Лабораторна робота №4

Налаштування реплікації та перевірка відмовостійкості MongoDB

ОС: Kali Linux | Середовище: Docker + MongoDB Replica Set

Мета роботи

Ознайомитись із реплікацією даних у MongoDB, розгорнути Replica Set у конфігурації Primary–Secondary–Secondary (P–S–S), перевірити роботу writeConcern/readConcern, продемонструвати перевибори Primary (elections) та проаналізувати продуктивність під паралельним навантаженням.

Вихідні умови та конфігурація

Розгортання виконано у Docker-контейнерах з трьома вузлами MongoDB 6.0 та окремим контейнером клієнта.

Replica Set: rs0 (3 члени: mongo1, mongo2, mongo3).

Connection string:

mongodb://mongo1:27017,mongo2:27017,mongo3:27017/?replicaSet=rs0

Підтвердження інструментів (Docker/Compose)

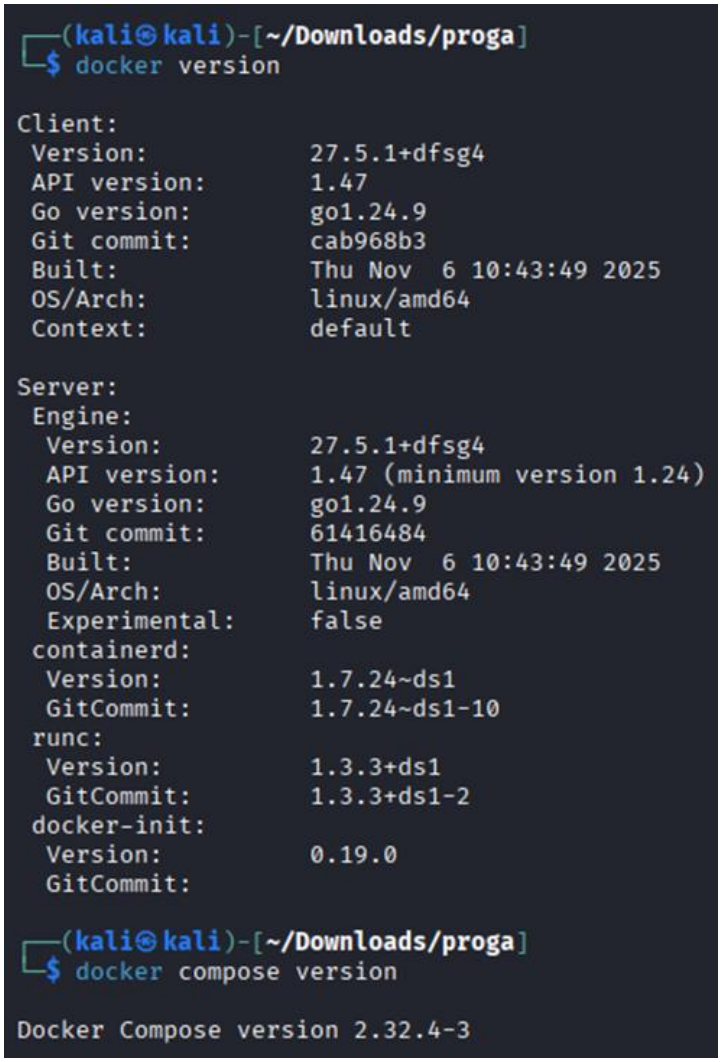

Рис. 1 — Перевірка версій Docker Engine та Docker Compose.

I. Налаштування реплікації та перевірка відмовостійкості

I.1. Підняття контейнерів та ініціалізація Replica Set

1) Запуск контейнерів та перевірка, що всі ноди у статусі Up.

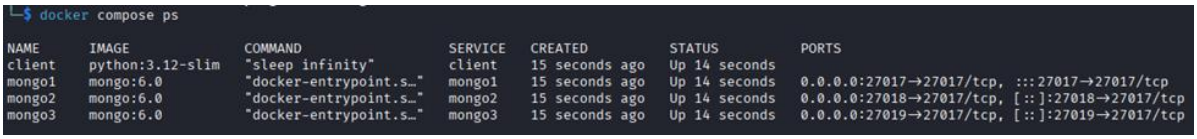

Рис. 2 — Стан контейнерів після запуску (mongo1/mongo2/mongo3 + client).

2) Ініціалізація Replica Set (rs.initiate).

```
└─$ docker exec -it mongo1 mongosh --quiet --eval '
rs.initiate({
  _id: "rs0",
  members: [
    { _id: 0, host: "mongo1:27017" },
    { _id: 1, host: "mongo2:27017" },
    { _id: 2, host: "mongo3:27017" }
  ]
})
'

{ ok: 1 }
```

Рис. 2а — Ініціалізація Replica Set rs0 командою rs.initiate(), результат ok:1.

3) Перевірка ролей нод (PRIMARY/SECONDARY) та health.

```
└─$ docker exec -it mongo1 mongosh --quiet --eval '
rs.status().members.map(m => ({name:m.name, stateStr:m.stateStr, health:m.health}))
'

[
  { name: 'mongo1:27017', stateStr: 'PRIMARY', health: 1 },
  { name: 'mongo2:27017', stateStr: 'SECONDARY', health: 1 },
  { name: 'mongo3:27017', stateStr: 'SECONDARY', health: 1 }
]
```

Рис. 3 — Статус Replica Set: mongo1=PRIMARY, mongo2/mongo3=SECONDARY, health=1.

I.2. Запис при відключеній ноді та writeConcern w=3

Мета: перевірити поведінку при вимозі підтвердження запису від усіх 3 нод (w=3) у ситуації, коли одна з нод тимчасово недоступна.

1) Відключення однієї Secondary (mongo3).

```
NAME      IMAGE              COMMAND                  SERVICE   CREATED          STATUS          PORTS
client    python:3.12-slim   "sleep infinity"         client    2 minutes ago    Up 2 minutes
mongo1    mongo:6.0          "docker-entrypoint.s…"   mongo1    2 minutes ago    Up 2 minutes    0.0.0.0:27017→27017/tcp, :::27017→27017/tcp
mongo2    mongo:6.0          "docker-entrypoint.s…"   mongo2    2 minutes ago    Up 2 minutes    0.0.0.0:27018→27017/tcp, [::]:27018→27017/tcp
```

Рис. 4 — Після docker stop mongo3 активні лише mongo1 та mongo2.

2) Запис з writeConcern { w: 3, wtimeout: 0 }. Значення wtimeout=0 означає відсутність таймауту (операція чекатиме на досягнення write concern). У момент, коли mongo3 був вимкнений, команда очікувала; після підняття mongo3 (docker start mongo3) запис завершився успішно.

```
rs0 [direct: primary] lab4> db.wc_test.drop()
...
false
rs0 [direct: primary] lab4> db.wc_test.insertOne(
...   { _id: 1, msg: "w3_infinite", ts: new Date() },
...   { writeConcern: { w: 3, wtimeout: 0 } }
... )
{ acknowledged: true, insertedId: 1 }
rs0 [direct: primary] lab4>
```

Рис. 5 — InsertOne з writeConcern w=3, wtimeout=0 завершився успішно після відновлення ноди.

I.3. writeConcern w=3 зі скінченним таймаутом та перевірка читання (readConcern: majority)

1) Запис з writeConcern { w: 3, wtimeout: 5000 } при вимкненій ноді призводить до таймауту очікування реплікації.

```
rs0 [direct: primary] lab4> db.wc_test.insertOne(
...   { _id: 2, msg: "w3_timeout_5s", ts: new Date() },
...   { writeConcern: { w: 3, wtimeout: 5000 } }
... )
Uncaught:
MongoWriteConcernError[WriteConcernFailed]: waiting for replication timed out
Additional information: {
  wtimeout: true,
  writeConcern: { w: 3, wtimeout: 5000, provenance: 'clientSupplied' }
}
Result: {
  n: 1,
  electionId: ObjectId('7fffffff0000000000000001'),
  opTime: { ts: Timestamp({ t: 1766480545, i: 1 }), t: 1 },
  writeConcernError: {
    code: 64,
    codeName: 'WriteConcernFailed',
    errmsg: 'waiting for replication timed out',
    errInfo: {
      wtimeout: true,
      writeConcern: { w: 3, wtimeout: 5000, provenance: 'clientSupplied' }
    }
  },
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1766480545, i: 1 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: 0
    }
  },
  operationTime: Timestamp({ t: 1766480545, i: 1 })
}
```

Рис. 6 — Помилка WriteConcernFailed: waiting for replication timed out (w=3, wtimeout=5000).

2) Попри таймаут writeConcern, документ може бути записаний на Primary; це підтверджено переглядом колекції wc_test.

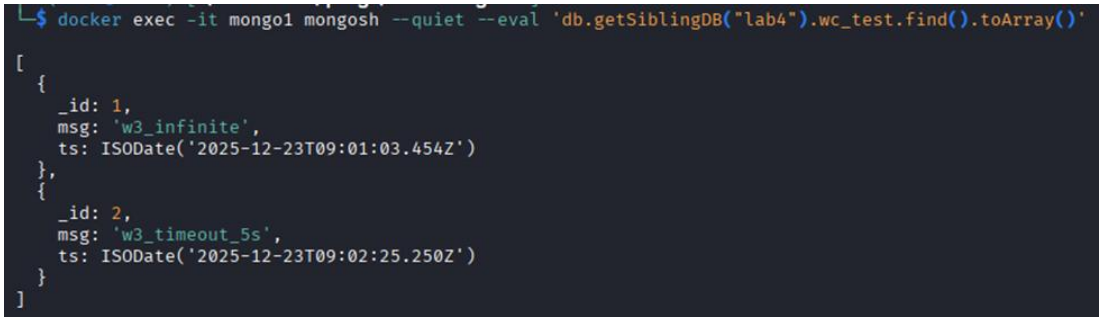

Рис. 7 — У колекції wc_test присутній документ з _id=2 (w3_timeout_5s).

3) Контроль PRIMARY ноди через rs.status().

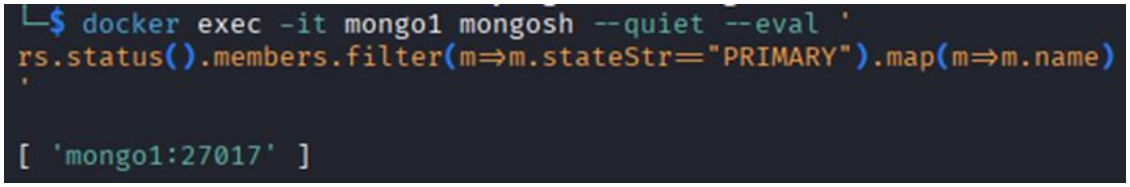

Рис. 8 — Визначення поточної PRIMARY ноди через rs.status().

Примітка (readConcern="majority"): після відновлення роботи третьої ноди та синхронізації реплікації документ стає доступним для majority-читання. Типова команда перевірки:

db.wc_test.find({ _id: 2 }).readConcern("majority")

I.4. Replica Set Elections: перевибори Primary та реплікація після відновлення старої Primary

Примусово відключено поточну Primary (docker kill). Replica Set обрав нову Primary, на якій виконано запис. Далі відновлено вимкнену ноду та перевірено, що вона отримала дані через реплікацію.

Рис. 9 — Після падіння mongo1: mongo2=PRIMARY, mongo3=SECONDARY, mongo1=not reachable/healthy.

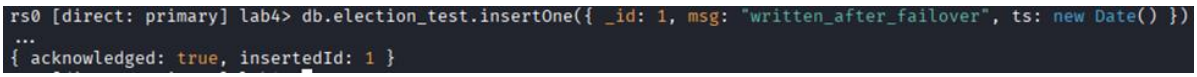

Рис. 10 — Запис у election_test під час простою старої Primary (acknowledged=true).

Рис. 11 — Після відновлення mongo1 (SECONDARY) дані репліковані: findOne повертає документ.

II. Аналіз продуктивності та перевірка цілісності (10 клієнтів × 10 000 інкрементів)

Для оновлення лічильника використано атомарну операцію findOneAndUpdate з $inc (без lost updates). У кожному сценарії виконано 10 паралельних клієнтів по 10 000 інкрементів (очікувано 100 000). Фінальне значення зчитувалось з readConcern="majority".

Приклад операції інкременту:

db.likes.findOneAndUpdate({ _id: "post1" }, { $inc: { likes: 1 } }, { upsert: true })

II.1. Методика вимірювання

Запуски виконувались командами:

docker exec -it client python /work/load_test.py --wc 1 --reset
docker exec -it client python /work/load_test.py --wc majority --reset

II.2. Результати експериментів

Сценарій	writeConcern	Час, с	final_likes (majority) / expected
E1	1	66.255	99973 / 100000
E2	majority	142.904	100000 / 100000
E3	1	65.406	99960 / 100000
E4	majority	160.145	100008 / 100000

Скріншоти запусків та результатів:

```
└─$ docker exec -it client python /work/load_test.py --wc 1 --reset

writeConcern=1 | clients=10 iters=10000
time_sec=66.255
final_likes(readConcern=majority)=99973 | expected=100000
```

Рис. 12 — writeConcern=1.

```
└─$ docker exec -it client python /work/load_test.py --wc majority --reset

writeConcern=majority | clients=10 iters=10000
time_sec=142.904
final_likes(readConcern=majority)=100000 | expected=100000
```

Рис. 13 — writeConcern=majority.

```
└─$ docker exec -it client python /work/load_test.py --wc 1 --reset

writeConcern=1 | clients=10 iters=10000
time_sec=65.406
final_likes(readConcern=majority)=99960 | expected=100000
```

Рис. 14 — writeConcern=1 (повторний прогін).

```
└─$ docker exec -it client python /work/load_test.py --wc majority --reset

writeConcern=majority | clients=10 iters=10000
time_sec=160.145
final_likes(readConcern=majority)=100008 | expected=100000
```

Рис. 15 — writeConcern=majority (повторний прогін).

II.3. Відмова Primary під час навантаження (failover)

Під час виконання тестів Primary нода примусово вимикалась командою docker kill. Replica Set виконує перевибори (elections) і призначає нову Primary.

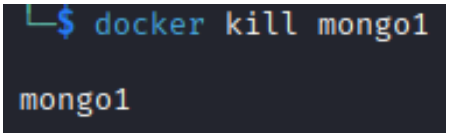

Рис. 16 — docker kill mongo1.

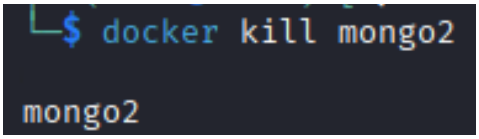

Рис. 17 — docker kill mongo2.

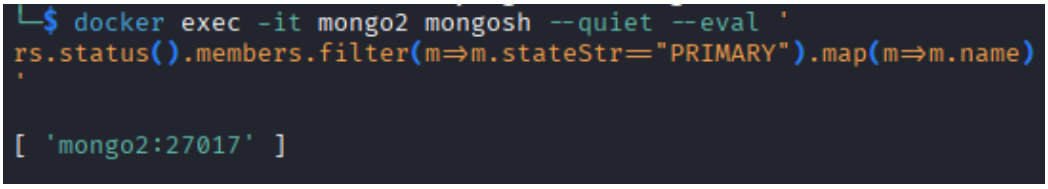

Рис. 18 — Перевірка поточної PRIMARY ноди через rs.status() після відмови.

II.4. Аналіз отриманих результатів

1) Продуктивність: writeConcern=1 працює швидше (≈65–66 с), оскільки підтвердження повертається після запису на Primary. writeConcern=majority повільніший (≈143–160 с), бо Primary очікує підтвердження запису від більшості нод.

2) Узгодженість: зчитування виконувалось з readConcern="majority". У сценаріях з writeConcern=1 можливі тимчасові відхилення у фінальному значенні (реплікаційний лаг), а під час відмови Primary — втрата частини операцій, що не стали majority-комітнутими. Для writeConcern=majority очікується значення 100000, оскільки кожна операція підтверджується лише після запису на більшість нод.

3) Випадок final_likes > expected (E4: 100008): інкремент ($inc) не є ідемпотентною операцією. Під час elections/перепідключення можливі повторні спроби клієнта (retry) після мережевих помилок, коли попередня операція вже була застосована на сервері, але відповідь не була отримана клієнтом. Такі повтори можуть додатково збільшити лічильник.

Висновки

1) Replica Set у конфігурації P–S–S успішно розгорнуто та ініціалізовано; стан нод підтверджено статусом rs.status().

2) Для writeConcern w=3 при відключеній ноді операція або очікує нескінченно (wtimeout=0), або завершується таймаутом (wtimeout=5000).

3) Перевибори Primary при відмові поточної Primary відбуваються автоматично; після відновлення стара Primary стає Secondary і синхронізує дані.

4) writeConcern=1 дає кращу швидкодію, але слабші гарантії majority-стійкості під час відмов; writeConcern=majority повільніший, зате надійніший.