Rendu 3 - BDD

Max

Janvier 2026

1 Introduction

Dans cette troisième partie, nous allons continuer avec la découverte de MongoDB en mettant en place des replicas de notre base. Nous allons ensuite faire une série de tests afin de vérifier la robustesse de notre base, avant de commencer la préparation de Django.

2 Création de Réplicas

Afin de créer les replicas, il faut commencer par démarrer 3 serveurs MongoDB à l'aide des commandes que l'on effectue dans des terminaux différents :

```
mongod --replSet rs0 --port 27017 --dbpath ./data/mongo/
    db-1 --bind_ip localhost

mongod --replSet rs0 --port 27018 --dbpath ./data/mongo/
    db-2 --bind_ip localhost

mongod --replSet rs0 --port 27019 --dbpath ./data/mongo/
    db-3 --bind_ip localhost
```

Ensuite, on peut se connecter à l'un d'entre eux depuis un quatrième terminal à l'aide de cette commande

```
mongosh --port 27017
```

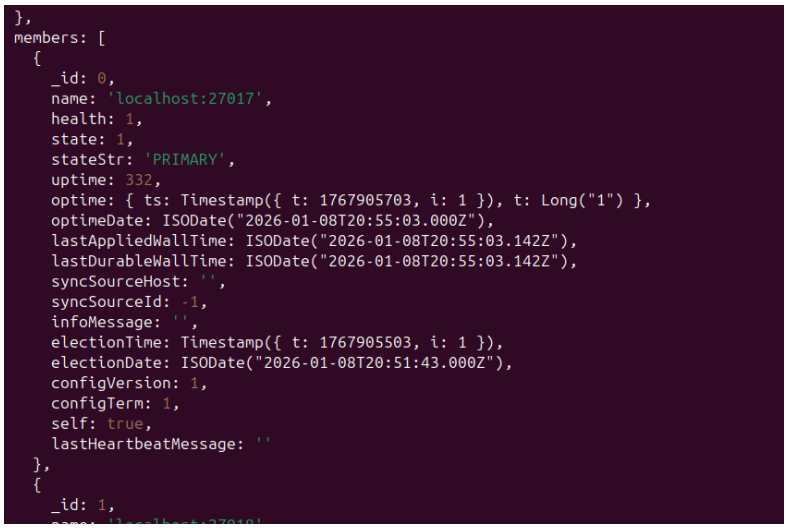
Une fois connecté, nous pouvons initialiser le replica set à l'aide de cette commande :

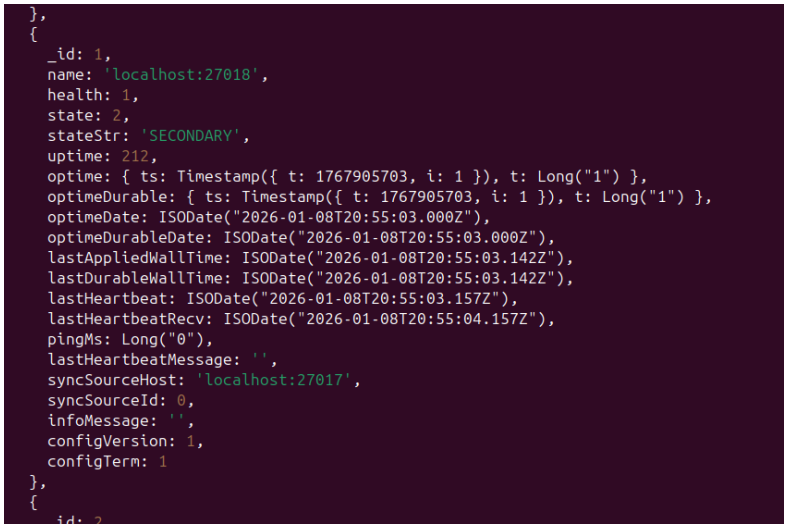
```
rs.initiate({
    _id: "rs0",
    members: [
        { _id: 0, host: "localhost:27017" },
        { _id: 1, host: "localhost:27018" },
        { _id: 2, host: "localhost:27019" }
    ]
})
```

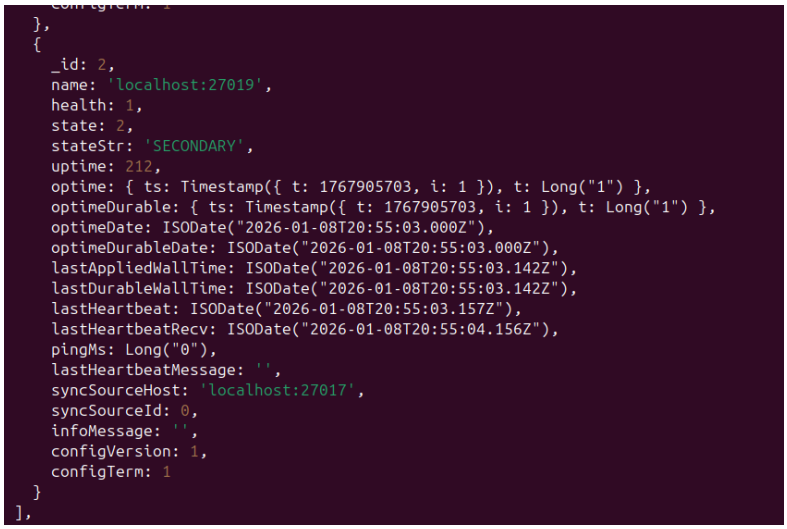
Afin de vérifier la connexion, on peut utiliser la commande

```
rs.status()
```

pour afficher le statut de chacun des serveurs mongoDB.



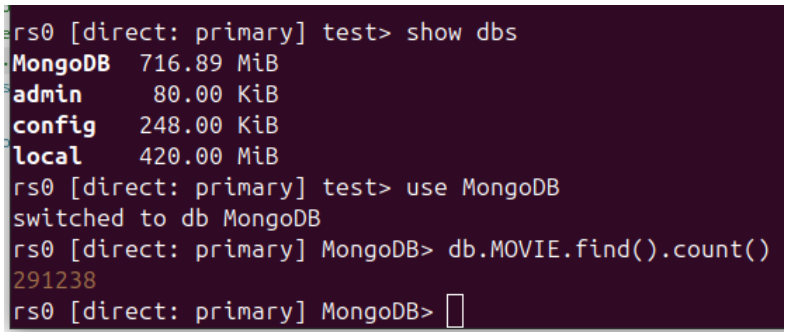




Nous pouvons voir sur la première image que le port 27017 contient bien la base principale et les deux autres ports contiennent les secondaires

Après l'insertion des données dans la base principale, on peut vérifier leur insertion à l'aide des commandes :

```
mongosh --port 27017

show dbs
use MongoDB
db.MOVIE.find().count()
```



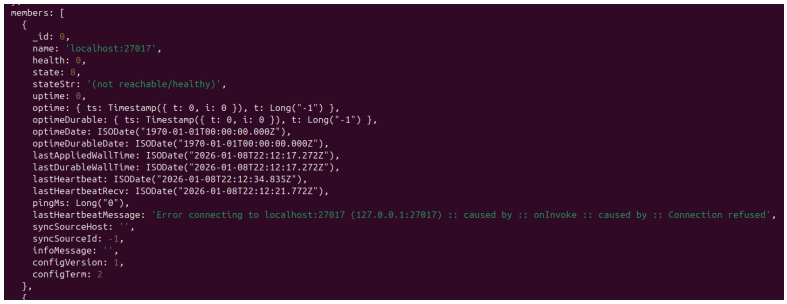
On peut y voir que les données ont bien été importées.

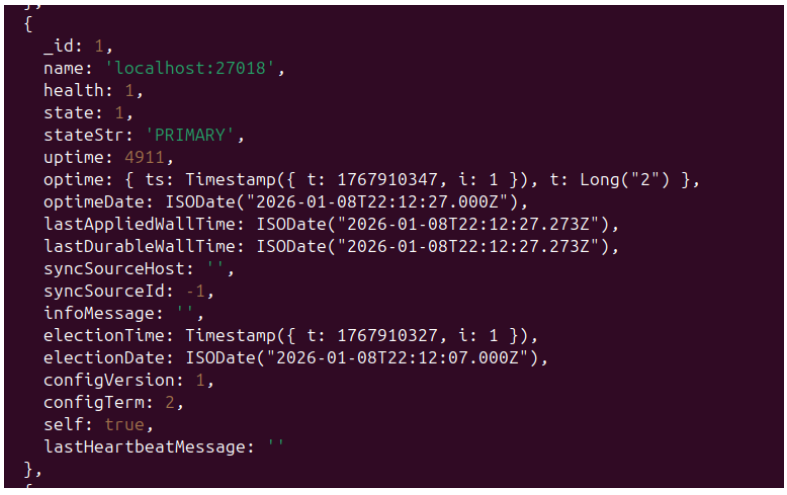
Il est possible de voir que les replicas ont bien été créées en faisant la même chose pour les ports 27018 et 27019.

```
mongosh --port 27018

db.getMongo().setReadPref("secondary")
show dbs
use MongoDB
db.MOVIE.find().count()
```

3 Test de tolérance aux pannes

On va maintenant simuler une panne, pour cela on coupe la connexion au Primary en faisant un ctrl+c dans le terminal 1.
Au bout d'une dizaine de secondes, on peut alors refaire rs.status() dans le terminal 4 afin de voir les changements :





Le replica sur le port 27018 est maintenant devenu le Primary tandis que celui sur 27017 n'est plus accessible.

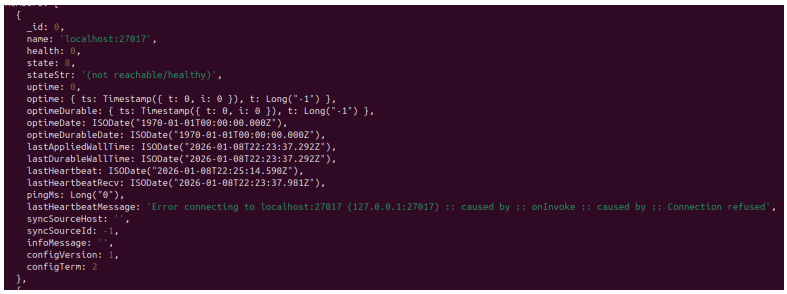
Quant aux données, elles sont toujours accessibles :

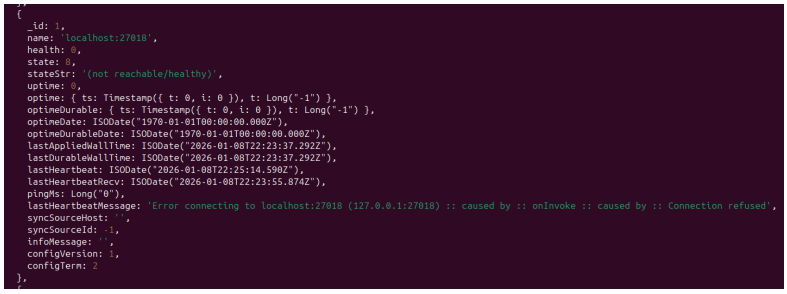
```
rs0 [direct: primary] MongoDB> db.MOVIE.find().count()
291238
```

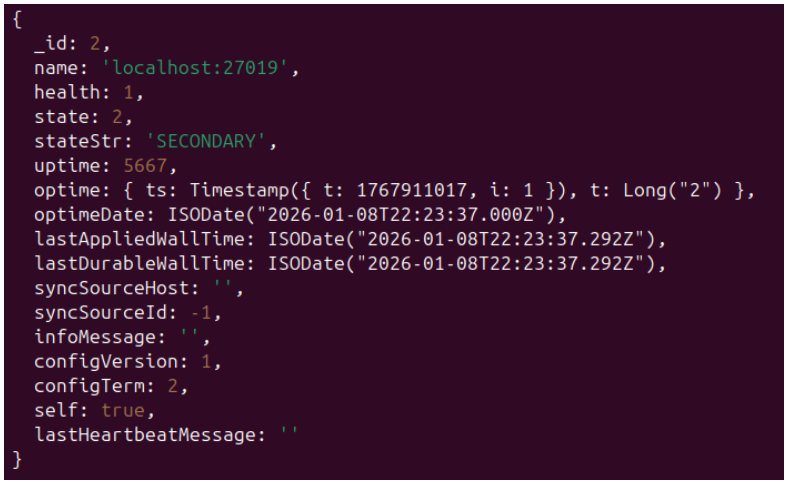
Après avoir relancé la connexion, nous pouvons voir que le port 27017 est maintenant secondaire.

```
members: [
  {
    _id: 0,
    name: 'localhost:27017',
    health: 1,
    state: 2,
    stateStr: 'SECONDARY',
    uptime: 17,
    optime: { ts: Timestamp({ t: 1767910843, i: 1 }), t: Long("2") },
    optimeDurable: { ts: Timestamp({ t: 1767910843, i: 1 }), t: Long("2") },
    optimeDate: ISODate("2026-01-08T22:20:43.000Z"),
    optimeDurableDate: ISODate("2026-01-08T22:20:43.000Z"),
    lastAppliedWallTime: ISODate("2026-01-08T22:20:57.287Z"),
    lastDurableWallTime: ISODate("2026-01-08T22:20:57.287Z"),
    lastHeartbeat: ISODate("2026-01-08T22:20:57.015Z"),
    lastHeartbeatRecv: ISODate("2026-01-08T22:20:57.967Z"),
    pingMs: Long("0"),
    lastHeartbeatMessage: '',
    syncSourceHost: 'localhost:27019',
    syncSourceId: 2,
    infoMessage: '',
    configVersion: 1,
    configTerm: 2
  },
```

Pour finir avec les tests, nous allons simuler 2 pannes en simultanée. Coupons les connexions aux ports 27017 et 27018. Nous avons maintenant :


{
_id: 0,
name: 'localhost:27017',
health: 0,
state: 8,
stateStr: '(not reachable/healthy)',
uptime: 0,
optime: { ts: Timestamp({ t: 0, i: 0 }), t: Long("-1") },
optimeDurable: { ts: Timestamp({ t: 0, i: 0 }), t: Long("-1") },
optimeDate: ISODate("1970-01-01T00:00:00.000Z"),
optimeDurableDate: ISODate("1970-01-01T00:00:00.000Z"),
lastAppliedWallTime: ISODate("2026-01-08T22:23:37.292Z"),
lastDurableWallTime: ISODate("2026-01-08T22:23:37.292Z"),
lastHeartbeat: ISODate("2026-01-08T22:25:14.590Z"),
lastHeartbeatRecv: ISODate("2026-01-08T22:23:37.981Z"),
pingMs: Long("0"),
lastHeartbeatMessage: 'Error connecting to localhost:27017 (127.0.0.1:27017) :: caused by :: onInvoke :: caused by :: Connection refused',
syncSourceHost: '',
syncSourceId: -1,
infoMessage: '',
configVersion: 1,
configTerm: 2
},


{
_id: 1,
name: 'localhost:27018',
health: 0,
state: 8,
stateStr: '(not reachable/healthy)',
uptime: 0,
optime: { ts: Timestamp({ t: 0, i: 0 }), t: Long("-1") },
optimeDurable: { ts: Timestamp({ t: 0, i: 0 }), t: Long("-1") },
optimeDate: ISODate("1970-01-01T00:00:00.000Z"),
optimeDurableDate: ISODate("1970-01-01T00:00:00.000Z"),
lastAppliedWallTime: ISODate("2026-01-08T22:23:37.292Z"),
lastDurableWallTime: ISODate("2026-01-08T22:23:37.292Z"),
lastHeartbeat: ISODate("2026-01-08T22:25:14.590Z"),
lastHeartbeatRecv: ISODate("2026-01-08T22:23:55.874Z"),
pingMs: Long("0"),
lastHeartbeatMessage: 'Error connecting to localhost:27018 (127.0.0.1:27018) :: caused by :: onInvoke :: caused by :: Connection refused',
syncSourceHost: '',
syncSourceId: -1,
infoMessage: '',
configVersion: 1,
configTerm: 2
},


{
_id: 2,
name: 'localhost:27019',
health: 1,
state: 2,
stateStr: 'SECONDARY',
uptime: 5667,
optime: { ts: Timestamp({ t: 1767911017, i: 1 }), t: Long("2") },
optimeDate: ISODate("2026-01-08T22:23:37.000Z"),
lastAppliedWallTime: ISODate("2026-01-08T22:23:37.292Z"),
lastDurableWallTime: ISODate("2026-01-08T22:23:37.292Z"),
syncSourceHost: '',
syncSourceId: -1,
infoMessage: '',
configVersion: 1,
configTerm: 2,
self: true,
lastHeartbeatMessage: ''
}

Les ports 27017 et 27018 sont bien coupés et notre port 27019 est encore secondaire car il faut à un port au moins la majorité pour être élu primaire, hors il est seul sur trois ports à pouvoir voter, il ne peut donc pas atteindre la majorité.

4 Mise en place de Django

Pour commencer, il faut créer le projet Django, pour cela on utilise ces commandes :

```
django-admin startproject config .
python manage.py startapp movies
```

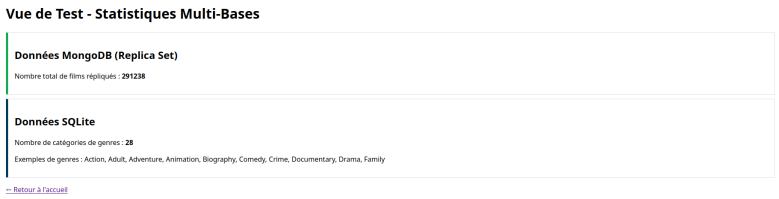
Elles permettent de créer une application 'movies' avec les fichiers nécessaires à Django.

Ensuite, on modifie le fichier settings.py de cineexplorer

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'movies',  <- On ajoute movies (le nom de notre app)
]

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'data' / 'imdb.db'
    }
}

MONGO_CONFIG = {
    'host': 'localhost:27017,localhost:27018,localhost
       :27019',
    'replicaSet': 'rs0',
    'db_name': 'MongoDB',
}

MONGO_URI = "mongodb://localhost:27017,localhost:27018,
   localhost:27019/?replicaSet=rs0"
MONGO_DB_NAME = "MongoDB"
```

Cette modification nous permet de nous connecter aux bases SQLite et MongoDB On remplit les fichiers sqlite_service.py et mongo_service.py avec nos fonctions permettant de récupérer des données depuis la base.

On termine cette partie en créant une vue (une page HTML), sans oublier d'ajouter le chemin dans le fichier urls.py de cineexplorer.



5 Conclusion

Pour conclure, nous avons vu dans cette partie la mise en place d'une base de données avec des réplicas, nous avons pu constater son efficacité contre les pannes.
Ensuite nous avons pu commencer la mise en place de Django en créant notre première vue.