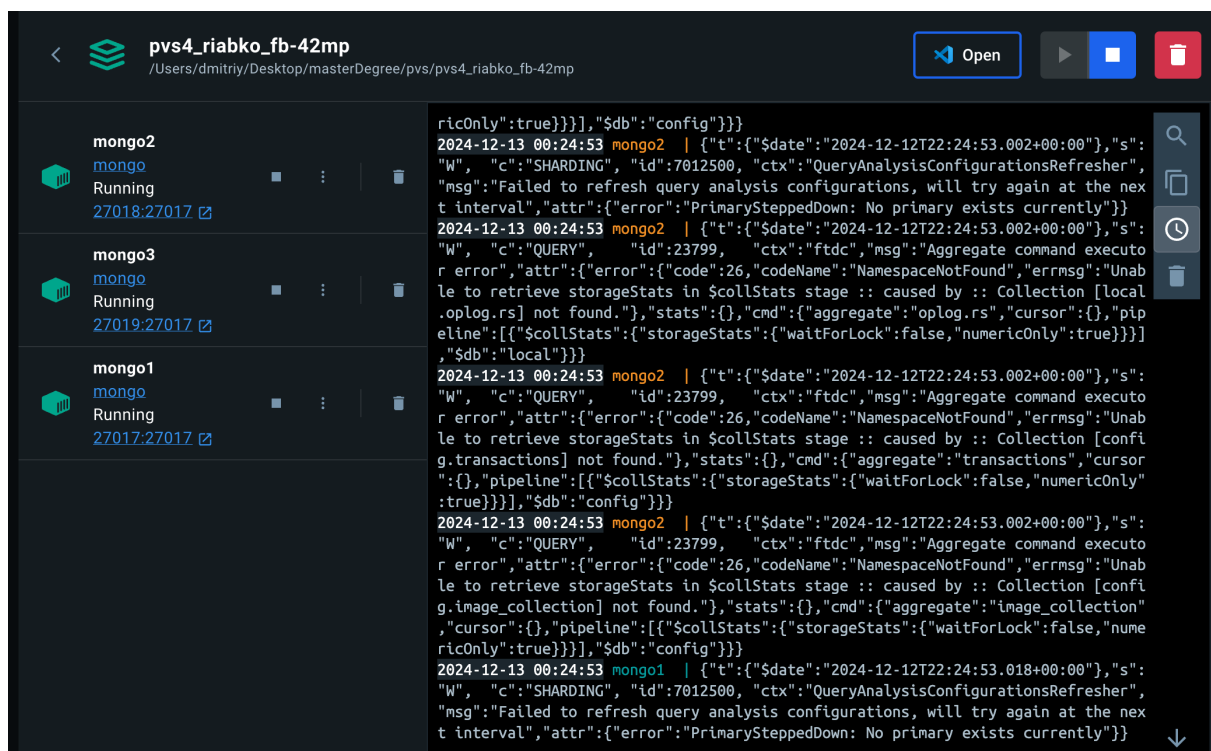


Рябко Дмитро ФБ-42мп

I Налаштування реплікації

1. Налаштувати реплікацію в конфігурації: Primary with Two Secondary Members (P-S-S) (всі ноди можуть бути запущені як окремі процеси або у Docker контейнерах) - <http://docs.mongodb.org/manual/core/replica-set-architecture-three-members/>
 - Deploy a Replica Set for Testing and Development-
<http://docs.mongodb.org/manual/tutorial/deploy-replica-set-for-testing/>
 - <http://www.tugberkugurlu.com/archive/setting-up-a-mongodb-replica-set-with-docker-and-connecting-to-it-with-a-net-core-app>



The screenshot shows a Docker Desktop window titled 'pvs4_riabko_fb-42mp'. It displays three MongoDB containers: 'mongo2', 'mongo3', and 'mongo1', all in a 'Running' state. The logs for 'mongo2' are visible, showing a 'PrimarySteppedDown' error at 2024-12-13 00:24:53. The error message is: 'PrimarySteppedDown: No primary exists currently'. The logs also show other MongoDB messages, including 'Failed to refresh query analysis configurations' and 'Aggregate command execution error'.

```
dmitriy@Dmitriys-MacBook-Air ~ % mongo --host localhost --port 27017
zsh: command not found: mongo
dmitriy@Dmitriys-MacBook-Air ~ % mongosh --host localhost --port 27017
Current Mongosh Log ID: 675b63b9cf1c73a045ab55ca
Connecting to:      mongodb://localhost:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.3.4
Using MongoDB:      8.0.4
Using Mongosh:       2.3.4

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/

To help improve our products, anonymous usage data is collected and sent to MongoDB periodically (https://www.mongodb.com/legal/privacy-policy).
You can opt-out by running the disableTelemetry() command.

-----
The server generated these startup warnings when booting
2024-12-12T22:21:22.951+00:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://dochub.mongodb.org/core/prodnotes-filesystem
2024-12-12T22:21:22.951+00:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
2024-12-12T22:21:22.951+00:00: For customers running the current memory allocator, we suggest changing the contents of the following sysfsFile
2024-12-12T22:21:22.951+00:00: We suggest setting the contents of sysfsFile to 0.
2024-12-12T22:21:22.951+00:00: Your system has glibc support for rseq built in, which is not yet supported by tcmalloc-google and has critical performance implications. Please set the environment variable GLIBC_TUNABLES=glibc.pthread.rseq=0
2024-12-12T22:21:22.951+00:00: vm.max_map_count is too low
2024-12-12T22:21:22.951+00:00: We suggest setting swappiness to 0 or 1, as swapping can cause performance problems.
```

Налаштовуєм реплікацію:

```

MongoServerError[NotYetInitialized]: no replset config has been received
test> rs.initiate({
...   _id: "rs0",
...   members: [
...     { _id: 0, host: "mongo1:27017" },
...     { _id: 1, host: "mongo2:27017" },
...     { _id: 2, host: "mongo3:27017" }
...   ]
... });
{
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1734042659, i: 1 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1734042659, i: 1 })
}

```

2. Спробувати зробити запис з однією відключеною ногою та *write concern* рівнім 3 та нескінченім таймаутом. Спробувати під час таймаута включити відключену ноду

Підключаємось до однієї з нод:

Ось Primary нода:

```

{
  _id: 2,
  name: 'mongo3:27017',
  health: 1,
  state: 1,
  stateStr: 'PRIMARY',
  uptime: 741,

```

Підключимось до цієї ноди:

```

root@mongo3:~# mongo
dmitriy@Dmitrys-MacBook-Air ~ % mongosh --host localhost --port 27017

```

```

db.test.insertOne( { name: "test_1", time: new Date() }, {
writeConcern: { w: 3, wtimeout: 0 } } );













```

```

[rs0 [direct: primary] test> db.test.insertOne( { name: "test_1", time: new Date() }, { writeConcern: { w: 3, wtimeout: 0 } } );
{
  acknowledged: true,
  insertedId: ObjectId('675b6cf4cd0d6dc01f61d6f3')
}
[rs0 [direct: primary] test>

```

Зупинимо одну з нод:

| | | | |
|---|--|---|--|
|  | mongo2 mongo Running 27018:27017 ↗ |    | , "host 2024-1 "I", "RSM m "mongo 2024-1 |
|  | mongo3 mongo Running 27019:27017 ↗ |    | "I", :{"hos 2024-1 "I", after ":{"co 17 :: voke : 2024-1 "I", "RSM r ble: E mongo1 |
|  | mongo1 mongo Exited (137) 27017:27017 |    | |













Та виконаємо ще раз цю команду:

```
db.test.insertOne( { name: "test_2", time: new Date() }, {  
writeConcern: { w: 3, wtimeout: 0 } } );
```

Бачимо, що програма зависла, та чекає запуску ноди.

```
rs0 [direct: primary] test> db.test.insertOne( { name: "test_2", time: new Date() }, { writeConcern: { w: 3, wtimeout: 0 } } );
```

Запустимо ноду знову:

| | | | |
|---|--|---|--|
|  | mongo2 mongo Running 27018:27017 ↗ |    | kEx :"a 202 "I" on is" 202 "I" on is" 202 "I" n i s": 202 "I" on is" 202 |
|  | mongo3 mongo Running 27019:27017 ↗ |    | |
|  | mongo1 mongo Running 27017:27017 ↗ |    | |

Бачимо, що все працює

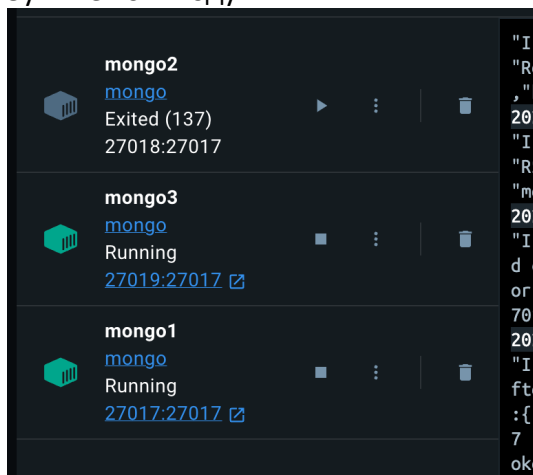
```
rs0 [direct: primary] test> db.test.insertOne( { name: "test_2", time: new Date() }, { writeConcern: { w: 3, wtimeout: 0 } } );  
{  
  acknowledged: true,  
  insertedId: ObjectId('675b6e85978f714ac1e0bd5b')  
}  
rs0 [direct: primary] test>
```

3. Аналогічно попередньому пункту, але задати скінченний таймаут та дочекатись його закінчення. Перевірити чи данні записались і чи доступні на читання з рівнем *readConcern: "majority"*

```
db.test.insert( { name: "test_timeout", time: new Date() }, {
writeConcern: { w: 3, wtimeout: 5000 } })
```

```
rs0 [direct: primary] test> db.test.insert( { name: "test_timeout", time: new Date() }, { writeConcern: { w: 3, wtimeout: 5000 } })
DeprecationWarning: Collection.insert() is deprecated. Use insertOne, insertMany, or bulkWrite.
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('675b717a978f714ac1e0bd5c') }
}
```

Зупинемо 2 ноду:



```
rs0 [direct: primary] test> db.test.insert( { name: "test_timeout_4", time: new Date() }, { writeConcern: { w: 3, wtimeout: 10000 } })
Uncaught:
MongoBulkWriteError: waiting for replication timed out
Result: BulkWriteResult {
  insertedCount: 1,
  matchedCount: 0,
  modifiedCount: 0,
  deletedCount: 0,
  upsertedCount: 0,
  upsertedIds: {},
  insertedIds: { '0': ObjectId('675b72c0b3329d54fac9b6ea') }
}
Write Errors: []
rs0 [direct: primary] test>

rs0 [direct: primary] test> db.test.find();
[
  {
    _id: ObjectId('675b6cbcd0d6dc01f61d6f2'),
    name: 'Test_1',
    time: ISODate('2024-12-12T23:07:34.829Z')
  },
  {
    _id: ObjectId('675b6cf4cd0d6dc01f61d6f3'),
    name: 'Test_1',
    time: ISODate('2024-12-12T23:08:36.701Z')
  },
  {
    _id: ObjectId('675b6e85978f714ac1e0bd5b'),
    name: 'Test_2',
    time: ISODate('2024-12-12T23:15:17.917Z')
  },
  {
    _id: ObjectId('675b717a978f714ac1e0bd5c'),
    name: 'test_timeout',
    time: ISODate('2024-12-12T23:27:54.058Z')
  },
  {
    _id: ObjectId('675b7292b3329d54fac9b6e8'),
    name: 'test_timeout_2',
    time: ISODate('2024-12-12T23:32:34.951Z')
  },
  {
    _id: ObjectId('675b72a8b3329d54fac9b6e9'),
    name: 'test_timeout_3',
    time: ISODate('2024-12-12T23:32:56.174Z')
  },
  {
    _id: ObjectId('675b72c0b3329d54fac9b6ea'),
    name: 'test_timeout_4',
    time: ISODate('2024-12-12T23:33:20.957Z')
  }
]
```













Запис зберігся:

```
{
  _id: ObjectId('675b72c0b3329d54fac9b6ea'),
  name: 'test_timeout_4',
  time: ISODate('2024-12-12T23:33:20.957Z')
}
```

4. Продемонстрував перевибори primary node відключивши поточний primary (Replica Set Elections) - <http://docs.mongodb.org/manual/core/replica-set-elections/>
 - і що після відновлення роботи старої primary на неї реплікуються нові дані, які з'явилися під час її простою

```
{
  _id: 0,
  name: 'mongo1:27017',
  health: 1,
  state: 1,
  stateStr: 'PRIMARY',
  uptime: 1206,
  optime: { ts: Timestamp({
```

Відключаємо поточний primary:

| | | | | |
|---|--|---|---|---|
|  | mongo2 mongo Running 27018:27017 |  |  |  |
|  | mongo3 mongo Running 27019:27017 |  |  |  |
|  | mongo1 mongo Exited (137) 27017:27017 |  |  |  |

```
[rs0 [direct: primary] test> rs.status();
{
```

Наша попередня основна нода тепер недоступна:

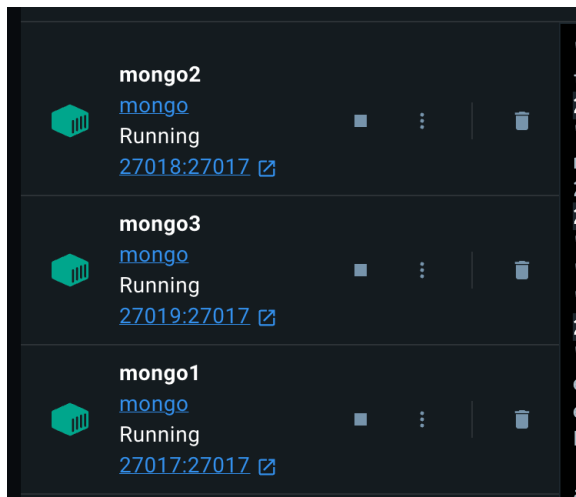
```
{
  _id: 0,
  name: 'mongo1:27017',
  health: 0,
  state: 8,
  stateStr: '(not reachable/healthy)',
  uptime: 0,
  optime: { ts: Timestamp({ t: 0, i: 0 }), t: 10
```

Тепер mongo2 – primary:

```
{
  _id: 1,
  name: 'mongo2:27017',
  health: 1,
  state: 1,
  stateStr: 'PRIMARY',
  uptime: 133,
```

```
rs0 [direct: primary] test> db.test.insertOne({ name: "task4", time: new Date() });
{
  acknowledged: true,
  insertedId: ObjectId('675b75431d13481e1877999b')
}
rs0 [direct: primary] test> █
```

Запустимо ноду:



Бачимо, що все на місці:

```
time: ISODate('2024-12-12T23:35:11.599Z')
},
{
  _id: ObjectId('675b75431d13481e1877999b'),
  name: 'task4',
  time: ISODate('2024-12-12T23:44:03.691Z')
}
]
```

```

}
rs0 [direct: primary] test> db.test.find().readConcern("majority");
[
  {
    _id: ObjectId('675b6cb6cd0d6dc01f61d6f2'),
    name: 'Test 1',
    time: ISODate('2024-12-12T23:07:34.829Z')
  },
  {
    _id: ObjectId('675b6cf4cd0d6dc01f61d6f3'),
    name: 'test_1',
    time: ISODate('2024-12-12T23:08:36.701Z')
  },
  {
    _id: ObjectId('675b6e85978f714ac1e0bd5b'),
    name: 'test_2',
    time: ISODate('2024-12-12T23:15:17.917Z')
  },
  {
    _id: ObjectId('675b717a978f714ac1e0bd5c'),
    name: 'test_timeout',
    time: ISODate('2024-12-12T23:27:54.058Z')
  },
  {
    _id: ObjectId('675b7292b3329d54fac9b6e8'),
    name: 'test_timeout_2',
    time: ISODate('2024-12-12T23:32:34.951Z')
  },
  {
    _id: ObjectId('675b72a8b3329d54fac9b6e9'),
    name: 'test_timeout_3',
    time: ISODate('2024-12-12T23:32:56.174Z')
  },
  {
    _id: ObjectId('675b72c0b3329d54fac9b6ea'),
    name: 'test_timeout_4',
    time: ISODate('2024-12-12T23:33:20.957Z')
  },
  {
    _id: ObjectId('675b732fb3329d54fac9b6eb'),
    name: 'test_timeout_5',
    time: ISODate('2024-12-12T23:35:11.599Z')
  },
  {
    _id: ObjectId('675b75431d13481e1877999b'),
    name: 'task4',
    time: ISODate('2024-12-12T23:44:03.691Z')
  }
]
rs0 [direct: primary] test>

```

II Аналіз продуктивності та перевірка цілісності

Аналогічно попереднім завданням, необхідно буде створити колекцію (таблицю) з каунтером лайків. Далі з 10 окремих клієнтів одночасно запустити інкрементацію каунтеру лайків по 10_000 на кожного клієнта з різними опціями взаємодії з MongoDB.

Для того, щоб не було lost updates, для оновлення каунтера необхідно використовувати функцію [findOneAndUpdate\(\)](#)

```
rs0 [direct: primary] performance_test> db.likes_counter.insertOne({ _id: 1, likes: 0 });
{ acknowledged: true, insertedId: 1 }
rs0 [direct: primary] performance_test> db.likes_counter.find()
[ { _id: 1, likes: 0 } ]
rs0 [direct: primary] performance_test>
```

1. Вказавши у параметрах *findOneAndUpdate writeConcern = 1* (це буде означати, що запис іде тільки на Primary ноду і не чекає відповіді від Secondary), запустіть 10 клієнтів з інкрементом по 10_000 на кожному з них. Виміряйте час виконання та перевірте чи кінцеве значення буде дорівнювати очікуваному - 100K
2. Вказавши у параметрах *findOneAndUpdate writeConcern = majority* (це буде означати, що Primary чекає поки значення запишеться на більшість нод), запустіть 10 клієнтів з інкрементом по 10_000 на кожному з них. Виміряйте час виконання та перевірте чи кінцеве значення буде дорівнювати очікуваному - 100K

```
def increment_likes(write_concern, increments):

    client = MongoClient(MONGO_URL)
    db = client["performance_test"]
    collection = db.get_collection("likes_counter", write_concern=write_concern)

    for _ in range(increments):
        collection.find_one_and_update(
            {"_id": 1},
            {"$inc": {"likes": 1}}
        )
    client.close()

def run_test(write_concern, increments, clients):

    threads = []
    for _ in range(clients):
        thread = Thread(target=increment_likes, args=(write_concern, increments))
        threads.append(thread)
        thread.start()

    for thread in threads:
        thread.join()

    print("Test completed.")
```



```
● dmitriy@Dmitrys-MacBook-Air pvs % /usr/local/bin/python3 /Users/dmitriy/  
Running test with writeConcern = 1...  
Test completed.  
Time with writeConcern = 1: 13.10546588897705  
Final likes count after writeConcern = 1: 100000  
Likes counter reset to 0.  
Running test with writeConcern = majority...  
Test completed.  
Time with writeConcern = majority: 109.90764498710632  
Final likes count after writeConcern = majority: 100000  
○ dmitriy@Dmitrys-MacBook-Air pvs % █
```

```
}  
[rs0 [direct: secondary] performance_test> db.likes_counter.find();  
[ { _id: 1, likes: 100000 } ]  
rs0 [direct: secondary] performance_test> █
```