

# Eksperimen Mekanisme Distributed Object Storage menggunakan MinIO

---

## Highlight

Menginstal dan mengoperasikan MinIO Distributed Mode ( $\geq 4$  node).

Memahami konsep inti distributed storage:

- Erasure coding (EC)
- Quorum read/write
- Object replication & consistency
- Distributed lock & metadata management

Mendesain skenario eksperimen failure dan recovery:

- Node crash
- Disk loss
- Network partition

Mengukur performa dasar dan menganalisis trade-off (latency, read/write quorum). Menghubungkan fenomena eksperimen dengan teori distributed systems:

- CAP theorem
  - Availability vs consistency
  - Durability
  - Fault tolerance
- 

## Tugas Project

Buatlah dan jalankan skenario sebuah cluster MinIO distributed dengan:

- Minimal 4 node (4 VM, 4 container, atau campuran)
  - Menggunakan Erasure Coding bawaan MinIO (misal 4 data + 2 parity)
  - Klien mengakses melalui MinIO client (mc) atau SDK
  - Semua konfigurasi dicatat dan dijelaskan dalam laporan.
  - Tugas berkelompok (3 orang)
  - Mendesain skenario eksperimen yang berhubungan dengan mekanisme distributed systems pada Minio
- 

## Skenario

### Erasure Coding (EC) & Sharding

MinIO menggunakan erasure coding (misal EC:4+2) untuk ketahanan data. Objek akan didistribusi ke beberapa disk/node. **Observasi:**

- Bagaimana objek dipecah menjadi shards.
- Apa yang terjadi jika 1-2 node hilang?
- Apakah cluster tetap bisa baca/tulis?

## Consistency Model

MinIO menggunakan:

- Read-after-write consistency untuk objek baru.
- Quorum-based consistency:
  - Write quorum = sebagian besar disk/shards
  - Read quorum = sebagian shards minimal **Observasi:**
- Apa terjadi jika sebagian node lambat atau gagal?
- Apakah write tetap bisa dilakukan?
- Apakah read tetap konsisten?

## Distributed Metadata & Locking

MinIO menggunakan distributed object metadata yang disimpan di setiap node. Lock (multipart upload locking) bersifat distributed. **Observasi:**

- Apakah metadata tersinkronisasi ketika node mati?
- Bagaimana recovery metadata berjalan?

## Failover & Healing

MinIO mempunyai:

- Self-healing otomatis
- Data reconstruction jika shards hilang
- Node replacement **Observasi:**
- Matikan satu node, apakah cluster masih bekerja?
- Matikan dua node, apakah cluster masih bisa baca? bisa tulis?
- Hidupkan node kembali, apakah proses healing berjalan?
- Hapus satu disk, apakah data dapat direkonstruksi?

---

## Skenario 1: Write Quorum & Konsistensi

### Aspek

- Quorum write
- Eventual vs strong consistency
- Latency vs durability trade-off

**Tujuan** Menganalisis bagaimana MinIO menangani:

- write quorum
- read quorum
- behavior saat satu node lambat atau mati

**Langkah:**

1. Buat cluster MinIO 4-node (EC 2+2 atau 4+2 jika memakai 6 disk).
2. Upload file besar (misal 200MB).
3. Buat satu/lebih node menjadi lambat (sehingga bisa menjadi tidak konsisten).
4. Upload 10 file secara paralel.

**Observasi:**

- Apakah upload gagal?
  - Apakah file langsung konsisten (di seluruh node ?)
  - Cek dari node lain, apakah hasilnya sama?
- 

## Skenario 2: Node Failure & Self-Healing

**Aspek:**

- Fault tolerance
- Erasure coding reconstruction
- Self-healing mechanism

**Tujuan** Menguji kemampuan cluster bertahan dari kegagalan node dan memperbaiki data.

**Langkah:**

1. Upload beberapa objek ( $\geq 20$  file).
  2. Matikan 1 node MinIO.
  3. Lakukan operasi baca dan tulis:
    - Apakah read tetap jalan?
    - Apakah write masih diizinkan?
  4. Hidupkan node kembali.
  5. Cek log cluster:
    - Apakah proses self-healing berjalan?
    - Apakah data direkonstruksi?
- 

## Skenario 3

**Tujuan** Mengamati reaksi cluster ketika disk di salah satu node rusak.

**Aspek**

- Durability
- Redundancy
- Recovery & rebuild

**Langkah:**

1. Hapus atau rename directory disk di satu node (misal /data/disk1/).
2. Restart node tersebut.

3. Upload dan download file.

**Observasi:**

- Apakah cluster mendeteksi disk failure?
  - Apakah objek lama tetap bisa dibaca?
  - Apakah cluster rekonstruksi shards yang hilang?
- 

## Pembagian Tugas

### Setup cluster & Arsitektur

- Menyiapkan cluster MinIO (Docker / VM).
- Mendesain topologi:
  - Jumlah node
  - Disk layout (EC: 2+2, 4+2, dsb)
- Dokumentasi konfigurasi:
  - Docker-compose.yml (jika memakai docker)
  - MinIO environment variables
  - Metadata layout diagram
- **Laporan yang ditulis:**
  - Arsitektur sistem
  - Penjelasan konsep Erasure Coding, quorum, metadata replication

### Eksperimen Failure & Recovery

- Bertanggung jawab menjalankan skenario failure:
  - Node crash
  - Disk deletion
  - Latency injection (membuat sebuah node menjadi lambat sehingga menambah latency, bisa menggunakan tc qdisc atau netem)
- Mengambil log dan screenshot
- Menganalisis hasil terkait:
  - Fault tolerance
  - Self-healing
  - CAP theorem (CP vs AP)
- **Laporan yang ditulis:**
  - Failure scenarios
  - Analisis failover behavior
  - Healing mechanism

### Client Workload & Measurement

- Membuat script client (Python/Go/NodeJS atau mc scripting):
  - Upload otomatis
  - Download otomatis
  - Checksum verification
- Mengambil metrik:

- Upload/download time
    - Behavior saat node mati
    - Konsistensi hasil read
  - Grafik sederhana
  - **Laporan yang ditulis:**
    - Konsistensi & performa
    - Analisis read/write quorum
    - Kesimpulan eksperimen
- 

## Isi Laporan

### Kode & Konfigurasi

- docker-compose.yml
- Script client
- Script simulasi failure

### Laporan (PDF)

- Penjelasan teori
- Topologi cluster
- Skenario eksperimen (dilengkapi tabel: tujuan, langkah, hasil, analisis)
- Hasil log & screenshot
- Analisis berdasarkan konsep distributed systems pada masing-masing skenario
- Kontribusi masing-masing anggota

### Demo Presentasi

- Jalankan cluster secara live
- Jalankan skenario failure
- Tunjukkan hasil recovery & konsistensi data