Pertemuan 12 — Arsitektur *Hot* dan *Cold* di Industri

Lambda, Kappa, *real-time* OLAP, dan cara menimbang *latency*, *consistency*, serta *cost*

Hendri Karisma

Semester Ganjil 2026/2027

Program Studi Teknik Informatika, STMIK Tazkia

Alur pertemuan hari ini

1. *Hot*, *warm*, *cold* sebagai keputusan bisnis
2. SLA, SLO, SLI untuk data
3. Lambda *architecture* dan masalah dua basis kode
4. Kappa *architecture*: semuanya *stream*
5. *Streaming lakehouse*
6. *Serving layer*: ClickHouse, Druid, Pinot, OpenSearch, Redis
7. *Feature store* dan *reverse ETL*
8. Pola dari industri: Uber, Netflix, LinkedIn, Gojek
9. Arsitektur acuan bank syariah digital
10. Latihan rancangan dan arahan tugas besar UAS

Sasaran pertemuan ini

Anda dapat memetakan setiap kebutuhan bisnis ke *tier* yang tepat, memilih antara Lambda dan Kappa dengan alasan, dan menuliskan SLO serta estimasi *cost* untuk sebuah arsitektur *hot–cold*.

Hot, *warm*, *cold* sebagai keputusan bisnis

Empat pertanyaan sebelum memilih *tier*

1. **Kesegaran** (*freshness*): seberapa baru data harus sampai ke pemakai? Detik, menit, jam, hari?
2. ***Latency query***: seberapa cepat jawaban harus kembali? Milidetik untuk aplikasi, detik untuk *dashboard*, menit boleh untuk analis.
3. **Frekuensi akses**: dibaca ribuan kali per detik, atau sebulan sekali?
4. ***Cost* salah atau terlambat**: apa yang hilang bila data datang terlambat satu jam?

Dua sumbu yang sering tertukar

Kesegaran adalah jarak antara kejadian dan data tersedia.
Latency query adalah jarak antara bertanya dan menerima jawaban.

Dashboard bisa menjawab dalam 200 ms (*latency* rendah) tetapi menampilkan data kemarin (kesegaran rendah). Keduanya harus ditulis terpisah.

Jebakan umum

“Semua harus *real-time*” hampir selalu salah. *Hot path* mahal dibangun, mahal dijalankan, dan sulit dibuat benar.

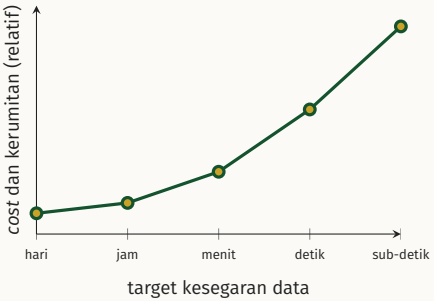
Memetakan kebutuhan bank syariah digital

Kebutuhan	Kesegaran	*Latency query*	Akses	*Tier*
Skor *fraud* per transaksi	< 1 detik	< 100 ms	sangat sering	***hot***
Saldo dan mutasi terakhir di aplikasi	detik	milidetik	sangat sering	***hot***
Dashboard transaksi per cabang hari ini	menit	detik	sering	***hot*** / ***warm***
Dashboard manajemen 13 bulan	harian	detik	harian	***warm***
Laporan bulanan regulator (OJK)	bulanan	menit	bulanan	***cold***
Training data model *fraud* 3 tahun	harian	menit	mingguan	***cold***
Bukti transaksi untuk audit	—	jam boleh	jarang	***archive***

Ringkasan

Satu *event* transaksi yang sama akan hidup di beberapa *tier* sekaligus. Arsitektur yang baik menulis sekali di log, lalu menurunkannya ke tiap *tier* sesuai kebutuhan.

Cost naik tajam ketika kesegaran dikejar



Kurva ilustrasi, bukan data pengukuran.

Yang bertambah ketika target makin segar:

- Komponen yang menyala terus: *broker*, *stream processor*, *store* cepat
- *State* yang harus dijaga dan di-*checkpoint*
- Penanganan *late data*, duplikat, dan urutan
- Jaga *on-call* 24 jam karena kegagalan langsung terasa
- Pengujian yang lebih sulit

Aturan praktis

Mulai dari *batch*. Naikkan satu kebutuhan ke *hot path* hanya bila ada nilai bisnis yang bisa disebutkan dan diukur.

SLA, SLO, SLI untuk data

Menulis janji yang bisa diukur

- **SLI** (*indicator*): angka yang diukur. Contoh: selisih waktu antara *event* terakhir di Kafka dan baris terakhir di ClickHouse.
- **SLO** (*objective*): target internal atas SLI. Contoh: 99% waktu, selisih itu di bawah 10 detik.
- **SLA** (*agreement*): janji kepada pihak lain beserta konsekuensinya. Biasanya lebih longgar daripada SLO.

Error budget

SLO 99% dalam 30 hari berarti boleh melanggar sekitar 7,2 jam ($0{,}01 \times 720$ jam). Bila *budget* habis, tim berhenti menambah *feature* dan memperbaiki *reliability*.

SLI data	Contoh ukuran
Freshness	umur baris terbaru di tabel
Latency	waktu *query* P95 *dashboard*
Completeness	jumlah baris vs sumber
Correctness	persentase *data quality test* lulus
Availability	persentase *query* berhasil

Ringkasan

Tanpa SLO tertulis, perdebatan "*hot* atau *cold*" hanya adu pendapat.

Contoh SLO untuk tiga jalur bank

Jalur	SLO (ilustrasi)	Cara mengukur SLI
Skor *fraud*	99,9% transaksi mendapat skor dalam 500 ms sejak masuk Kafka	*timestamp event* vs *timestamp* keputusan, P99 per menit
Dashboard cabang	data tidak lebih tua dari 2 menit selama jam operasional, 99% waktu	`now() - max(event_time)` di ClickHouse
Laporan regulator	tabel *gold* siap paling lambat H+3 tiap bulan, lengkap 100%	jumlah transaksi *gold* vs rekonsiliasi *core banking*

Perhatikan

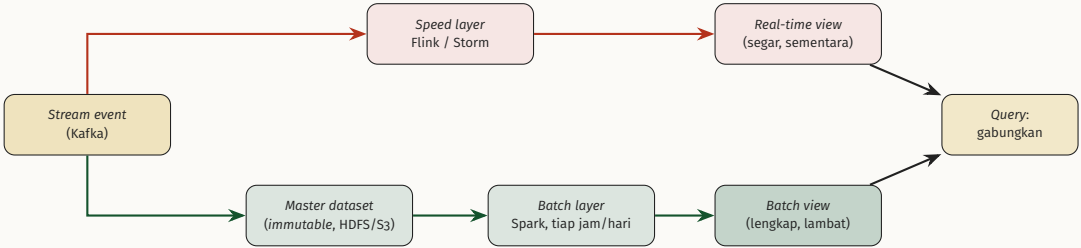
SLO laporan regulator longgar soal waktu, tetapi **ketat soal kelengkapan**. *Cold* bukan berarti boleh asal.

Siapa memegang SLO

Pemilik data produk (pertemuan 13) menyetujui SLO; tim platform memantau SLI dan memasang *alert* (pertemuan 15).

Lambda *architecture*

Lambda: dua jalur, satu *serving layer*



- Dirumuskan Nathan Marz (buku *Big Data*, Marz dan Warren, 2015).
- *Batch layer* menghitung ulang dari seluruh data mentah: benar dan lengkap.
- *Speed layer* hanya menutup celah sejak *batch* terakhir.
- Kesalahan di *speed layer* "terhapus sendiri" saat *batch* berikutnya selesai.
- Masuk akal ketika mesin *streaming* belum bisa menjamin *exactly-once* dan *state* besar.

Masalah Lambda: logika yang sama ditulis dua kali

Gejalanya di bank

- Aturan “transaksi mencurigakan” ditulis di Spark SQL (*batch*) dan di Flink (*speed*)
- Analis *fraud* mengubah ambang di satu tempat, lupa di tempat lain
- *Dashboard* siang hari dan laporan esok pagi menampilkan angka berbeda
- Dua sistem untuk dioperasikan, dipantau, dan diuji

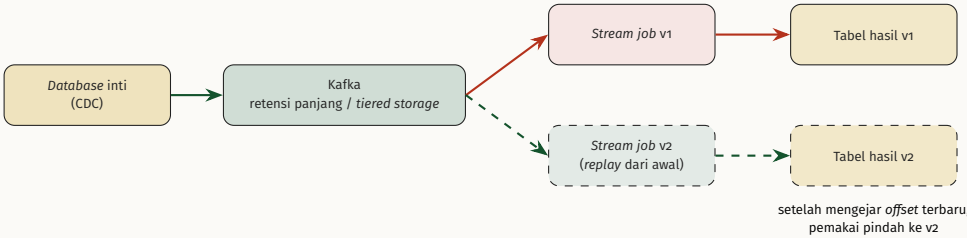
Jalan keluar yang dicoba industri

- Satu *API* untuk dua mesin: Apache Beam, Spark (*batch* dan *Structured Streaming* memakai DataFrame yang sama)
- Kappa: buang *batch layer* sama sekali
- *Streaming lakehouse*: *stream* langsung ke tabel Iceberg/Delta

Sumber: Jay Kreps, *Questioning the Lambda Architecture*, 2014.

Kappa *architecture*

Kappa: log adalah sumber kebenaran



- Satu basis kode: *stream processing* saja.
- "Menghitung ulang" = menjalankan *job* versi baru dari *offset* awal, lalu memindahkan pemakai.
- Butuh log yang menyimpan cukup lama dan mesin *stream* yang benar (*exactly-once*, *event time*, pertemuan 11).

Batasnya

Replay bertahun-tahun data bisa memakan waktu lama dan mahal. Analitik historis besar (*join* besar, *ad-hoc*) tetap lebih cocok di *lakehouse*. Kappa murni jarang dijumpai di bank.

Lambda atau Kappa?

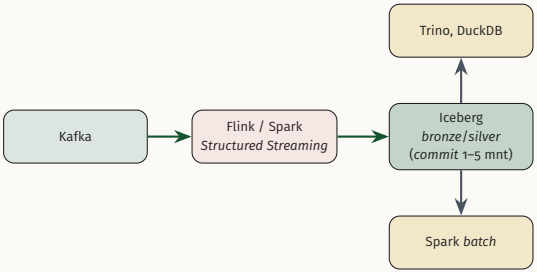
	Lambda	**Kappa**
Basis kode	dua (*batch* + *stream*)	satu (*stream*)
Menghitung ulang	jalankan ulang *batch*	*replay* log
Kebenaran jangka panjang	dijamin *batch layer*	bergantung mesin *stream*
Analitik historis besar	alami	canggung bila hanya log
Beban operasional	tinggi	sedang, tetapi butuh keahlian *streaming*
Cocok bila	ada laporan resmi yang harus tepat dan *batch* sudah matang	hampir semua kebutuhan bersifat *stream*, sejarah pendek

Ringkasan

Di industri, bentuk yang paling sering dijumpai hari ini adalah **campuran**: satu log (Kafka) sebagai jalur utama, *hot path* berbasis *stream*, dan *lakehouse* untuk *cold path*, dengan logika bisnis didefinisikan sekali di tempat yang disepakati.

Streaming lakehouse

Stream yang langsung menjadi tabel



- *Stream job* menulis *micro-batch* ke tabel Iceberg atau Delta; tiap *commit* menjadi *snapshot* baru (pertemuan 4).
- Satu tabel dibaca oleh *query batch* **dan** *query* yang hanya mengambil *snapshot* terbaru: kesegaran menit tanpa *batch layer* terpisah.
- Jebakan: *commit* terlalu sering → ribuan *file* kecil. Butuh *compaction* terjadwal (Airflow).

Ringkasan

Streaming lakehouse memberi *warm* yang murah: kesegaran menit dengan *cost object storage*. Untuk *query* milidetik, kita tetap butuh *serving layer* khusus.

Serving layer

Pilihan *store* untuk *hot serving*

Store	Kekuatan	Pakai di bank untuk
Redis	*lookup* per kunci dalam sub-milidetik, struktur data di memori	profil risiko per rekening, hitungan transaksi per menit, *cache* saldo
ClickHouse	OLAP kolumnar, *ingest* cepat dari Kafka, *aggregation* miliaran baris	*dashboard* cabang *real-time*, *monitoring fraud*
Druid / **Pinot**	OLAP *real-time* untuk banyak pengguna serentak, segmen ber-indeks	analitik yang dibuka ke banyak pengguna atau nasabah
OpenSearch	*inverted index*: teks bebas, log, *vector search*	mencari keterangan transaksi, investigasi log akses
PostgreSQL	OLTP, *serving* data teragregasi kecil–menengah	API laporan ringkas per nasabah

Ringkasan

Pertanyaan kuncinya: **bentuk *query***. *Lookup* per kunci → Redis. *Aggregation* atas banyak baris → ClickHouse/Druid/Pinot. Teks → OpenSearch.

Mengapa ClickHouse cocok untuk *hot/warm* analitik

- **Kolumnar dan *vectorized***: membaca hanya kolom yang dibutuhkan (pertemuan 2, Sister P10).
- **MergeTree**: data ditulis sebagai bagian terurut, digabung di latar (mirip LSM, pertemuan 7).
- **Kafka *engine*** dan *materialized view*, membaca topik langsung dan menghitung agregat saat data masuk.
- **TTL** per tabel atau kolom, memindahkan atau menghapus data lama, termasuk ke *disk* atau *object storage* yang lebih murah.

TTL: *tiering* di dalam satu *database*

Data 7 hari terakhir di SSD lokal, 7–90 hari dipindah ke volume yang lebih murah, lebih dari 90 hari dihapus karena salinan lengkapnya ada di Iceberg.

Bukan untuk

Transaksi OLTP, *update* per baris yang sering, dan sumber kebenaran tunggal. ClickHouse adalah **turunan** dari log, bukan pengganti *core banking*.

Contoh: dari topik Kafka ke agregat per menit

```
-- 1. tabel penampung yang membaca Kafka
CREATE TABLE trx_kafka (
  trx_id String, rekening String,
  nominal Decimal(18,2), event_time DateTime64(3)
) ENGINE = Kafka
SETTINGS kafka_broker_list = 'redpanda:9092',
         kafka_topic_list  = 'bank.transaksi',
         kafka_group_name  = 'ch-hot',
         kafka_format      = 'JSONEachRow';

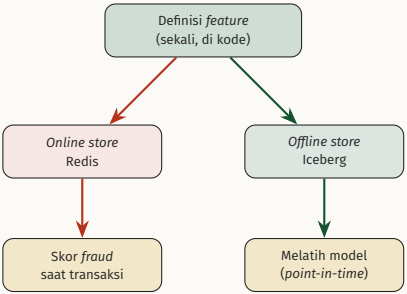
-- 2. tabel agregat per cabang per menit
CREATE TABLE trx_per_menit (
  cabang String, menit DateTime,
  jumlah UInt64, total Decimal(18,2)
) ENGINE = SummingMergeTree
ORDER BY (cabang, menit)
TTL menit + INTERVAL 90 DAY;
```

```
-- 3. materialized view mengisi agregat
CREATE MATERIALIZED VIEW mv_trx
TO trx_per_menit AS
SELECT dictGet('dim_rek', 'cabang',
               rekening)   AS cabang,
       toStartOfMinute(event_time)
                           AS menit,
       count()             AS jumlah,
       sum(nominal)        AS total
FROM trx_kafka
GROUP BY cabang, menit;
```

Data mentah tetap mengalir ke Iceberg lewat jalur lain; ClickHouse hanya menyimpan agregat 90 hari.

Feature store dan *reverse ETL*

Feature store: *hot* dan *cold* untuk model



- Contoh *feature*: jumlah transaksi rekening dalam 10 menit terakhir, rata-rata nominal 30 hari, jumlah perangkat berbeda hari ini.
- **Online**: nilai terbaru per kunci, dibaca dalam milidetik.
- **Offline**: riwayat nilai *feature*, untuk melatih model dengan *point-in-time join* agar tidak bocor informasi masa depan.
- Tujuan utamanya mencegah *training-serving skew* (pertemuan 13).
- Contoh alat: Feast (terbuka), layanan *feature store* di platform *cloud*.

Reverse ETL: mengembalikan hasil ke sistem operasional

Arah biasa: sistem operasional → *lakehouse*. *Reverse ETL* membalik arahnya: hasil olahan di *layer gold* dikirim kembali ke alat yang dipakai orang bekerja.

- Segmen nasabah berpotensi pembiayaan → CRM tim pemasaran
- Skor risiko kolektibilitas → aplikasi petugas penagihan
- Daftar rekening yang perlu verifikasi ulang KYC → sistem tiket

Yang perlu dijaga

- *Idempotent*: kirim ulang tidak menggandakan tiket
- Hanya kolom yang perlu, jangan menyalin PII ke banyak alat (pertemuan 14)
- *Lineage*: CRM harus tahu dari model dan tabel mana skor itu datang (pertemuan 15)
- SLO kesegaran yang jelas: harian biasanya cukup

Pola dari industri

Apa yang bisa dipelajari dari arsitektur yang dipublikasikan

LinkedIn

Tempat lahirnya Kafka: log terdistribusi sebagai jalur utama yang menyatukan *data stream* antar sistem. Pola: **log sebagai sumber kebenaran**.

Netflix

Apache Iceberg berawal dari kebutuhan tabel analitik besar di sana. Pola: **format tabel terbuka di atas *object storage***.

Uber

Banyak menulis tentang *data lake* skala besar (Apache Hudi lahir di sana untuk *upsert* di atas *data lake*) dan analitik *real-time*. Pola: **kesegaran menit di atas *lake***.

Gojek

Menulis tentang platform data internal untuk *stream event* dan *feature store* (Feast awalnya dikembangkan bersama Gojek dan Google Cloud). Pola: ***feature* yang sama untuk *training* dan *serving***.

Sumber: Blog teknik masing-masing perusahaan. Baca aslinya; ringkasan ini sengaja hanya memuat pola umum, bukan angka skala.

Pola yang berulang

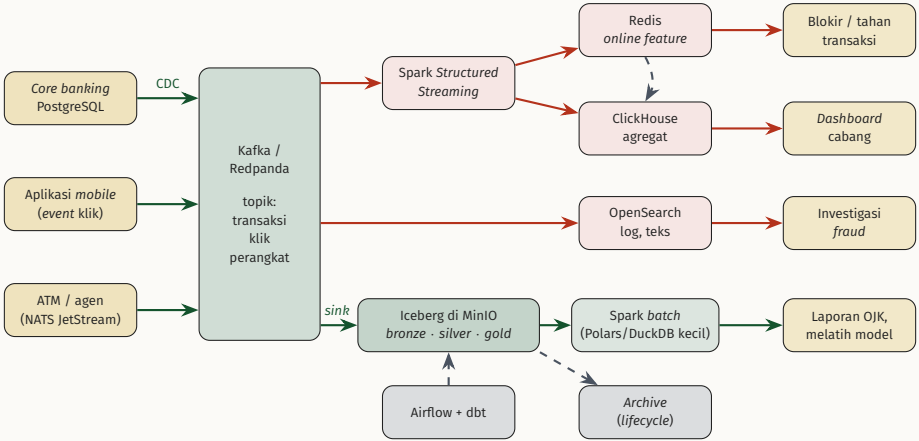
1. **Satu log, banyak turunan.** *Event* ditulis sekali; setiap *store* adalah *materialized view* dari log itu.
2. **Format terbuka di *object storage*** sebagai *tier* dingin dan hangat, dibaca banyak mesin.
3. ***Store* khusus untuk *hot serving***, dipilih menurut bentuk *query*.
4. **Definisi sekali** untuk *metric* dan *feature*, agar *hot* dan *cold* tidak berselisih.
5. **Platform swalayan**: tim domain membangun *pipeline* sendiri di atas platform bersama (pertemuan 13, *data mesh*).

Jangan menyalin skala

Arsitektur perusahaan dengan ratusan *engineer* data bukan cetak biru untuk tim lima orang. Ambil polanya, bukan jumlah komponennya.

Arsitektur acuan bank syariah digital

Arsitektur acuan: gambaran utuh



Merah: *hot path* (detik). **Hijau**: *cold/warm path* (menit–bulan). Abu-abu: orkestrasi dan *archive*. *Layer* keamanan dan *governance* (pertemuan 14–15) menyelimuti seluruhnya.

Menelusuri satu transaksi

1. Nasabah transfer Rp4,7 juta lewat aplikasi; baris baru tercatat di *core banking*.
2. Debezium membaca WAL dan menulis *event* ke topik `bank.transaksi` (pertemuan 11).
3. Spark *Structured Streaming* membaca *event*, memperbarui *state* "jumlah transaksi rekening ini dalam 1 menit", dan menulis *feature* ke Redis (pertemuan 11).
4. Layanan *fraud* membaca *feature* dari Redis dan memutuskan: lolos, tahan, atau tolak.
5. Agregat per cabang per menit bertambah di ClickHouse; *dashboard* kepala cabang ikut berubah.
6. Tiap beberapa menit, *event* yang sama di-*commit* ke Iceberg *bronze*.
7. Malam hari, Airflow menjalankan Spark *batch*: *bronze* → *silver* (bersih, PII ditokenisasi) → *gold* (fakta transaksi harian).
8. Akhir bulan, *layer gold* menjadi bahan laporan regulator; tahun berikutnya, *partition* lama berpindah ke *tier* dingin lalu *archive*.

Ringkasan

Satu *event*, delapan langkah, empat *tier*. Setiap langkah punya SLO dan pemilik.

Consistency antara jalur *hot* dan *cold*

Angka di *dashboard* siang hari dan laporan esok pagi bisa berbeda. Penyebabnya:

- *Late data*: *event* terlambat masuk setelah *window* ditutup
- Koreksi transaksi (pembatalan, *reversal*) yang hanya diterapkan di *batch*
- Definisi berbeda: “transaksi” termasuk biaya admin atau tidak?
- Zona waktu: WIB vs UTC pada batas hari

Cara industri menanganinya

- Tandai jelas: angka *hot* adalah **sementara**, angka *gold* adalah **final**
- Definisi *metric* di satu tempat (*semantic layer*, pertemuan 13)
- Rekonsiliasi otomatis harian: selisih *hot* vs *cold* di atas ambang $\rightarrow$ *alert*
- Simpan *event time* dalam UTC, tampilkan dalam WIB

Sumber: Kaitan teori: *trade-off consistency* dan *latency* (PACELC, pertemuan 7; mekanismenya di Sister P7).

Estimasi *cost*: cara berpikir dan komponennya

Asumsi (ilustrasi)

1 juta nasabah $\times$ 5 transaksi/hari $\times$ 1 KB $\approx$ 5 GB/hari
Ditambah *event* klik dan log: misalnya 10$\times$ $\approx$ 50 GB/hari
Setahun $\approx$ 18 TB mentah; Parquet terkompresi mungkin 3–10$\times$ lebih kecil

Komponen *cost*

- *Hot*: mesin yang menyala 24 jam (*broker*, Spark *Streaming*, ClickHouse, Redis), biasanya **dominan**
- *Warm/cold*: GB-bulan di *object storage* + *compute batch* yang hanya menyala saat dijalankan
- *Egress*: memindahkan data keluar *region* atau ke internet

Cara menghitung

biaya $= \sum_{tier}(\text{GB} \times \text{harga/GB-bulan}) +$
$\sum(\text{jam mesin} \times \text{harga/jam}) + \textit{egress}$

Ambil harga terkini dari kalkulator penyedia *cloud* atau dari harga server *on-premise*. Harga berubah; yang penting adalah **orde besaran** dan komponen mana yang dominan. $\sum$ (dibaca “sigma”): jumlahkan semua suku.

Ringkasan

Pertanyaan terbaik saat *review* arsitektur: “Kalau jalur *hot* ini kita matikan, kerugian bisnisnya berapa?” Bila tidak bisa dijawab, jalur itu belum layak *hot*.

When to use dan *best practice*

When to use: komponen arsitektur *hot–cold*

Komponen	Pakai bila	Hindari atau tunda bila
Kafka / Redpanda	banyak konsumen, *replay*, CDC, retensi panjang	hanya dua layanan bertukar perintah (NATS cukup)
NATS JetStream	perangkat *edge*, ATM, IoT, jejak kecil	butuh ekosistem konektor *lakehouse* yang luas
Flink	*state* besar, *event time*, *latency* sub-detik	kesegaran menit sudah cukup (*micro-batch* lebih sederhana)
Spark *Streaming*	tim sudah memakai Spark, kesegaran detik–menit	butuh *latency* sub-detik konsisten
ClickHouse	*dashboard* agregat *real-time*	sumber kebenaran, OLTP
Iceberg di MinIO	*warm/cold* murah, banyak mesin	*query* milidetik
Lambda	laporan resmi wajib tepat, *batch* sudah matang	tim kecil (dua basis kode)
Kappa	hampir semuanya *stream*, sejarah pendek	analitik historis besar dan *ad-hoc*

Best practice industri

1. **SLO dulu, teknologi kemudian.** Tulis kesegaran dan *latency* per kebutuhan sebelum memilih alat.
2. **Log sebagai jalur utama.** Semua *tier* diturunkan dari log yang sama; tidak ada salinan manual.
3. ***Layer gold* adalah sumber angka final.** *Hot* boleh sementara, asal diberi label.
4. **Definisi sekali.** *Metric* dan *feature* didefinisikan di satu tempat.
5. **TTL di setiap *store hot*.** Data tidak menumpuk di *tier* mahal.
6. **Rekonsiliasi otomatis** antara jalur *hot* dan *cold*.
7. **Rancang untuk *replay*.** *Job idempotent*, retensi log cukup untuk memperbaiki kesalahan.
8. **Ukur *cost* per jalur.** Tagihan dipisah per komponen agar keputusan *tiering* berbasis angka.

Latihan dan arahan tugas besar UAS

Latihan kelompok: rancang platform data bank syariah

Kebutuhan:

1. Deteksi *fraud* sebelum transaksi selesai
2. Saldo *real-time* di aplikasi
3. *Dashboard* manajemen harian dan 13 bulan
4. Laporan bulanan regulator
5. Arsip bukti transaksi 10 tahun
6. *Training data* model risiko pembiayaan

Yang harus dihasilkan (45 menit)

- Tabel kebutuhan → kesegaran, *latency*, *tier*, SLO
- Diagram *data flow*: sumber, log, *hot*, *cold*, *archive*
- Pilihan *store* per *tier* beserta alasannya
- Lambda, Kappa, atau campuran, dan mengapa
- Estimasi *cost* kasar: komponen dominan
- Dua risiko terbesar dan mitigasinya

Presentasi singkat 5 menit per kelompok; kelompok lain bertugas mencari satu kebutuhan yang tidak terpenuhi.

Praktikum: kerangka *docker-compose* jalur *hot* dan *cold*

```
services:
  redpanda:
    image: redpandadata/redpanda:latest
    command: redpanda start --smp 1 --memory 1G
             --overprovisioned
    ports: ["9092:9092"]
  clickhouse:
    image: clickhouse/clickhouse-server:latest
    ports: ["8123:8123"]
  minio:
    image: minio/minio:latest
    command: server /data --console-address :9001
    ports: ["9000:9000", "9001:9001"]
  redis:
    image: redis:7
  spark:
    image: apache/spark:latest   # job streaming
```

1. Jalankan generator transaksi sintetis ke topik `bank.transaksi`.
2. Pasang tabel Kafka dan *materialized view* di ClickHouse (contoh tadi).
3. Jalankan *job* Spark *Structured Streaming* yang menulis ke Iceberg di MinIO tiap 1 menit.
4. Ukur SLI: `now() - max(event_time)` di ClickHouse dan di Iceberg.
5. Bandingkan jumlah transaksi per cabang di kedua jalur setelah 30 menit; jelaskan selisihnya.

Versi *image* dikunci di *file* yang dibagikan di kelas; `latest` di sini hanya untuk ringkas.

Arahan UAS: *pipeline* Spark sampai *dashboard* atau aplikasi

Rancangan (dikumpulkan pekan ini)

- Domain dan *dataset* besar (terbuka atau sintetis)
- Diagram *pipeline* Spark *bronze* → *silver* → *gold* di Iceberg/Delta
- Rencana ukur *scaling* vs *baseline* UTS
- *Serving store* dan bentuk penyajian: *dashboard* atau *data app*
- Tabel kebutuhan dengan SLO; estimasi *cost* bila di *cloud*
- Opsional (nilai tambah): *hot path* Kafka + *Structured Streaming*

Yang akan ditambahkan pekan depan

- P13: KPI, *semantic layer*, *dashboard* dan *data app*
- P14: *encryption*, *role*, *tokenization* PII, *audit trail*
- P15: *catalog*, *lineage*, *data contract*, *risk register*, *observability*, presentasi progres
- P16: demo dan presentasi UAS

Ringkasan

Setiap kotak di diagram menjawab satu kebutuhan di tabel, dan berakhir di layar yang dipakai untuk mengambil keputusan.

Tugas untuk pertemuan berikutnya

1. Selesaikan dokumen rancangan tugas besar UAS (format di atas), satu dokumen per kelompok.
2. Kumpulkan laporan praktikum: SLI kesegaran jalur ClickHouse dan Iceberg, serta penjelasan selisih angka antar jalur.
3. Baca Kreps, *Questioning the Lambda Architecture* (2014), dan Reis dan Housley Bab 3 (*data architecture*).
4. Pilih satu artikel blog teknik (Uber, Netflix, LinkedIn, atau Gojek) tentang arsitektur data; tuliskan dalam satu halaman: masalahnya, polanya, dan apa yang **tidak** cocok diterapkan di tim kecil.

Pertemuan depan

Dari *pipeline* ke keputusan: KPI, *semantic layer*, lalu membangun *dashboard* (Superset/Metabase) dan *data app* (Streamlit) di atas *layer gold*, bahan utama demo UAS.

Terima kasih

Pertanyaan?