Pertemuan 3 — *Storage* Skala Besar dan *Tiering*

HDFS, *object storage*, *replication* vs *erasure coding*, dan *hot–warm–cold–archive*

Hendri Karisma

Semester Ganjil 2026/2027

Program Studi Teknik Informatika, STMIK Tazkia

Alur pertemuan hari ini

1. *Layer* fisik: disk, rak, data center
2. *Block*, *file*, dan *object storage*
3. HDFS: arsitektur dan cara menulis
4. HDFS tahan gagal: *replica*, rak, *heartbeat*
5. *Replication* vs *erasure coding*: hitung durabilitasnya
6. *Data locality* dan mengapa HDFS dirancang begitu
7. *Object storage*: S3, MinIO, dan mengapa ia menang di *cloud*
8. *Storage tiering* dan *lifecycle policy*
9. *Storage cost*, *retrieval cost*, *egress cost*
10. *File layout* dan praktikum MinIO

Sasaran pertemuan ini

Anda dapat menjelaskan bagaimana HDFS dan *object storage* menyimpan data dengan aman di atas mesin yang pasti rusak, menghitung *cost* dan durabilitas *replication* dibanding *erasure coding*, dan merancang kebijakan *tiering* beserta *cost*-nya.

Layer fisik

Dari disk ke data center

Media	Akses acak	*Cost* per GB
DRAM	$\approx$ 100 ns	paling mahal
NVMe SSD	puluhan–ratusan μs	mahal
SATA SSD	ratusan μs	sedang
HDD	$\approx$ 5–10 ms	murah
Pita magnetik	detik–menit	paling murah

Sumber: orde besaran, berbeda per generasi perangkat; cek spesifikasi terkini. ns: nanodetik; μs (dibaca “mikro”): mikrodetik.

- Ribuan server dipasang di **rak**; satu rak berbagi *switch* dan catu daya. Bila *switch* rak mati, seluruh rak hilang sekaligus.
- Rak dikelompokkan ke ruang, gedung, dan *availability zone*.
- Setiap tingkatan adalah satu ***fault domain***: satuan yang bisa gagal bersama.
- Pada ribuan disk, kerusakan disk terjadi **setiap hari**.

Ringkasan

Storage skala besar dirancang dengan asumsi: **disk, mesin, dan rak pasti rusak**. Pertanyaannya hanya bagaimana data tetap utuh dan tersedia ketika itu terjadi.

Hitungan kasar: seberapa sering disk rusak?

Misalkan (ilustrasi) tingkat kerusakan tahunan sebuah HDD sekitar 1–2%.

Cluster 3.000 disk

$$3.000 \times 1{,}5\% \approx 45 \text{ disk rusak per tahun}$$

$$\approx 1 \text{ disk per minggu}$$

Belum termasuk mesin mati, kabel lepas, pemeliharaan terjadwal, dan rak yang padam.

Akibat bagi rancangan

- Tidak boleh ada satu salinan data saja.
- Pemulihan harus **otomatis**; tidak ada yang sempat menangani manual setiap minggu.
- Salinan harus disebar ke *fault domain* berbeda, bukan hanya mesin berbeda.

Sumber: Laporan statistik disk yang dipublikasikan penyedia *storage* memberi angka tahunan per model; angka di kiri hanya ilustrasi.

Block, *file*, dan *object storage*

Tiga cara menyajikan *storage*

Jenis	Antarmuka	Contoh	Cocok untuk
Block	Deretan blok bernomor; OS memasang *file system* di atasnya	disk lokal, AWS EBS, iSCSI, Ceph RBD	*database*, *boot disk*
File	Hierarki direktori, POSIX (buka, baca, tulis di tengah, *lock*)	NFS, HDFS (mirip), CephFS	berbagi *file*, *home directory*
Object	`PUT`/`GET`/`DELETE` objek utuh lewat HTTP dengan *key*	Amazon S3, GCS, Azure Blob, MinIO, Ceph RGW	*data lake*, arsip, *backup*, media

Ciri *object storage*

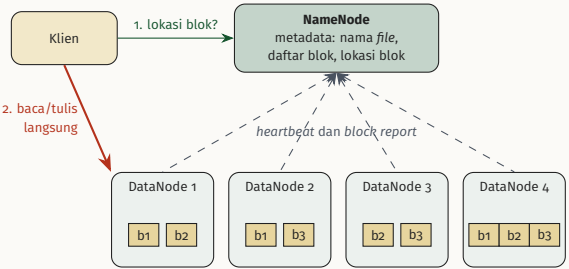
Objek tidak diubah di tengah; yang ada hanya ditulis ulang utuh. Ruang nama datar (*bucket* + *key*); "direktori" hanyalah awalan *key*.

Mengapa cocok untuk analitik

Data analitik ditulis sekali dan dibaca berkali-kali secara berurutan. Kemampuan "ubah di tengah *file*" hampir tidak pernah dibutuhkan.

HDFS: arsitektur dan cara menulis

HDFS dalam satu gambar



- Pola dirumuskan oleh GFS (Google, 2003); HDFS adalah versi *open source* di ekosistem Hadoop.
- *File* dipecah menjadi **blok** besar, *default* 128 MB.
- **NameNode**: satu pusat metadata, disimpan di memori.
- **DataNode**: menyimpan isi blok di disk lokal.
- Data **tidak** melewati NameNode; klien bicara langsung ke DataNode.
- Tiap blok direplikasi, *default* 3 salinan.

Sumber: Ghemawat dkk., *The Google File System*, SOSP 2003; HDFS Architecture Guide.

Mengapa blok sebesar 128 MB?

Alasan

- **Metadata sedikit**: NameNode menyimpan satu entri per blok di memori. Blok kecil berarti jutaan entri lebih banyak.
- **Baca berurutan panjang**: waktu *seek* HDD (milidetik) menjadi kecil dibanding waktu transfer satu blok.
- **Satuan kerja paralel**: satu blok biasanya menjadi satu *task* pemrosesan (Spark, P9).

Hitungan kasar

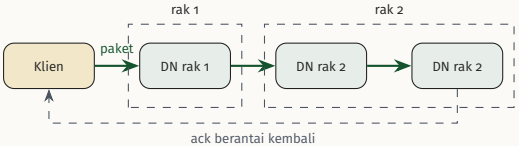
Membaca 128 MB pada $\approx$ 150 MB/s $\approx$ 0,85 detik; *seek* $\approx$ 10 ms hanya $\approx$ 1% dari waktu itu.

Dengan blok 4 KB, *seek* bisa mendominasi seluruh waktu baca.

Small files problem

Satu juta *file* 10 KB tetap menjadi satu juta objek metadata di NameNode, masing-masing memakan orde ratusan byte memori, dan satu juta *task* kecil saat diproses. Gabungkan *file* kecil.

Cara HDFS menulis: *replication pipeline*



1. Klien meminta NameNode mengalokasikan blok baru; NameNode membalas daftar tiga DataNode.
2. Klien mengirim paket kecil ke DataNode pertama, yang meneruskan ke kedua, lalu ketiga.
3. *Ack* mengalir balik; blok dianggap tertulis setelah seluruh *pipeline* mengonfirmasi.

Mengapa berantai?

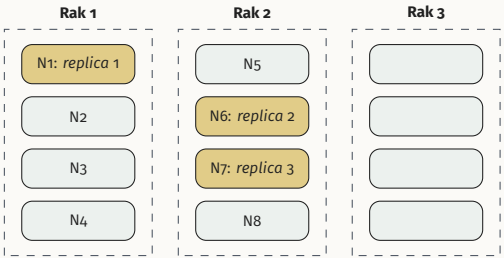
Klien hanya mengirim data **sekali**. Bandwidth keluar klien tidak terbagi tiga, dan tiap tautan jaringan hanya dilewati satu salinan.

Bila satu DataNode gagal di tengah

Pipeline ditutup, DataNode bermasalah dikeluarkan, penulisan dilanjutkan ke DataNode yang tersisa. Kekurangan *replica* diperbaiki belakangan oleh NameNode.

HDFS tahan gagal

Default replica placement policy



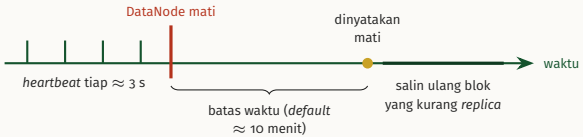
1. ***Replica* 1** di mesin penulis (bila penulis adalah DataNode), atau mesin acak.
2. ***Replica* 2** di **rak lain**.
3. ***Replica* 3** di rak yang sama dengan *replica* 2, mesin berbeda.

Mengapa tidak tiga rak?

Kompromi: data tetap selamat bila satu rak padam, sementara lalu lintas antar-rak (yang lebih mahal) hanya terjadi sekali per blok saat menulis.

NameNode mengetahui rak tiap mesin dari *configuration* ***rack awareness***. Tanpa itu, ketiga *replica* bisa jatuh di satu rak.

Heartbeat dan *re-replication*



- Tiap DataNode mengirim ***heartbeat*** berkala dan ***block report*** (daftar blok yang ia punya).
- Bila *heartbeat* berhenti melewati batas waktu, NameNode menandai DataNode mati dan menghitung blok mana yang *replica*-nya kurang dari target.
- DataNode lain diperintahkan menyalin blok itu dari *replica* yang masih ada.

Mengapa tidak langsung menyalin?

Mesin yang sedang *reboot* atau jaringan yang sesaat tersendat akan memicu badai penyalinan ribuan blok tanpa perlu. Batas waktu menyeimbangkan kecepatan pulih dan beban sia-sia.

Integritas

Tiap blok punya *checksum*. *Replica* yang rusak diam-diam (*bit rot*) terdeteksi saat dibaca atau dipindai, lalu diganti dari *replica* sehat.

Sumber: Nilai *default* dapat diatur; cek `hdfs-default.xml` versi yang dipakai.

NameNode: titik tunggal yang harus dilindungi

- Seluruh metadata ada di NameNode. Tanpa NameNode, blok di DataNode hanyalah potongan tanpa nama.
- Perubahan metadata dicatat di ***edit log***; keadaan lengkap disimpan berkala sebagai ***fsimage***.
- ***High availability***: NameNode *active* dan *standby* berbagi *edit log* lewat sekumpulan JournalNode yang memakai *quorum*; ZooKeeper membantu memilih siapa yang aktif.

Kaitan dengan Sistem Terdistribusi P7

Quorum pada JournalNode dan pemilihan pemimpin adalah pola *replication* dan *consistency* yang dibahas di Sister P7. Di sini cukup memahami **akibatnya**: satu NameNode boleh mati tanpa kehilangan metadata.

Batas skala

Karena metadata di memori satu mesin, jumlah *file* dan blok di satu *cluster* dibatasi memori NameNode. Ini salah satu alasan *object storage* lebih mudah diskalakan.

Replication vs erasure coding

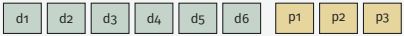
Dua cara bertahan dari kehilangan

Replication 3×



- Ruang: **3**× ukuran data (*overhead* 200%)
- Tahan kehilangan **2** salinan
- Pemulihan murah: salin satu blok utuh
- Baca cepat dari salinan mana pun

Erasure coding Reed–Solomon RS(6,3)



- 6 potongan data + 3 potongan paritas, disebar ke 9 mesin
- Ruang: **9/6 = 1,5**× (*overhead* 50%)
- Tahan kehilangan **3** potongan mana pun
- Pemulihan mahal: baca 6 potongan untuk membangun ulang 1

Ringkasan

Erasure coding memberi ketahanan lebih tinggi dengan separuh ruang, dibayar dengan CPU dan lalu lintas jaringan saat menulis dan memulihkan. HDFS 3 menyediakan kebijakan `RS-6-3-1024k`; *object storage* memakai pola serupa.

Matematika durabilitas (model sederhana)

Misalkan p = peluang satu mesin kehilangan datanya **sebelum** sempat dipulihkan, dan kegagalan antar-mesin saling bebas.

Replication 3×: hilang bila ketiganya hilang

$$P_{\text{hilang}} \approx p^3$$

RS(6,3): hilang bila $\geq$ 4 dari 9 potongan hilang

$$P_{\text{hilang}} \approx \binom{9}{4} p^4 = 126\, p^4$$

$\binom{9}{4}$: banyaknya cara memilih 4 potongan dari 9.

Contoh angka (ilustrasi, $p = 10^{-3}$)

	P_{hilang}	ruang
Replication 3×	10^{-9}	3,0×
RS(6,3)	$1{,}26 \times 10^{-10}$	1,5×

Mengapa model ini terlalu optimis

Kegagalan sering **berkorelasi**: satu rak padam membawa banyak mesin sekaligus. Karena itu penempatan per rak atau *fault domain* sama pentingnya dengan rumusnya. Makin cepat pemulihan, makin kecil p.

Space cost: hitungan untuk bank kita

Data analitik bank: 200 TB (ilustrasi).

Skema	Kapasitas mentah	Selisih
Replication 3×	600 TB	—
RS(6,3)	300 TB	−300 TB
RS(10,4)	280 TB	−320 TB

RS(10,4): 14 potongan, ruang 1,4×, tahan kehilangan 4, tetapi butuh minimal 14 mesin (idealnya 14 rak) dan pemulihan membaca 10 potongan.

When to use

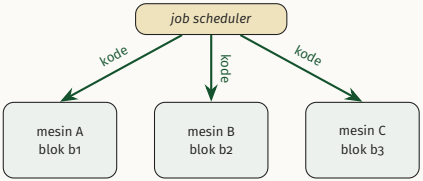
- ***Replication***: data panas yang sering dibaca dan ditulis, *file* kecil, *cluster* kecil dengan sedikit rak.
- ***Erasure coding***: data hangat sampai dingin yang besar dan jarang berubah; *cluster* dengan banyak mesin dan rak.

Praktik industri

Banyak platform memakai keduanya: *replication* untuk data baru, lalu dikonversi ke *erasure coding* setelah data mendingin.

Data locality

Kirim program ke data, bukan data ke program



program kecil berpindah; blok besar tetap di tempat

NameNode tahu letak setiap blok, sehingga *scheduler* dapat menjalankan *task* di mesin yang menyimpan bloknya (***node-local***), atau minimal di rak yang sama (***rack-local***).

Mengapa HDFS dirancang begini

Pada era Hadoop (pertengahan 2000-an), jaringan antar-rak jauh lebih lambat daripada membaca disk lokal. Menyatukan *storage* dan komputasi di mesin yang sama menghindari jaringan sebagai leher botol.

Yang berubah sesudahnya

Jaringan data center menjadi jauh lebih cepat, dan *cloud* menjual komputasi dan *storage* terpisah. Arsitektur modern banyak yang **memisahkan** keduanya: data di *object storage*, mesin komputasi boleh hidup-mati sesuai kebutuhan.

Object storage

Object storage: S3 dan kerabatnya

- ***Bucket*** berisi **objek**; tiap objek punya ***key***, isi, dan metadata.
- Diakses lewat HTTP: `PUT`, `GET` (termasuk *range read*), `LIST`, `DELETE`.
- ***Multipart upload***: objek besar dikirim per bagian secara paralel dan dapat dilanjutkan bila putus.
- Ketahanan dari *replication* atau *erasure coding* lintas *availability zone*, ditangani penyedia.
- Amazon S3 menjamin *strong read-after-write consistency* untuk objek sejak akhir 2020; dulu *eventual*, dan banyak alat lama dirancang untuk itu.

Tidak ada "ganti nama" yang murah

"Direktori" hanyalah awalan *key*. Memindahkan direktori berisi sejuta objek berarti sejuta salin-lalu-hapus. Inilah salah satu alasan lahirnya *table format* seperti Iceberg dan Delta (P4).

Pilihan *on-premise* dan lab

MinIO (S3-*compatible*, ringan, memakai *erasure coding* per set disk), Ceph RGW (bagian Ceph yang juga menyediakan *block* dan *file*), SeaweedFS, Garage. Periksa lisensi dan kebijakan distribusi terbaru sebelum memilih.

HDFS vs *object storage*: *when to use*

Aspek	HDFS	*Object storage*
Letak komputasi	menyatu dengan *storage* (*data locality*)	terpisah, lewat jaringan
Skala metadata	dibatasi memori NameNode	praktis tanpa batas, dikelola penyedia
Operasi	tim sendiri mengelola *cluster*	terkelola (*cloud*) atau MinIO/Ceph sendiri
Cost	bayar mesin walau menganggur	bayar per GB tersimpan dan per permintaan
Ganti nama direktori	cepat (operasi metadata)	mahal (salin dan hapus per objek)
Latency per operasi	rendah di jaringan lokal	lebih tinggi per permintaan HTTP

Praktik industri

Di *cloud*, *object storage* hampir selalu menjadi pilihan bawaan untuk *data lake*. HDFS masih dijumpai pada *cluster* Hadoop *on-premise* yang sudah lama berjalan, termasuk di sektor yang data-nya wajib berada di dalam negeri.

Storage tiering

Empat *tier* di dunia nyata

Tier	Contoh layanan	Akses	Simpan minimum	Data bank
Hot	SSD lokal, Redis, ClickHouse, S3 Standard	milidetik	tidak ada	7 hari terakhir
Warm	S3 Standard-IA, GCS Nearline	milidetik, ada *retrieval cost*	≈ 30 hari	13 bulan terakhir
Cold	S3 Glacier Instant Retrieval, GCS Coldline	milidetik, *retrieval cost* lebih tinggi	≈ 90 hari	2–5 tahun
Archive	S3 Glacier Deep Archive, GCS Archive, pita	jam	≈ 180 hari ke atas	bukti transaksi sampai 10 tahun

Storage cost turun …

makin dingin makin murah per GB-bulan, kira-kira satu orde besaran antara *hot* dan *archive* atau lebih.

…*retrieval cost* naik

cost per GB yang dibaca, *cost* per permintaan, durasi simpan minimum, dan waktu tunggu pada *archive*.

Sumber: Nama kelas dan durasi minimum mengikuti dokumentasi AWS S3 dan GCS; cek harga terkini.

Lifecycle policy: biarkan mesin yang memindahkan

```
{ "Rules": [ {
  "ID": "transaksi-tiering",
  "Filter": { "Prefix": "bronze/transaksi/" },
  "Status": "Enabled",
  "Transitions": [
    { "Days": 30,  "StorageClass": "STANDARD_IA" },
    { "Days": 395, "StorageClass": "GLACIER_IR" },
    { "Days": 1825,"StorageClass": "DEEP_ARCHIVE" }
  ],
  "Expiration": { "Days": 3653 }
} ] }
```

Format *lifecycle configuration* S3. MinIO memakai perintah `mc ilm` dengan gagasan yang sama.

Yang diatur

- ***Transition***: pindah kelas setelah umur tertentu
- ***Expiration***: hapus setelah masa retensi berakhir
- ***Legal hold*** dan ***object lock***: larang hapus atau ubah selama masa tertentu, misalnya untuk audit atau perkara

Retensi adalah keputusan kepatuhan

Berapa lama data wajib disimpan, dan kapan wajib dihapus, ditentukan regulasi dan kebijakan internal, bukan selera engineer. Angka di kiri hanya contoh; dibahas lagi bersama UU PDP di P15.

Cost: simpan, ambil, kirim

Tiga komponen *storage cost*

Simpan

per GB per bulan, tergantung kelas. Komponen paling terlihat, sering bukan yang terbesar.

Ambil dan permintaan

per GB dibaca dari kelas dingin, per 1.000 `PUT`/`GET`/`LIST`. Jutaan *file* kecil membuat *request cost* melonjak.

Egress

per GB keluar dari region atau dari *cloud*. Masuk umumnya gratis, keluar tidak. Memindahkan petabyte ke penyedia lain bisa sangat mahal.

Kasus yang sering terjadi

Data dipindah ke *archive* demi hemat, lalu tim analitik setiap bulan memindai ulang seluruh arsip untuk melatih model. *Retrieval cost* dan waktu tunggu melebihi penghematan. ***Tier* harus mengikuti pola akses nyata**, bukan hanya umur data.

Ringkasan

Egress dan pola akses menciptakan *lock-in* terselubung. Ukur dan perkirakan sebelum memilih tempat menyimpan data besar.

Hitungan: 10 TB log transaksi disimpan 7 tahun

Harga ilustrasi per GB-bulan (**bukan harga sungguhan**, hanya rasio kasar):

Hot	Rp 350
Warm	Rp 200
Cold	Rp 70
Archive	Rp 15

10 TB ≈ 10.000 GB; 7 tahun = 84 bulan. *Retrieval cost* dan permintaan diabaikan dulu.

Skenario A: semua *hot*

$10.000 \times 350 \times 84 \approx$ **Rp 294 juta**

Skenario B: bertingkat

Hot 1 bulan, *warm* 12, *cold* 23, *archive* 48:

$$10.000 \times (350 + 2.400 + 1.610 + 720)$$

$$= 10.000 \times 5.080 \approx \textbf{Rp 50,8 juta}$$

Selisih sekitar 5,8 kali. Tambahkan *retrieval cost* bila arsip sering dibaca, dan hitung ulang dengan harga terkini penyedia yang Anda pilih.

File layout

Partition gaya Hive dan masalah *file* kecil

```
s3://bank-lake/bronze/transaksi/
  tanggal=2026-10-18/
    part-0000.parquet   (~256 MB)
    part-0001.parquet
  tanggal=2026-10-19/
    part-0000.parquet
```

- Nilai *partition* ada di nama "direktori". *Query* `WHERE tanggal = '2026-10-19'` hanya membuka satu awalan.
- Pilih kolom *partition* yang **sering difilter** dan **kardinalitasnya rendah**: tanggal, bukan nomor rekening.

Small files problem lagi

Partition per jam per cabang per kanal bisa menghasilkan jutaan *file* berukuran KB. Akibatnya: `LIST` lambat, *request cost* tinggi, *query engine* membuka terlalu banyak *file*.

Praktik industri

- Target *file* ratusan MB sampai sekitar 1 GB untuk analitik.
- Lakukan *compaction* berkala (P4).
- Mulai dengan *partition* kasar; perhalus hanya bila *query* membuktikan perlu.

Praktikum

Praktikum 1: MinIO dengan Docker Compose

```yaml
services:
  minio:
    image: minio/minio
    command: server /data --console-address ":9001"
    environment:
      MINIO_ROOT_USER: admin
      MINIO_ROOT_PASSWORD: ganti-dengan-rahasia
    ports: ["9000:9000", "9001:9001"]
    volumes: ["./data:/data"]
```

Jangan pernah menaruh kata sandi sungguhan di *file* yang masuk Git; pakai *file* `.env` yang di-`gitignore` (dibahas di P14).

```python
import boto3
s3 = boto3.client("s3",
    endpoint_url="http://localhost:9000",
    aws_access_key_id="admin",
    aws_secret_access_key="ganti-dengan-rahasia")
s3.create_bucket(Bucket="bank-lake")
s3.upload_file("trx.parquet", "bank-lake",
  "bronze/transaksi/tanggal=2026-10-19/"
  "part-0000.parquet")
```

Praktikum 2: *query* langsung ke *object storage*

```
INSTALL httpfs; LOAD httpfs;
SET s3_endpoint = 'localhost:9000';
SET s3_use_ssl = false;
SET s3_url_style = 'path';
SET s3_access_key_id = 'admin';
SET s3_secret_access_key = 'ganti-dengan-rahasia';

SELECT tanggal, kanal, SUM(nominal) AS total
FROM read_parquet(
  's3://bank-lake/bronze/transaksi/*/*.parquet',
  hive_partitioning = true)
WHERE tanggal = '2026-10-19'
GROUP BY ALL;
```

Eksperimen yang diukur

1. Unggah 30 hari data dalam *partition* tanggal.
2. Bandingkan waktu *query* satu hari vs seluruh 30 hari.
3. Pecah satu hari menjadi 1.000 *file* kecil; ukur lagi. Jelaskan bedanya.
4. Pasang aturan `mc ilm` untuk *expiration* dan catat hasilnya.

Penutup

Rangkuman

- Disk, mesin, dan rak pasti rusak; *storage* skala besar dirancang untuk itu.
- HDFS: NameNode memegang metadata, DataNode memegang blok 128 MB, *replica* ditulis lewat *pipeline* dan disebar ke dua rak.
- *Heartbeat*, *block report*, dan *checksum* memicu *re-replication* otomatis.
- RS(6,3): ruang 1,5×, tahan 3 kehilangan; *replication* 3×: ruang 3×, tahan 2 kehilangan.
- *Data locality* menjelaskan rancangan HDFS; jaringan cepat dan *cloud* mendorong pemisahan *compute* dan *storage*.
- *Object storage* adalah fondasi *data lake* modern.
- *Tiering* menurunkan *storage cost*, tetapi *retrieval cost*, durasi minimum, dan *egress* harus ikut dihitung.
- *Partition* seperlunya; hindari jutaan *file* kecil.

Tugas untuk pertemuan berikutnya

1. Laporan praktikum: tabel waktu *query* (satu hari vs 30 hari vs 1.000 *file* kecil) beserta penjelasannya.
2. Hitung *cost* menyimpan 10 TB log transaksi selama 7 tahun untuk tiga skenario *tiering* pilihan Anda, memakai **harga terkini** satu penyedia *cloud*. Sertakan *retrieval cost* bila tim analitik membaca 1 TB data dingin per bulan.
3. Hitung kapasitas mentah dan peluang kehilangan (model sederhana) untuk 200 TB data dengan *replication* $3\times$, RS(6,3), dan RS(10,4) bila $p = 10^{-3}$.
4. Baca Reis dan Housley Bab 6 dan Armbrust dkk., *Lakehouse* (CIDR 2021) bagian 1–3.

Pertemuan depan

Dari folder Parquet ke tabel sungguhan: *data warehouse*, *data lake*, dan *lakehouse* (Iceberg, Delta), lalu *medallion architecture* bronze, silver, gold.

Terima kasih

Pertanyaan?