ARCHITETTURE DATI

Progetto MongoDB

Relazione di: Riccardo Andena 859643
Andrea Messa 856435
Andrea Tirico 851958

OBIETTIVI E TECNOLOGIE UTILIZZATE

L'obiettivo del progetto nel verificare la documentazione di MongoDB per quanto riguarda le transazioni e la verifica della gestione dei dati in caso di spegnimento di un nodo.

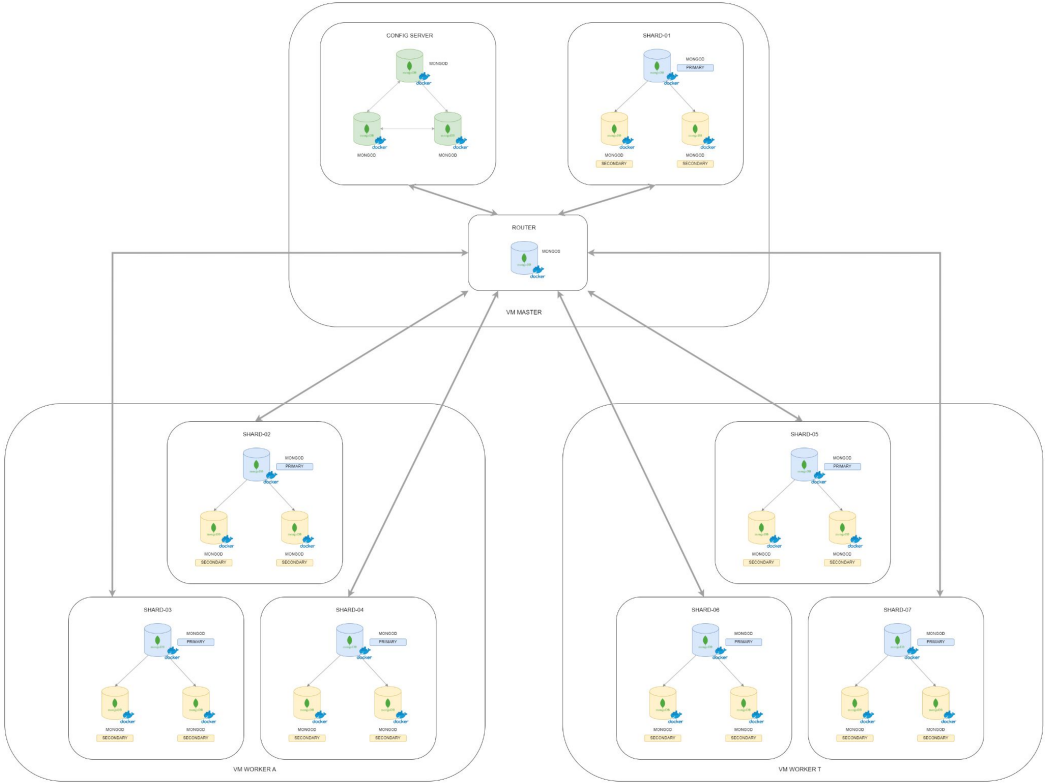
Per svolgere il progetto sono state utilizzate le seguenti tecnologie:

- Docker
- MongoDB
- Mongo Compass
- Visual Studio Code
- Azure Virtual Machine
- Python

ARCHITETTURA

Per il progetto è stata utilizzata un’architettura di sharding ovvero un’architettura di database distribuita che consente di suddividere i dati tra più server o nodi, nei quali ci possono essere uno o più shard.

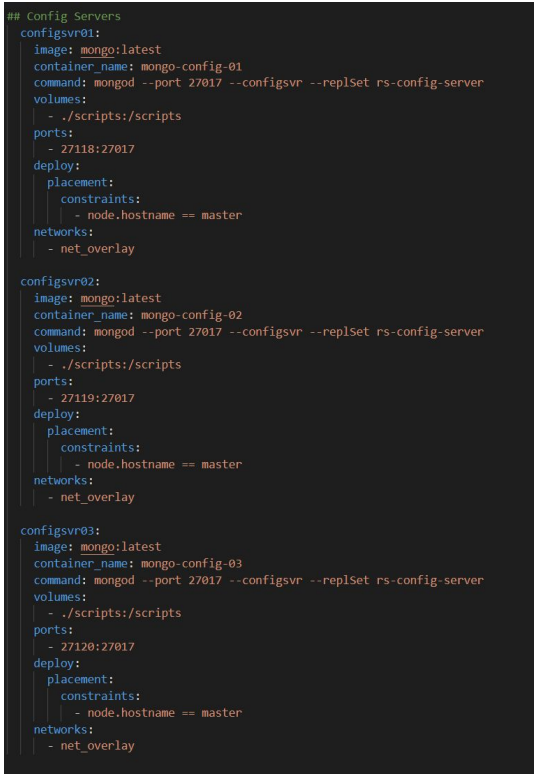
Nel nostro caso sono stati distribuiti su 3 Virtual Machine: 7 shard, 1 router e 1 config server.



FILE YAML

Nel file YAML si possono avere le seguenti etichette:

- Image
- Container-Name
- Command
- Volumes
- Ports
- Deploy
  - Placement
  - Constraints
- Networks

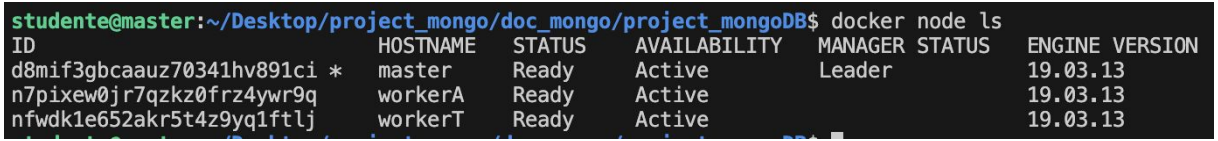


```yaml
## Config Servers
  configsvr01:
    image: mongo:latest
    container_name: mongo-config-01
    command: mongod --port 27017 --configsvr --replSet rs-config-server
    volumes:
      - ./scripts:/scripts
    ports:
      - 27118:27017
    deploy:
      placement:
        constraints:
          - node.hostname == master
    networks:
      - net_overlay

  configsvr02:
    image: mongo:latest
    container_name: mongo-config-02
    command: mongod --port 27017 --configsvr --replSet rs-config-server
    volumes:
      - ./scripts:/scripts
    ports:
      - 27119:27017
    deploy:
      placement:
        constraints:
          - node.hostname == master
    networks:
      - net_overlay

  configsvr03:
    image: mongo:latest
    container_name: mongo-config-03
    command: mongod --port 27017 --configsvr --replSet rs-config-server
    volumes:
      - ./scripts:/scripts
    ports:
      - 27120:27017
    deploy:
      placement:
        constraints:
          - node.hostname == master
    networks:
      - net_overlay
```

CREAZIONE NETWORK

Per la creazione network ci siamo affidati a una rete overlay. Quest'ultima è una rete virtuale che consente alle diverse istanze distribuite di Docker di comunicare tra loro. Vi sono diversi passaggi da eseguire per creare una network overlay in Docker:

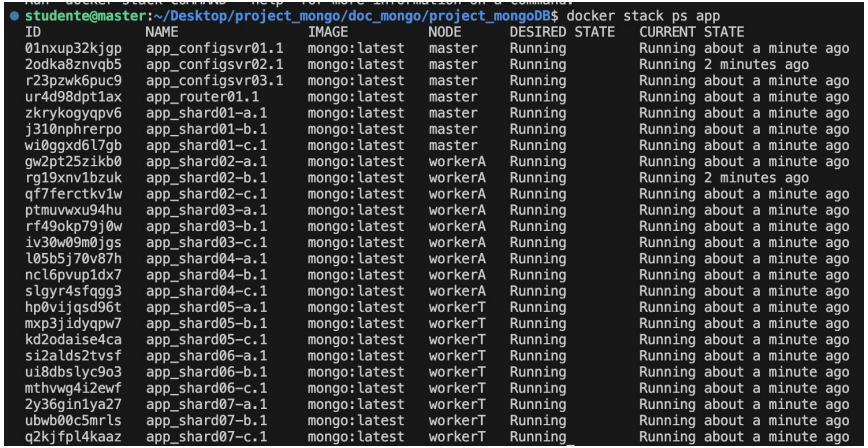
1. Viene configurata un'infrastruttura di rete sottostante che supporti l'overlay, per fare ciò verrà utilizzato Docker Swarm che è un servizio di orchestrazione dei container nativo di docker, il quale permette di avere un'architettura scalabile per gestire i container distribuiti su più host;



2. In seguito viene creata la rete overlay con il comando: *docker network create --driver overlay my-overlay-network*

CREAZIONE NETWORK

3. Viene letto il file YAML per determinare i servizi, le configurazioni e le dipendenze specificate e distribuire i nodi sugli host che appartengono alla rete overlay.



```
studente@master:~/Desktop/project_mongo/doc_mongo/project_mongoDB$ docker stack ps app
ID             NAME                IMAGE          NODE      DESIRED STATE   CURRENT STATE
01nxup32kjgp   app_configsvr01.1   mongo:latest   master    Running         Running about a minute ago
2odka8znvqb5   app_configsvr02.1   mongo:latest   master    Running         Running 2 minutes ago
r23pzwk6puc9   app_configsvr03.1   mongo:latest   master    Running         Running about a minute ago
ur4d98dpt1ax   app_router01.1      mongo:latest   master    Running         Running about a minute ago
zkrykogyqpv6   app_shard01-a.1     mongo:latest   master    Running         Running about a minute ago
j310nphrerpo   app_shard01-b.1     mongo:latest   master    Running         Running about a minute ago
wi0ggxd6l7gb   app_shard01-c.1     mongo:latest   master    Running         Running about a minute ago
gw2pt25zikb0   app_shard02-a.1     mongo:latest   workerA   Running         Running about a minute ago
rg19xnv1bzuk   app_shard02-b.1     mongo:latest   workerA   Running         Running 2 minutes ago
qf7ferctkv1w   app_shard02-c.1     mongo:latest   workerA   Running         Running about a minute ago
ptmuvwxu94hu   app_shard03-a.1     mongo:latest   workerA   Running         Running about a minute ago
rf49okp79j0w   app_shard03-b.1     mongo:latest   workerA   Running         Running about a minute ago
iv30w09m0jgs   app_shard03-c.1     mongo:latest   workerA   Running         Running about a minute ago
l05b5j70v87h   app_shard04-a.1     mongo:latest   workerA   Running         Running about a minute ago
ncl6pvup1dx7   app_shard04-b.1     mongo:latest   workerA   Running         Running about a minute ago
slgyr4sfqgg3   app_shard04-c.1     mongo:latest   workerA   Running         Running about a minute ago
hp0vijqsd96t   app_shard05-a.1     mongo:latest   workerT   Running         Running about a minute ago
mxp3jidyqpw7   app_shard05-b.1     mongo:latest   workerT   Running         Running about a minute ago
kd2odaise4ca   app_shard05-c.1     mongo:latest   workerT   Running         Running about a minute ago
si2alds2tvsf   app_shard06-a.1     mongo:latest   workerT   Running         Running about a minute ago
ui8dbslyc9o3   app_shard06-b.1     mongo:latest   workerT   Running         Running about a minute ago
mthvwg4i2ewf   app_shard06-c.1     mongo:latest   workerT   Running         Running about a minute ago
2y36gin1ya27   app_shard07-a.1     mongo:latest   workerT   Running         Running about a minute ago
ubwb00c5mrls   app_shard07-b.1     mongo:latest   workerT   Running         Running about a minute ago
q2kjfpl4kaaz   app_shard07-c.1     mongo:latest   workerT   Running         Running about a minute ago
```

DISTRIBUZIONE DEI DATI

La procedura utilizzata per distribuire in modo omogeneo il database su ogni shard che compone l'architettura è la seguente:

1. Il primo passo che si effettua è abilitare lo sharding del database;
2. Come secondo step verrà abilitata la collezione che fa parte del database e indicata la shard key;
3. Il passo successivo è quello di caricare il database dentro l'infrastruttura effettuando la mongoimport direttamente dal terminale del router.

```
import json
import random
```

```
example = {
    "id": 1,
    "name": "Pippo",
    "surname": "Pluto",
    "date_of_birth": "18/03/2000",
    "bank": "UniCredit",
    "balance": "10000"
}

list_name = ['Sofia', 'Matteo', 'Giuseppe', 'Isabella', 'Federico', 'Giulia', 'Carlo', 'Veronica', '
list_surname = ["Rossi", "Bianchi", "Russo", "Esposito", "Ricci", "Ferrari", "Rizzo", "Romano", "Con
italian_banks = [
    "Intesa Sanpaolo",
    "UniCredit",
    "Banco BPM",
    "Monte dei Paschi di Siena",
    "UBI Banca",
    "Banca Nazionale del Lavoro",
    "Banca Generali",
    "Mediobanca",
    "Cassa Depositi e Prestiti",
    "Banca Popolare di Sondrio"
]
```

```
banks = []
type(banks)
```

```
list
```

```
for i in range(1, 1001):
    item = example.copy()

    rand_name = random.randint(0, 99)
    rand_year = random.randint(1960, 2004)
    rand_month = random.randint(1, 12)
    rand_day = random.randint(1, 31)
    rand_banca = random.randint(0,9)
    rand_balance = random.randint(1000, 1000000)

    item["id"] = i
    item["name"] = list_name[rand_name]
    item["surname"] = list_surname[rand_name]
    item["date_of_birth"] = f"{rand_day}/{rand_month}/{rand_year}"
    item["bank"] = italian_banks[rand_banca]
    item["balance"] = rand_balance

    banks.append(item)
```

```
with open("banks.json", "w") as outfile:
    json.dump(banks, outfile)
```

```
example
```

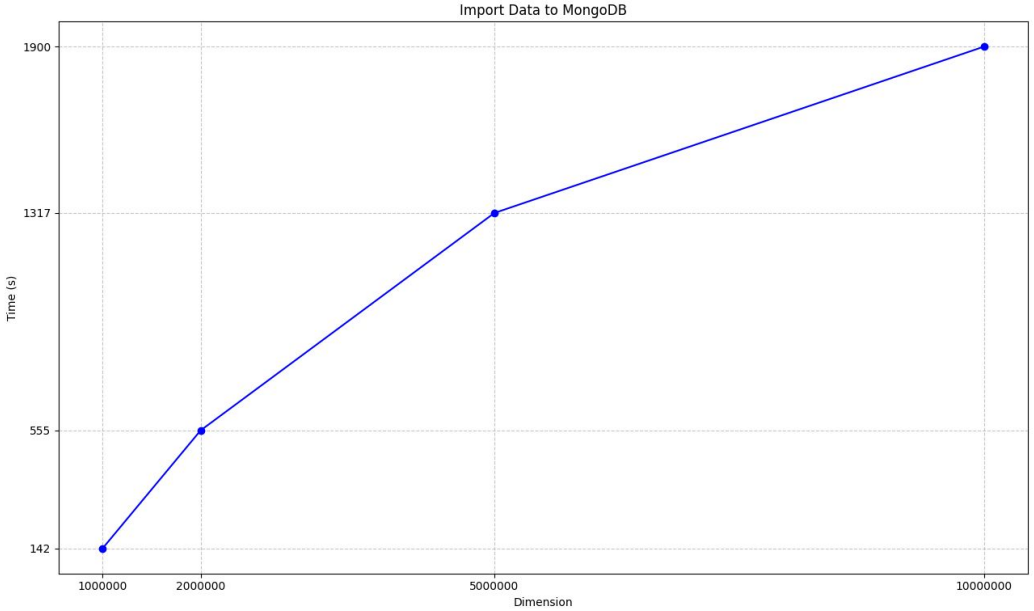
```
{'id': 1,
 'name': 'Pippo',
 'surname': 'Pluto',
 'date_of_birth': '18/03/2000',
 'bank': 'UniCredit',
 'balance': '10000'}
```

DISTRIBUZIONE DEI DATI

```
[direct: mongos] test> sh.enableSharding("db_users")
{
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1685530354, i: 466 }),
    signature: {
      hash: Binary(Buffer.from("0000000000000000000000000000000000000000", "hex"), 0),
      keyId: Long("0")
    }
  },
  operationTime: Timestamp({ t: 1685530354, i: 464 })
}
[direct: mongos] test> sh.shardCollection("db_users.users", {"id":"hashed"})
{
  collectionsharded: 'db_users.users',
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1685530434, i: 6 }),
    signature: {
      hash: Binary(Buffer.from("0000000000000000000000000000000000000000", "hex"), 0),
      keyId: Long("0")
    }
  },
  operationTime: Timestamp({ t: 1685530434, i: 2 })
}
[direct: mongos] test> show dbs
admin     80.00 KiB
config     3.44 MiB
db_users  84.00 KiB
```

```
Totals
{
  data: '131.73MiB',
  docs: 1000000,
  chunks: 14,
  'Shard rs-shard-02': [
    '14.25 % data',
    '14.25 % docs in cluster',
    '138B avg obj size on shard'
  ],
  'Shard rs-shard-03': [
    '14.33 % data',
    '14.33 % docs in cluster',
    '138B avg obj size on shard'
  ],
  'Shard rs-shard-05': [
    '14.28 % data',
    '14.28 % docs in cluster',
    '138B avg obj size on shard'
  ],
  'Shard rs-shard-06': [
    '14.32 % data',
    '14.32 % docs in cluster',
    '138B avg obj size on shard'
  ],
  'Shard rs-shard-04': [
    '14.19 % data',
    '14.19 % docs in cluster',
    '138B avg obj size on shard'
  ],
  'Shard rs-shard-01': [
    '14.34 % data',
    '14.34 % docs in cluster',
    '138B avg obj size on shard'
  ],
  'Shard rs-shard-07': [
    '14.26 % data',
    '14.26 % docs in cluster',
    '138B avg obj size on shard'
  ]
}
```

TEST - CARICAMENTO DEI DATI



TEST - TRANSAZIONI

Dirty read

T1	T2
START	
	START
Read → 1000	
Write → -100	
	Read → 900
COMMIT	
	Write → -100
	COMMIT

Phantom read

T1	T2
START	
	START
Read →9000 righe dove balance > 900.000	
	Read → righe dove balance > 900.000
	Write → balance di uno = 300
	COMMIT
Read → 8999 righe dove balance > 900.000	
COMMIT	

Non repeatable read

T1	T2
START	
	START
Read → 1000	
	Read → 1000
	Write → -100
	COMMIT
READ → 900	
COMMIT	

TEST - TRANSAZIONI

Dirty read

Non repeatable read

Phantom read

Livelli di ReadConcern:

- majority
- local
- snapshot
- available

Livelli di WriteConcern:

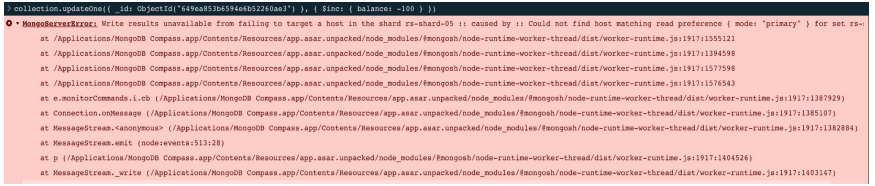
- majority
- 1

TEST - SPEGNIMENTO DEI NODI

Comandi docker utilizzati:

- docker node update –availability drain id_host
- docker node update –availability active id_host
- docker node update –availability pause id_host

Update dopo spegnimento di un nodo



Read dopo spegnimento di un nodo

TEST - SPEGNIMENTO DEI NODI

Comandi docker utilizzati:

- docker node update –availability drain id_host
- docker node update –availability active id_host
- docker node update –availability pause id_host

Spegnimento due nodi con RC: majority e WC: majority

```
[direct: mongos] test> collection.updateOne({ _id: ObjectId("646bd4c8cc2d06e28d68c6d3") }, { $inc: { balance: -100 } })
MongoServerError: Write results unavailable from failing to target a host in the shard rs-shard-02 :: caused by :: Could not fi
nd host matching read preference { mode: "primary" } for set rs-shard-02
```

Spegnimento due nodi con RC: majority e WC: 1

```
[direct: mongos] test> collection.updateOne({ _id: ObjectId("646bd4c8cc2d06e28d68c6d3") }, { $inc: { balance: -100 } })
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

TEST - SCRIPT

```
Lettura in T1:  {'_id': ObjectId('64a958a37db95684b99cf37f'), 'id': 299, 'name': 'Gaia', 'surname': 'Pellegrini', 'date_of_birt
h': '1/11/1985', 'bank': 'UBI Banca', 'balance': 700}
Lettura in T2:  {'_id': ObjectId('64a958a37db95684b99cf37f'), 'id': 299, 'name': 'Gaia', 'surname': 'Pellegrini', 'date_of_birt
h': '1/11/1985', 'bank': 'UBI Banca', 'balance': 700}
Transazione 1 committata
Transazione 2 committata
Lettura finale  {'_id': ObjectId('64a958a37db95684b99cf37f'), 'id': 299, 'name': 'Gaia', 'surname': 'Pellegrini', 'date_of_birt
h': '1/11/1985', 'bank': 'UBI Banca', 'balance': 900}
```

```
try:
    # Avvia la transazione per la sessione 1
    with session2.start_transaction(read_concern=read_concern, write_concern=write_concern) as s2,
    session1.start_transaction(read_concern=read_concern, write_concern=write_concern) as s1:
        # Ottieni la collezione nella sessione 1

        collection1 = session1.client['db_bank']['bank']
        collection2 = session2.client['db_bank']['bank']

        result1 = collection1.find_one({ 'id': 299  })
        print("Lettura in T1: ", result1)

        # Esegui l'operazione di modifica nella sessione 1
        collection1.update_one(
            {'_id': result1['_id']},
            {'$inc': {'balance': 100}}, session = session1
        )

         # Esegui l'operazione di lettura nella sessione 2
        result2 = collection2.find_one({ 'id': 299  }, session = session2)
        print("Lettura in T2: ",result2)


        session1.commit_transaction()
        print("Transazione 1 committata")

        # Esegui l'operazione di modifica nella sessione 2
        collection2.update_one(
            {'_id': result2['_id']},
            {'$inc': {'balance': 100}}, session = session2
        )

        # Commit della transazione per la sessione 2
        session2.commit_transaction()
        print("Transazione 2 committata")

    result2 = collection2.find_one({ 'id': 299  }, session = session2)
    print(result2)

except Exception as e:
    # Esegui il rollback delle transazioni in caso di errore
   # session1.abort_transaction()
   # session2.abort_transaction()
    collection2.update_one(
            {'_id': result2['_id']},
            {'$inc': {'balance': 100}}, session = session2
        )
    print("Transazione 2 committata")
    result2 = collection2.find_one({ 'id': 299  }, session = session2)
    print("Lettura finale ", result2)

finally:
    # Chiudi le sessioni
    session1.end_session()
    session2.end_session()
```

TEST - SCRIPT

```
#example update
execute_func(client, update_users_transaction, read_c = read_concern, write_c = write_concern, read_p = read_preference)
```

```
it's ok

[{'_id': ObjectId('64ae718457a8c8cbd2746489'),
  'id': 1000007,
  'name': 'Enrico',
  'surname': 'Marini',
  'date_of_birth': '28/10/1962',
  'bank': 'UniCredit',
  'balance': 926677,
  'test': 1,
  'date': datetime.datetime(2023, 7, 12, 11, 25, 24, 158000),
  'trans': [{'id_user': ObjectId('64ae718457a8c8cbd274648d'),
    'type': -1,
    'value': 1000,
    'date': '07/12/23'}]},
 {'_id': ObjectId('64ae718457a8c8cbd274648d'),
  'id': 1000011,
  'name': 'Enrico',
  'surname': 'Donati',
  'date_of_birth': '8/2/1988',
  'bank': 'Banca Nazionale del Lavoro',
  'balance': 924677,
  'test': 1,
  'date': datetime.datetime(2023, 7, 12, 11, 25, 24, 158000),
  'trans': [{'id_user': ObjectId('64ae718457a8c8cbd2746489'),
    'type': 1,
    'value': 1000,
    'date': '07/12/23'}]}]
```

```
found = execute_func(client, find_bank_user, read_c = read_concern, write_c = write_concern, read_p = read_preference)
print(len(found))
```

```
it's ok
100266
```

```
found = execute_func(client, find_date_transaction, read_c = read_concern, write_c = write_concern, read_p = read_preference)
print(found)
```

```
it's ok
[{'_id': ObjectId('64ae718457a8c8cbd2746489'), 'id': 1000007, 'name': 'Enrico', 'surname': 'Marini', 'date_of_birth': '28/10/19
62', 'bank': 'UniCredit', 'balance': 926677, 'test': 1, 'date': datetime.datetime(2023, 7, 12, 11, 25, 24, 158000), 'trans':
[{'id_user': ObjectId('64ae718457a8c8cbd274648d'), 'type': -1, 'value': 1000, 'date': '07/12/23'}]}, {'_id': ObjectId('64ae7184
57a8c8cbd274648d'), 'id': 1000011, 'name': 'Enrico', 'surname': 'Donati', 'date_of_birth': '8/2/1988', 'bank': 'Banca Nazionale
del Lavoro', 'balance': 924677, 'test': 1, 'date': datetime.datetime(2023, 7, 12, 11, 25, 24, 158000), 'trans': [{'id_user': Ob
jectId('64ae718457a8c8cbd2746489'), 'type': 1, 'value': 1000, 'date': '07/12/23'}]}]
```

```
execute_func(client, remove_bank_many, read_c = read_concern, write_c = write_concern, read_p = read_preference)
```

```
it's ok

[]
```

CONCLUSIONI

Si può dedurre:

- Più dati si inseriscono, più i tempi si riducono in maniera proporzionale
- Le transazioni sono molto stringenti
- Lo spegnimento di nodi restituisce risultati differenti in base a come si impostano le Concern