F1E Praktischer Nachweis

> Ich kann ein Konzept für die Skalierung einer NoSQL Datenbank erstellen.

Beispielszenario: E-Commerce Plattform

Beschreibung: Eine wachstumsstarke E-Commerce-Plattform mit globaler Nutzerbasis.

Datenstruktur:

- `users` → Kundendaten
- `products` → Katalog von Artikeln
- `orders` → Bestellungen
- `sessions` → temporäre Logins & Warenkörbe

Anforderungen:

- Hohe Verfügbarkeit (24/7)
- Globale Lesezugriffe
- Schreibintensive Region in Europa
- Konsistente Daten bei Bestellungen
- Backup und Disaster Recovery

Skalierungskonzept mit Replica Sets

1. Replica Set Aufbau

```
Replica Set Name: rs0
Mitglieder:
- Primary: Frankfurt (eu-central-1)
- Secondary 1: Paris (eu-west-3)
- Secondary 2: Dublin (eu-west-1)
- Arbiter: Stockholm (eu-north-1)
```

Begründung:

- Primary ist nahe dem grössten Schreibaufkommen (Europa)
- Secondaries in getrennten Zonen für Redundanz
- Arbiter sorgt für Quorum bei Ausfall eines Nodes, ohne Extra-Datenhaltung

2. Vertikale Skalierung (Scale-Up)

Wann anwenden?

- Wenn CPU, RAM oder IO-Bandbreite am Limit sind
- Datenbankgrösse < 2 TB
- Schreibdurchsatz kann nicht verteilt werden

Massnahmen:

- SSD statt HDD verwenden
- RAM ausbauen (MongoDB profitiert stark von viel RAM)
- `wiredTigerCacheSizeGB` anpassen
- Schnellere CPUs einsetzen
- Disk-IOPS erhöhen

Vertikale Skalierung ist schnell umsetzbar, aber begrenzt und teuer.

3. Horizontale Skalierung (Scale-Out)

Wann anwenden?

- Replikation alleine reicht nicht mehr aus
- Lese- und Schreiblast wächst kontinuierlich
- Datenvolumen > 2–3 TB oder sehr grosse/heisse Collections

Massnahme: Kombination von Replica Sets mit Sharding:

```
Sharded Cluster:

            mongos
              |
     ------------------
     |        |       |
  shard1   shard2   shard3
   rs0      rs1      rs2
```

Sharding-Konzept:

- Jeder **Shard** ist ein eigenes **Replica Set**.
- Der **Shard-Key** (z. B. `userId`, `region`, `orderId`) bestimmt, wie die Daten auf die Shards verteilt werden.
- **Global verteilbare Daten** (z. B. Nutzerprofile, Sessions) werden über alle Shards verteilt.

- **Kritische, konsistente Daten** (z. B. Zahlungsdaten, Bestellungen) können gezielt auf einem einzelnen Shard gehalten werden.
- **Horizontale Skalierung** durch Sharding ist komplexer in der Verwaltung, ermöglicht aber nahezu unbegrenztes Wachstum und Lastverteilung.

2. Konfiguration (mongod)

Alle Nodes starten mit:

```
mongod --replSet rs0 --bind_ip_all
```

Initialisierung per Mongo Shell:

```
rs.initiate({
  _id: "rs0",
  members: [
    { _id: 0, host: "mongo1:27017", priority: 2 },     // Primary
    { _id: 1, host: "mongo2:27017", priority: 1 },     // Secondary
    { _id: 2, host: "mongo3:27017", priority: 1 },     // Secondary
    { _id: 3, host: "mongo-arbiter:27017", arbiterOnly: true }
  ]
})
```

4. Lesen & Schreiben optimieren

- **Reads:**
  - *Read Preference*: `nearest` → Optimale globale Performance
- **Writes:**
  - *Write Concern*: `majority` → Hohe Konsistenz
- **Analysen:**
  - Auf Secondary-Knoten, ggf. mit verzögerter Replikation

5. Betrieb & Monitoring

- **MongoDB URI:**

  ```
  MONGODB_URI=mongodb://mongo1,mongo2,mongo3/mydb?replicaSet=rs0
  ```

- **Monitoring:**
  - Prometheus + Grafana (Self-Hosted)
  - MongoDB Atlas (Managed)
  - Alerts bei:

    - Latenz > 100 ms
    - Repl-Lag > 5 s
    - Disk usage > 80 %

6. Backup & Recovery

- **Strategie:**
    - `mongodump` auf Secondary oder Hidden Node
    - Snapshots über Volume-Provider
    - Tägliche Dumps & wöchentliche Snapshots
    - Disaster-Recovery-Test 1× pro Monat

7. Failover & Maintenance

- **Replica Set ermöglicht:**
    - Automatisches Failover (~10 s)
    - Zero-Downtime Rolling Upgrades
    - Wartung einzelner Nodes ohne Downtime

8. Langfristiger Plan

- **Phase 1:** Replica Set + Vertikale Skalierung (bis ca. 1–2 TB)
- **Phase 2:** Sharded Cluster (je ein Replica Set pro Shard)
- **Phase 3:** Geo-Sharding für Nutzer in z. B. USA / APAC