cost*	Contoh di bank
Hot	Memori, SSD/NVMe, Redis, ClickHouse	mikro–milidetik	paling mahal	saldo, skor *fraud*, 7 hari terakhir
Warm	SSD/HDD, *lakehouse* di *object storage*	detik	sedang	13 bulan terakhir untuk *dashboard*
Cold	*Object storage* kelas *infrequent access*	detik, ada *retrieval cost*	murah	data historis untuk melatih model
Archive	Glacier, pita magnetik	menit–jam	sangat murah	bukti transaksi 10 tahun

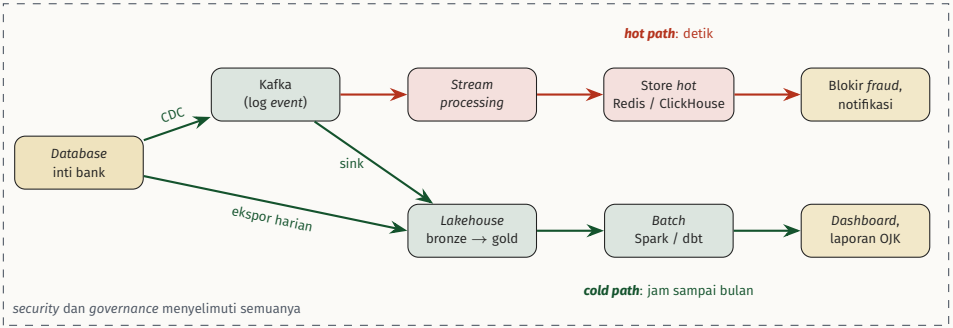
Hot path

Stream event yang diproses segera (*streaming*): Kafka, Flink, Spark *Structured Streaming*, *real-time* OLAP. Dibahas setelah UTS (P11–P12).

Cold path

Data dikumpulkan lalu diproses berkala (*batch*): *object storage*, *lakehouse*, Polars, DuckDB, dbt, Airflow (sebelum UTS); Spark di *cluster* setelah UTS (P9–P10).

Dua jalur dalam satu arsitektur



Pertanyaan yang akan sering kita ajukan: **seberapa segar** data ini harus sampai, **berapa *cost*-nya**, dan **apa risikonya** bila salah atau bocor?

Keamanan dan GRC: data adalah amanah

Mengapa *security* dan *governance* masuk mata kuliah ini

Data security

- *Encryption in transit* dan *at rest*
- *Least privilege*: akses seminimal mungkin
- *Masking* dan *tokenization* untuk PII
- *Audit trail*: siapa membaca apa, kapan

Data governance

- *Data ownership*: siapa pemiliknya, apa artinya
- Dari mana asalnya (*lineage*)
- *Data quality* dan *retention*

GRC: *Governance, Risk, Compliance*

Governance: aturan dan peran. ***Risk***: apa yang bisa salah, seberapa mungkin, seberapa parah. ***Compliance***: bukti bahwa aturan internal dan regulasi dipatuhi.

UU No. 27 Tahun 2022 (UU PDP)

Berlaku penuh sejak Oktober 2024. Antara lain: kegagalan pelindungan data pribadi wajib diberitahukan paling lambat **3 × 24 jam**; sanksi administratif dapat berupa denda hingga **2% pendapatan tahunan**.

Standar yang akan kita rujuk: ISO/IEC 27001 (keamanan informasi) dan ISO/IEC 27701 (privasi), serta DAMA-DMBOK (manajemen data).

Amanah sebagai *engineering principle*

Nasabah menitipkan data pribadinya kepada bank seperti menitipkan hartanya. Dalam praktik *engineering*, amanah itu berarti:

1. ***Data minimization***: data yang tidak dikumpulkan tidak bisa bocor.
2. ***Purpose limitation***: data untuk pembiayaan tidak dipakai untuk hal lain tanpa dasar yang sah.
3. ***Access control***: tidak semua engineer perlu melihat nomor KTP.
4. ***Retention* dan *right to erasure***: hapus bila tidak berhak lagi disimpan, termasuk di *cold tier* dan *backup*.
5. ***Accountability***: bisa menunjukkan bukti kepada auditor.

Pertanyaan diskusi

Sebuah tim ingin menyalin seluruh tabel nasabah produksi (nama, NIK, alamat, saldo) ke laptop untuk "mencoba model baru". Apa saja risikonya, dan bagaimana mencapai tujuan yang sama tanpa risiko itu?

Ringkasan

Keamanan dan kepatuhan yang baik diputuskan saat merancang arsitektur. Formulir yang diisi menjelang audit tidak menggantikan keputusan itu.

Studi kasus semester: bank syariah digital

Satu kasus, enam belas pertemuan

Sepanjang semester kita membangun platform data untuk sebuah bank syariah digital fiktif. Datanya sintetis; masalahnya nyata.

- **Nasabah**: identitas, KYC, segmen
- **Rekening dan transaksi**: tabungan wadiah, deposito mudharabah, transfer, QRIS
- **Pembiayaan**: murabahah, musyarakah, ijarah; angsuran dan kolektibilitas
- **Kanal**: aplikasi *mobile*, ATM, agen
- **Kepatuhan**: laporan regulator, anti pencucian uang, UU PDP

Kebutuhan yang akan kita penuhi

- Deteksi transaksi mencurigakan dalam hitungan detik (*hot*)
- *Dashboard* manajemen harian (*warm*)
- Laporan bulanan dan analitik tahunan (*cold*)
- Arsip bukti transaksi bertahun-tahun (*archive*)
- Data nasabah terlindungi dan dapat diaudit (*security*, GRC)

Kelompok boleh mengganti domain pada tugas besar (e-commerce, IoT, ESG) selama kebutuhan yang setara terpenuhi.

Bahasa dan perangkat: kapan memakai apa

Tiga bahasa, tiga peran

Python

Perekat. *Orchestration* (Airflow, Dagster), API, analitik, ML, PySpark, PyFlink, Polars.

\+ *library* terlengkap, mudah dibaca, dipakai analis dan *data scientist*.
— lambat bila logika berat ditulis di Python murni; GIL membatasi *thread*.

Scala

Bahasa asli ekosistem Apache di JVM. Spark dan Kafka ditulis dengan Scala; Flink dengan Java dan Scala.

\+ API bertipe (Dataset), tanpa *cost* lintas bahasa, cocok untuk *job* besar *enterprise*.
— kurva belajar, waktu kompilasi, *garbage collection* JVM.

Rust

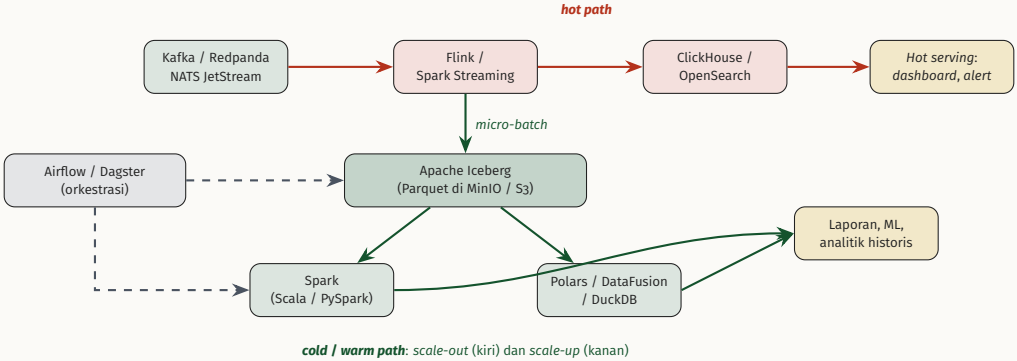
Mesin berkinerja tinggi. Polars, DataFusion, Vector, delta-rs, iceberg-rust.

\+ aman memori tanpa *garbage collector*, hemat CPU dan memori, *latency* stabil.
— kurva belajar paling curam; ekosistem data masih muda.

Praktik industri

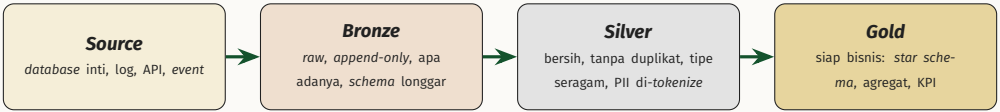
Kebanyakan engineer **menulis Python dan SQL**, sementara mesinnya berjalan di JVM atau Rust. Kita belajar Python dan SQL sebagai bahasa sehari-hari, Scala untuk memahami Spark dari dalam, dan Rust lewat Polars dan DataFusion untuk merasakan kinerja tanpa *cluster*.

Arsitektur acuan yang akan kita bangun



Satu *stack* yang utuh lebih berguna daripada mengenal sepuluh alat secara terpisah. Setiap pertemuan menambahkan satu bagian dari gambar ini.

Medallion: *bronze*, *silver*, *gold* sebagai benang merah semester



Bronze

Bukti asli: bisa diproses ulang kapan saja bila logika *downstream* salah. Akses paling dibatasi karena masih berisi PII.

Silver

Satu versi data yang bisa dipercaya: aturan *data quality* ditegakkan di sini.

Gold

Yang dibaca *dashboard*, *app*, dan model ML. Dibuat per kebutuhan bisnis.

UTS membangun ketiganya di satu mesin (P4–P6); UAS membangunnya ulang dengan Spark di *cluster* (P9–P10) sampai ke *dashboard* atau aplikasi (P13).

When to use: compute engine

Alat	Pakai bila	Pertimbangkan alternatif bila
Spark (Scala, PySpark)	*Batch* dan ETL ratusan GB–PB, SQL analitik di *cluster*, tim sudah memakai ekosistem JVM	Data muat di satu mesin; butuh *latency* sub-detik
Flink	*Stream stateful*, *event time*, *latency* sub-detik, *exactly-once*	Hanya butuh *batch*; tim kecil tanpa pengalaman operasi JVM
Beam	Satu kode untuk beberapa *runner*; memakai Google Cloud Dataflow	Hanya satu *engine*; butuh *feature* khas *engine* tertentu
DataFusion / Ballista	Membangun *query engine* atau produk sendiri di atas Arrow; SQL *columnar* tanpa JVM	Butuh *cluster* terdistribusi yang sudah matang (Ballista masih berkembang)
Polars	*Dataframe* puluhan GB di satu mesin, pengganti pandas yang jauh lebih hemat memori	Data melebihi kapasitas satu mesin besar
DuckDB	SQL analitik di satu proses, langsung atas Parquet atau Iceberg	Banyak penulis bersamaan; beban OLTP

Sumber: Ringkasan praktik umum; rinciannya dibahas dan diukur pada pertemuan 5 (satu mesin) serta 9–11 (Spark, *streaming*).

When to use: *messaging* dengan Kafka atau NATS

Alat	Pakai bila	Pertimbangkan alternatif bila
Kafka	Log *event* tahan lama yang dapat diputar ulang, *throughput* sangat tinggi, CDC (Debezium), Kafka Connect, *schema registry*	*Cluster* kecil dengan *resource* terbatas; hanya butuh *messaging* antar layanan
Redpanda	Ingin API Kafka tanpa JVM dan ZooKeeper; lab lokal; *latency* stabil	Butuh *feature* atau lisensi yang hanya ada di distribusi Kafka tertentu
NATS (*core*)	*Pub-sub* dan *request-reply* antar *microservice* yang sangat ringan; tidak perlu menyimpan *message*	*Message* tidak boleh hilang (*core* bersifat *at-most-once*)
NATS JetStream	Butuh persistensi dan *replay* tetapi tetap ringan: *edge*, IoT, *microservices*, satu biner kecil	Ekosistem konektor ke *lakehouse* dan CDC selengkap Kafka
Vector (Rust)	Mengumpulkan dan merutekan log, *metric*, *trace* dengan jejak memori kecil	Data bisnis yang butuh semantik transaksi dan *schema* ketat

Ringkasan

Kafka adalah ***log*** yang bisa dipakai sebagai *queue*; NATS adalah ***messaging system*** yang dengan JetStream bisa menyimpan. Pilih menurut *replay*, konektor, dan *resource* (pertemuan 11).

When to use: storage, serving, orchestration

Alat	Pakai bila	Pertimbangkan alternatif bila
Iceberg	Tabel terbuka yang dibaca banyak *engine* (Spark, Flink, Trino, DuckDB), *hidden partitioning*, *time travel*	Seluruh platform sudah Databricks (Delta lebih alami)
Delta Lake	Ekosistem Spark/Databricks; delta-rs untuk Python dan Rust	Butuh netralitas *engine* seluas mungkin
MinIO / **Ceph**	*Object storage* S3-*compatible* di lab atau *on-premise*; Ceph bila butuh *block*, *file*, dan *object* sekaligus di skala besar	Ada layanan *cloud* terkelola; cek lisensi terbaru
ClickHouse	*Aggregation real-time* atas miliaran baris, ingest *streaming* cepat	Butuh pencarian teks bebas atau transaksi OLTP
OpenSearch	Pencarian teks, analitik log, *vector search*	*Aggregation* numerik berat (ClickHouse lebih hemat)
PostgreSQL	OLTP, *metadata store*, *serving* data teragregasi; TimescaleDB, pgvector	Analitik terabyte (pakai *columnar engine*)
Airflow / **Dagster**	Airflow: standar industri, ekosistem operator terluas. Dagster: berorientasi *asset*, mudah diuji	Pemicu berbasis *event* murni (pakai *streaming*)

Best practice industri yang kita pegang

1. ***Start simple.*** DuckDB atau Polars dulu; *cluster* hanya bila ukuran data, *concurrency*, atau *availability* menuntutnya.
2. ***Open formats.*** Parquet dan Iceberg agar data tidak terkunci pada satu vendor atau *engine*.
3. **Pisahkan *compute* dan *storage*.** Data di *object storage*; mesin boleh berganti.
4. ***Idempotent pipeline.*** Menjalankan ulang tidak menggandakan data.
5. ***Single source of truth* per *metric*.** Definisi di satu tempat (*semantic layer*).
6. ***Hot* seperlunya.** Hanya data yang benar-benar butuh detik yang dibayar dengan harga *hot*.
7. ***Schema as contract.*** *Schema registry* dan *data contract* di batas antar-tim.
8. ***Security by design.*** *Encryption*, *least privilege*, PII ditokenisasi sedini mungkin.
9. ***Infrastructure as Code.*** Semua *docker-compose*, DAG, dan *configuration* ada di Git.
10. ***Measure, don't guess.*** Klaim cepat atau murah disertai angka.

Kurikulum acuan

Pola dari lima kampus Eropa

Kampus	Mata kuliah	Yang kita adopsi
ETH Zurich	Big Data (263-3010-00L)	*Layer storage* (HDFS vs *object storage*), MapReduce ke Spark, ragam *storage engine*
EPFL	COM-490; CS-423	*Data lake columnar*, Spark *batch*, transisi *hot path* (*streaming*) dan *cold path*
Cambridge	R244; *Distributed Systems*	CAP, PACELC, *partition*, *trade-off throughput–latency*
TUM	IN2258; *Cloud Databases*	*Messaging*, *publish-subscribe*, pemisahan *compute* dan *storage*
KU Leuven	DoSo6B	Sudut pandang *data-driven*: *pipeline* melayani keputusan dan *metric* bisnis

Ringkasan

Kita mengambil kerangka teori dari kampus-kampus itu, lalu menambah dua hal yang dibutuhkan industri di Indonesia: **keamanan data** dan **kepatuhan UU PDP**.

Kesinambungan dengan mata kuliah Sistem Terdistribusi

Sister berjalan setiap Sabtu, DE setiap Senin. Prinsipnya: **DE memakai dan memilih alat, Sister membedah mesinnya**.

Topik	Di Data Engineering	Di Sistem Terdistribusi
HDFS, *replication*, *erasure coding*	P3: sepenuhnya di sini	hanya disebut
Hadoop, MapReduce, Spark	P9–P10: sepenuhnya di sini, termasuk RDD dan *lineage*	hanya disebut; P11: *collective* MPI sebagai pembanding *shuffle*
Format *columnar*, Arrow, SIMD	P2, P5: mengukur bahwa ia cepat	P10: mengapa cepat (*cache*, SIMD)
Kafka, *pub-sub*	P11: CDC, *streaming*	P5: model *messaging*
Replication, CAP, *consensus*	P7: lensa memilih *database*	P7: mekanismenya (Raft, BFT)
TLS dan mTLS	P14: mengamankan *pipeline*	P3: cara kerjanya

Bila sebuah topik muncul di keduanya, yang satu merujuk yang lain, tidak diulang dari nol. Hitungan “1 TB dari satu disk” tadi sama dengan yang Anda lihat di Sister P1 Sabtu lalu.

Rencana satu semester

Peta mata kuliah

Capaian A — *pipeline* ETL di satu mesin

1 Data Engineering 101
2 *Data modeling* dan format *columnar*
3 *Storage*, HDFS, dan *tiering*
4 *Lakehouse* dan *medallion*: *bronze*, *silver*, *gold*
5 ETL satu mesin: Arrow, Polars, DuckDB, DataFusion
6 *Orchestration*, dbt, *data quality*
7 *Storage engine*, NoSQL, *serving*, *ledger*
8 UTS: *pipeline* ETL *bronze–silver–gold*

Capaian B — skala besar dengan Spark sampai *dashboard/app*

9 Hadoop ke Spark (Scala, PySpark)
10 Spark skala besar (HPC data)
11 Kafka/NATS, CDC, *Structured Streaming*
12 Arsitektur *hot–cold* dan *serving*
13 *Dashboard* dan *data app*
14 *Data security*
15 *Governance*, GRC, *observability*
16 UAS: *pipeline* Spark sampai *dashboard/app*

Spark mendapat dua pertemuan karena menjadi inti UAS; topik lain satu pertemuan.

Dua tugas besar

UTS — “*Pipeline* ETL *bronze–silver–gold*”

- Satu mesin: Polars, DuckDB, atau dbt (Spark tidak wajib)
- *Ingest dataset* nyata ke *object storage* (*bronze*)
- *Silver*: bersih, tanpa duplikat, PII di-*tokenize*
- *Gold*: *star schema* siap bisnis
- Ter-*orchestrate* Airflow/Dagster, *data quality test*
- Kebijakan *tiering* dan *cost*-nya; satu *dashboard* sederhana

UAS — “Spark skala besar sampai *dashboard/app*”

- Data jauh lebih besar; Spark multi-*worker* (Scala/PySpark)
- *Bronze–silver–gold* di Iceberg/Delta
- Bukti *strong/weak scaling* vs *baseline* UTS
- *Serving store* lalu *dashboard* (Superset/Metabase) atau *data app* (Streamlit/FastAPI) yang didemokan
- *Security* dan *governance*; *hot path* sebagai nilai tambah

Kelompok 3–4 orang. Keduanya dinilai dari kode yang berjalan, laporan, dan presentasi.

Penilaian dan aturan main

Komponen nilai

Tugas, praktikum, dan kuis	20%
UTS, tugas besar	35%
UAS, tugas besar	45%

UTS dan UAS berupa **tugas besar**: kode, *pipeline* yang berjalan, laporan, dan presentasi.

Aturan main

- Praktikum memakai Python, SQL, dan Docker Compose; Docker wajib terpasang
- Setiap rancangan disertai alasan ***latency*, *cost*, dan risiko**, dengan angka
- Data pribadi pada praktikum **selalu sintetis**; dilarang memakai data nasabah atau pengguna sungguhan
- Boleh memakai alat bantu AI, asalkan **dituliskan** bagian mana yang dibantu dan Anda mampu menjelaskan kodenya
- Menyalin pekerjaan orang lain tanpa menyebut sumber dinilai nol

Perangkat praktikum

Kebutuhan	Perangkat (semuanya terbuka, berjalan di laptop)
Dasar	Linux atau WSL2, Docker Compose, Git, Python 3, JDK 17 + sbt (Scala), `cargo` (opsional, Rust); RAM disarankan $\geq$ 8 GB
Query satu mesin	DuckDB, Polars, DataFusion (`datafusion-python`), Apache Arrow
Storage dan *lakehouse*	MinIO, Apache Iceberg (PyIceberg) atau Delta Lake (delta-rs), Trino
Batch terdistribusi	Apache Spark (Scala dan PySpark), dbt
Orchestration dan *data quality*	Apache Airflow atau Dagster, uji dbt, Great Expectations atau Soda
Streaming	Kafka atau Redpanda, NATS JetStream, Debezium, Vector, Apache Flink atau Spark *Structured Streaming*, Apache Beam (pengenalan)
Serving dan penyajian	ClickHouse, OpenSearch, PostgreSQL (TimescaleDB, pgvector), Redis; Superset atau Metabase, Streamlit, FastAPI
Security dan *governance*	Microsoft Presidio, HashiCorp Vault, OpenMetadata

Kerangka `docker-compose` tiap praktikum dibagikan di kelas; tidak perlu akun *cloud* berbayar.

Referensi

Utama

- Reis dan Housley, *Fundamentals of Data Engineering*, O'Reilly, 2022
- Kleppmann, *Designing Data-Intensive Applications*, O'Reilly, 2017
- Kimball dan Ross, *The Data Warehouse Toolkit*, ed. 3, 2013
- Chambers dan Zaharia, *Spark: The Definitive Guide*, 2018
- Shapira dkk., *Kafka: The Definitive Guide*, ed. 2, 2021
- Akidau dkk., *Streaming Systems*, O'Reilly, 2018

Security, governance, data-driven

- DAMA International, DAMA-DMBOK, ed. 2
- UU No. 27 Tahun 2022 tentang Pelindungan Data Pribadi
- ISO/IEC 27001:2022 dan ISO/IEC 27701
- Baesens, *Analytics in a Big Data World*, Wiley, 2014
- Dehghani, *Data Mesh*, O'Reilly, 2022
- Jarmul, *Practical Data Privacy*, O'Reilly, 2023

Sumber: Daftar lengkap beserta makalah pendukung ada di RPS.

Tugas untuk pertemuan berikutnya

1. Baca Reis dan Housley Bab 1 dan 2.
2. Pilih satu aplikasi yang Anda pakai sehari-hari (dompet digital, e-commerce, ojek daring). Gambar dugaan alur datanya dari sumber sampai *dashboard*, lalu tandai:
   - tahap *lifecycle* (*generation* sampai *serving*),
   - data mana yang *hot*, *warm*, *cold*, dan *archive*,
   - satu data pribadi yang menurut Anda paling berisiko bila bocor.
3. Pasang Docker dan DuckDB di laptop; pastikan `docker run hello-world` berjalan.

Pertemuan depan

Sumber data dan cara memodelkannya: OLTP vs OLAP, *star schema*, *slowly changing dimension*, dan mengapa Parquet bisa puluhan kali lebih cepat daripada CSV untuk analitik.

Terima kasih

Pertanyaan?