3ème année cycle d'ingénieur - Spécialité Informatique.

TP3 - MongoDB Replica Set

Réalisé par :
ER-RACHDI Ilias

Encadrés par :
M. YOUCEF

Année Scolaire : 2025/2026

1 Introduction

Ce TP est consacré à l'étude de la réplication dans MongoDB. Nous allons mettre en place un **Replica Set** et comprendre son fonctionnement : sélection d'un nœud principal, propagation des opérations et bascule automatique en cas de panne.

Un Replica Set regroupe plusieurs processus MongoDB qui partagent les mêmes données. L'objectif est d'assurer la continuité de service, même si un serveur devient indisponible.

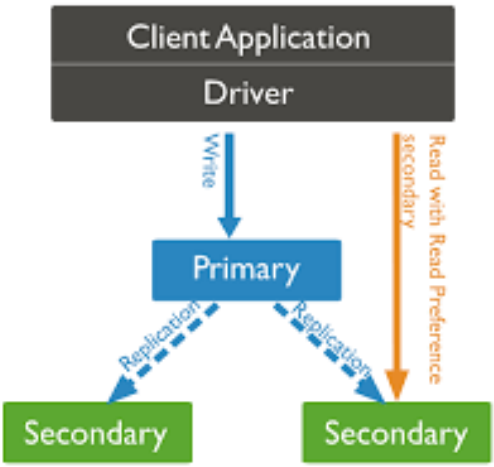

FIGURE 1 – Schéma simplifié d'un Replica Set

2 Initialisation du Replica Set

Nous avons utilisé Docker pour créer trois instances MongoDB faisant partie du même réseau.

2.1 Création du réseau

```
docker network create mycluster
```

2.2 Lancement des conteneurs

```
docker run -d --name mongo1 --net mycluster \
-p 27017:27017 mongo --replSet

docker run -d --name mongo2 --net mycluster \
-p 27018:27017 mongo --replSet

docker run -d --name mongo3 --net mycluster \
-p 27019:27017 mongo --replSet
```

2.3 Initialisation du cluster

Depuis `mongo1` :

```
docker exec -it mongo1 mongosh

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

La commande `rs.status()` permet ensuite de vérifier les rôles.

3 Fonctionnement général

Un Replica Set comprend :

— un **Primary** qui accepte toutes les écritures,

— des **Secondaries** qui répliquent l'oplog,

— éventuellement un **Arbitre** qui participe aux votes.

En cas d'indisponibilité du Primary, une **élection automatique** est déclenchée. Un Secondary devient alors Primary, tant qu'une majorité des membres est accessible.

Avantages principaux :

— haute disponibilité,

— aucune interruption lors d'un redémarrage,

— possibilité de répartir la charge en lecture.

4 Connexion au Replica Set

Deux méthodes sont possibles.

4.1 Connexion directe

```
mongosh "mongodb://IP:27017/?directConnection=true"
```

4.2 Connexion au cluster complet

```
mongosh \
"mongodb://IP:27017,IP:27018,IP:27019/?replicaSet=rs0"
```

MongoDB détecte automatiquement le Primary.

5 Questions et réponses sur la réplication

Dans cette partie, nous répondons aux questions relatives au fonctionnement des Replica Sets.

Rôle du Primary

Le Primary reçoit toutes les modifications. Il met à jour les données et écrit les opérations dans l'oplog. Les Secondaries rejouent ces opérations dans le même ordre.

Pourquoi ne pas écrire sur un Secondary ?

MongoDB impose une **source unique de vérité**. Accepter des écritures sur plusieurs nœuds provoquerait des conflits.

Lecture sur Secondary

Lire sur un Secondary est utile lorsque :

— la cohérence immédiate n'est pas indispensable,

— on souhaite soulager le Primary.

Cohérence forte

Pour garantir une lecture totalement à jour :

— `readPreference: primary`

— `writeConcern: majority`

— `readConcern: majority`

Arbitre

Un arbitre ne contient aucune donnée. Il sert uniquement à voter pour faciliter les élections.

Nombre impair de membres

Un nombre impair de votants facilite l'obtention d'une majorité. Sans majorité, aucune élection n'est possible.

Basculement du Primary

Si le Primary devient injoignable et qu'une majorité subsiste, un Secondary est élu automatiquement.

Retard de réplication

Un Secondary peut être configuré avec un retard volontaire (`slaveDelay`). Cela permet de se protéger d'erreurs humaines récentes.

Ajout et retrait de nœuds

Ajout :

```
rs.add("new:27017")
```

Retrait :

```
rs.remove("retrait :27017")
```

Surveillance

Plusieurs commandes permettent de suivre l'état :

```
rs.status()
rs.printSlaveReplicationInfo()
db.printReplicationInfo()
```

Split-brain

Si un Primary perd la majorité, il se dégrade volontairement en Secondary afin d'éviter une duplication divergente.

Réseau instable

Un réseau instable peut créer :

— des élections répétées,

— des erreurs d'écriture,

— une augmentation du retard de réplication.

6 Conclusion

Ce TP nous a permis de :

— créer un cluster MongoDB via Docker,

— configurer un Replica Set,

— comprendre le mécanisme d'élection,

— tester la bascule en cas de panne.

La réplication est un élément central de MongoDB, indispensable pour assurer la continuité de service et l'intégrité des données. Le Replica Set offre une solution robuste, automatique et adaptée aux environnements modernes où la disponibilité est primordiale.

7 Questions / Réponses sur la réplication MongoDB

Partie 1 — Compréhension de base

Question 1 : Qu'est-ce qu'un Replica Set dans MongoDB ?

Réponse : Dans MongoDB, un Replica Set correspond à un ensemble d'instances `mongod` qui partagent la même base de données. Les écritures effectuées sur le nœud principal sont enregistrées dans un journal d'opérations (oplog), puis rejouées sur les autres nœuds. Cela permet à la fois de dupliquer les données et d'assurer la haute disponibilité grâce à un *Primary*, un ou plusieurs *Secondaries* et éventuellement des arbitres qui participent uniquement aux votes.

Question 2 : Quel est le rôle du Primary dans un Replica Set ?

Réponse : Le nœud *Primary* est le seul qui accepte les opérations d'écriture. Il applique ces modifications aux collections, les consigne dans l'oplog, puis les autres nœuds (Secondaries) se synchronisent en lisant cet oplog. Par défaut, les lectures des clients sont également dirigées vers ce Primary lorsque la préférence de lecture est `"primary"`.

Question 3 : Quel est le rôle essentiel des Secondaries ?

Réponse : Les nœuds *Secondaries* maintiennent une copie quasi identique des données en rejouant les opérations présentes dans l'oplog du Primary. Ils peuvent :

— prendre le relais et devenir Primary lors d'une élection s'ils sont suffisamment à jour ;

— répondre aux requêtes de lecture si le client utilise une préférence de lecture adaptée (`"secondary"`, `"secondaryPreferred"`, etc.).

Question 4 : Pourquoi MongoDB n'autorise-t-il pas les écritures sur un Secondary ?

Réponse : Si plusieurs nœuds pouvaient accepter des écritures simultanément, il faudrait résoudre des conflits potentiellement nombreux entre des versions divergentes des données. Pour éviter cela, MongoDB impose un point d'entrée unique pour les mises à jour : le Primary. Les Secondaries ne font que répliquer ce qui a été validé sur le Primary, ce qui simplifie fortement la cohérence.

Question 5 : Qu'est-ce que la cohérence forte dans le contexte MongoDB ?

Réponse : On parle de cohérence forte lorsqu'un client, après avoir effectué une écriture confirmée, lit une donnée qui reflète cette modification. En pratique, cela signifie :

- lire sur le Primary ;
- utiliser un `writeConcern` adapté (par exemple `"majority"`) ;
- combiner avec un `readConcern` tel que `"majority"` voire `"linearizable"`.

Dans ces conditions, les lectures renvoient un état des données cohérent avec les écritures validées.

Question 6 : Différence entre `readPreference: "primary"` et `"secondary"` ?

Réponse :

- `"primary"` : toutes les lectures sont envoyées vers le Primary. On maximise la cohérence mais on concentre la charge sur un seul nœud.
- `"secondary"` : les lectures sont dirigées uniquement vers les Secondaries. Le Primary est soulagé, mais les données consultées peuvent avoir un léger retard dû au caractère asynchrone de la réplication.

Question 7 : Dans quel cas lire sur un Secondary malgré les risques ?

Réponse : On accepte de lire sur un Secondary lorsque des données potentiellement légèrement obsolètes ne posent pas de problème, par exemple :

- génération de rapports, tableaux de bord, statistiques ;
- exports ou tâches batch ;
- lectures réalisées sur un Secondary géographiquement proche de l'utilisateur dans un cluster distribué.

Partie 2 — Commandes & configuration

Question 8 : Quelle commande permet d'initialiser un Replica Set ?

Réponse : Après s'être connecté sur un nœud (par exemple `mongo1`) avec `mongosh`, on peut initialiser le Replica Set ainsi :

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

Question 9 : Comment ajouter un nœud à un Replica Set après son initialisation ?

Réponse : On commence par démarrer un nouveau conteneur, par exemple `mongo4`, dans le même réseau :

```
docker run -d --name mongo4 --net mycluster \
  -p 27020:27017 \
  mongo --replSet
```

Puis, depuis le Primary :

```
rs.add("<host_ip>:27020")
```

Question 10 : Quelle commande permet d'afficher l'état actuel du Replica Set ?

Réponse : Depuis `mongosh`, on utilise :

```
rs.status()
```

Via Docker, on peut aussi exécuter :

```
docker exec -it mongo1 mongosh --eval "rs.status()"
```

Question 11 : Comment identifier le rôle actuel (Primary / Secondary / Arbiter) d'un nœud ?

Réponse :

— Sur un nœud donné : `db.isMaster()` (ou `rs.isMaster()`) indique si le nœud est `ismaster: true` (Primary) ou `secondary: true`.

— Pour une vue globale : `rs.status().members` fournit, pour chaque membre, le champ `stateStr` (`"PRIMARY"`, `"SECONDARY"`, `"ARBITER"`).

Question 12 : Quelle commande permet de forcer le basculement du Primary ?

Réponse : On peut demander au Primary actuel de renoncer temporairement à son rôle :

```
docker exec -it mongo1 mongosh --eval "rs.stepDown(60)"
```

Cela déclenche une nouvelle élection.

Question 13 : Comment désigner un nœud comme Arbitre ? Pourquoi le faire ?

Réponse : On lance d'abord un conteneur sans données :

```
docker run -d --name mongo-arb --net mycluster \
  mongo --replSet
```

Puis, depuis le Primary :

```
rs.addArb("mongo-arb:27017")
```

Un arbitre n'héberge pas de données, mais il permet d'augmenter le nombre de voix pour atteindre plus facilement une majorité lors des élections.

Question 14 : Commande pour configurer un Secondary avec un délai de réplication (`slaveDelay`)

Réponse : Après avoir lancé le conteneur retardé :

```
docker run -d --name mongo-delayed --net mycluster \
  -p 27021:27017 \
  mongo --replSet
```

On l'ajoute dans la configuration avec un délai :

```
rs.add({
  host: "mongo-delayed:27017",
  priority: 0,
  hidden: true,
  slaveDelay: 120
})
```

Partie 3 — Résilience et tolérance aux pannes

Question 15 : Que se passe-t-il si le Primary tombe en panne et qu'il n'y a pas de majorité ?

Réponse : Si aucun groupe de nœuds n'atteint la majorité des votes, MongoDB ne peut élire aucun nouveau Primary. Le Replica Set fonctionne alors sans Primary :

— les écritures deviennent impossibles ;
— seules des lectures sur des Secondaries sont envisageables (si la configuration client le permet).

Question 16 : Comment MongoDB choisit-il un nouveau Primary ? Quels critères sont utilisés ?

Réponse : Lors d'une élection :
— seuls les nœuds appartenant au groupe majoritaire peuvent être candidats ;
— MongoDB favorise les nœuds dont la priorité est la plus élevée ;
— parmi ceux ayant la même priorité, celui qui est le plus à jour au niveau de l'oplog a l'avantage.

Question 17 : Qu'est-ce qu'une élection dans MongoDB ?

Réponse : Il s'agit d'un mécanisme automatique par lequel les membres votants décident quel nœud doit jouer le rôle de Primary. Un nœud candidat sollicite les voix des autres ; s'il obtient la majorité, il est promu Primary.

Question 18 : Que signifie auto-dégradation du Replica Set ? Dans quel cas cela survient-il ?

Réponse : On parle d'auto-dégradation lorsqu'un Primary se transforme lui-même en Secondary après avoir détecté qu'il ne fait plus partie de la majorité (par exemple lors d'une partition réseau). Il se met alors en lecture seule pour éviter d'accepter des écritures qui seraient en conflit avec le reste du cluster (protection contre le *split-brain*).

Question 19 : Pourquoi est-il conseillé d'avoir un nombre impair de nœuds dans un Replica Set ?

Réponse : Avec un nombre impair de membres votants, il est plus simple d'obtenir une majorité claire. Cela réduit les cas d'égalité et donc les situations où aucune décision (et aucune élection) ne peut être prise.

Question 20 : Quelles conséquences a une partition réseau sur le fonctionnement du cluster ?

Réponse : En cas de partition du réseau :
— la partition qui conserve la majorité peut garder ou élire un Primary ;
— les partitions minoritaires ne peuvent pas avoir de Primary et se retrouvent en mode lecture seule.

Partie 4 — Scénarios pratiques

Question 21 : 3 nœuds : 27017 (Primary), 27018 (Secondary), 27019 (Arbitre). Que se passe-t-il si le Primary devient injoignable ?

Réponse : Il reste alors deux membres votants : le Secondary et l'Arbitre. Comme ils forment encore la majorité, ils peuvent organiser une élection, et le Secondary peut être élu comme nouveau Primary.

Question 22 : Secondary avec `slaveDelay = 120` secondes : utilité et usages ?

Réponse : Ce nœud applique les opérations avec un décalage de deux minutes par rapport au Primary. Cela permet, par exemple :

— de revenir en arrière après une suppression ou modification accidentelle récente ;
— d'effectuer des analyses sur un état légèrement antérieur de la base.

En revanche, ce nœud n'est pas adapté pour des lectures nécessitant des données à jour.

Question 23 : Un client exige une lecture toujours à jour, même en cas de bascule. Que recommander en `readConcern` et `writeConcern` ?

Réponse : On recommande de :

— écrire avec `writeConcern: "majority"` ;
— utiliser `readPreference: "primary"` ;
— lire avec `readConcern: "majority"` ou `"linearizable"` suivant le niveau de garantie souhaité.

Question 24 : Garantir que l'écriture est confirmée par au moins deux nœuds. Quel `writeConcern` ?

Réponse : Dans un Replica Set à trois membres, on peut utiliser :

— `writeConcern: { w: 2 }` pour exiger l'acquittement d'au moins deux nœuds ;
— ou `"majority"`, ce qui donne un résultat équivalent dans ce cas.

Question 25 : Un étudiant lit depuis un Secondary et récupère une donnée obsolète. Pourquoi, et comment éviter ça ?

Réponse : La réplication étant asynchrone, le Secondary peut avoir un léger retard sur le Primary. Il est donc possible de lire une version plus ancienne des données. Pour éviter ce problème, on peut :

— lire directement sur le Primary ;
— utiliser un `readConcern` de type `"majority"` ;
— surveiller régulièrement le *lag* des Secondaries.

Question 26 : Commande pour vérifier quel nœud est actuellement Primary dans le Replica Set

Réponse : Par exemple, depuis `mongo1` :

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

Question 27 : Forcer une bascule manuelle du Primary sans interruption majeure

Réponse : Une approche consiste à vérifier d'abord l'état de réplication, puis à demander au Primary actuel de se retirer :

```
docker exec -it mongo1 mongosh --eval "rs.printSlaveReplicationInfo()"
docker exec -it mongo1 mongosh --eval "rs.stepDown(60)"
```

Question 28 : Procédure pour ajouter un nouveau nœud secondaire dans un Replica Set en fonctionnement

Réponse :

1. Lancer le nouveau conteneur :

   ```
   docker run -d --name mongo4 --net mycluster \
     -p 27020:27017 \
     mongo --replSet
   ```

2. Depuis le Primary, l'ajouter au Replica Set :

   ```
   rs.add("mongo4:27017")
   ```

Question 29 : Quelle commande permet de retirer un nœud défectueux d'un Replica Set ?

Réponse : On retire d'abord le nœud de la configuration :

```
docker exec -it mongo1 mongosh --eval 'rs.remove("mongo4:27017")'
```

puis on peut arrêter et supprimer le conteneur :

```
docker stop mongo4
docker rm mongo4
```

Question 30 : Configurer un nœud secondaire pour qu'il soit caché (non visible aux clients). Pourquoi ?

Réponse : On modifie la configuration :

```
cfg = rs.conf()
cfg.members[2].hidden = true
cfg.members[2].priority = 0
rs.reconfig(cfg)
```

Un nœud « caché » ne reçoit pas de trafic de lecture des applications. Il peut être réservé à des usages particuliers (sauvegardes, reporting, analyses) sans impacter le reste du cluster.

Question 31 : Modifier la priorité d'un nœud afin qu'il devienne le Primary préféré

Réponse : On ajuste les priorités dans la configuration :

```
cfg = rs.conf()
cfg.members[0].priority = 10
cfg.members[1].priority = 1
cfg.members[2].priority = 0
rs.reconfig(cfg)
```

Ainsi, le nœud avec la priorité la plus élevée sera favorisé lors des élections.

Question 32 : Vérifier le délai de réplication d'un Secondary par rapport au Primary

Réponse : On peut utiliser :

```
docker exec -it mongo1 mongosh --eval "rs.printSlaveReplicationInfo()"
```

qui affiche, pour chaque Secondary, le retard de réplication par rapport au Primary.

Question 33 : Que fait la commande `rs.freeze()` et dans quel scénario est-elle utile ?

Réponse : La commande `rs.freeze(N)` empêche un nœud de se porter candidat comme Primary pendant `N` secondes :

```
docker exec -it mongo2 mongosh --eval "rs.freeze(300)"
```

C'est utile lorsqu'on veut éviter qu'un certain nœud devienne Primary (par exemple parce qu'il est moins puissant ou géographiquement moins bien placé).

Question 34 : Comment redémarrer un Replica Set sans perdre la configuration ?

Réponse : La configuration du Replica Set est stockée dans la base locale (`local`). Tant que le répertoire de données (`dbPath`) est conservé et que l'option `-replSet` reste identique, la configuration persiste après redémarrage :

```
docker restart mongo1
docker restart mongo2
docker restart mongo3
```

Question 35 : Surveiller en temps réel la réplication via les logs MongoDB ou commandes shell

Réponse : On peut utiliser :

— les commandes `rs.status()`, `rs.printSlaveReplicationInfo()`, `db.printReplicationInfo(`

— la consultation des logs, par exemple :

```
docker logs -f mongo1
docker exec -it mongo1 mongosh --eval "rs.printSlaveReplicationInfo()"
```

Questions complémentaires

Question 37 : Qu'est-ce qu'un Arbitre (Arbiter) et pourquoi ne stocke-t-il pas de données ?

Réponse : Un arbitre est un membre spécial du Replica Set qui ne contient aucune donnée utilisateur et ne peut pas devenir Primary. Son rôle est uniquement de participer aux votes pour aider à atteindre une majorité. On le crée par exemple ainsi :

```
docker run -d --name mongo-arb --net mycluster \
  mongo --replSet
docker exec -it mongo1 mongosh --eval 'rs.addArb("mongo-arb:27017")'
```

Question 38 : Comment vérifier la latence de réplication entre le Primary et les Secondaries ?

Réponse : On utilise la commande :

```
docker exec -it mongo1 mongosh --eval "rs.printSlaveReplicationInfo()"
```

qui affiche le retard de chaque Secondary par rapport au Primary.

Question 39 : Quelle commande MongoDB permet d'afficher le retard de réplication des membres secondaires ?

Réponse : La commande est :

```
rs.printSlaveReplicationInfo()
```

Question 40 : Différence entre réplication asynchrone et synchrone ? Quel type utilise MongoDB ?

Réponse :

— Réplication synchrone : les écritures ne sont considérées comme validées que lorsque plusieurs nœuds ont confirmé l'opération, ce qui peut augmenter la latence.

— Réplication asynchrone : le Primary valide l'écriture sans attendre que tous les Secondaries aient appliqué l'opération.

MongoDB utilise une réplication asynchrone, mais permet de contrôler le niveau de garantie à l'aide de `writeConcern`.

Question 41 : Peut-on modifier la configuration d'un Replica Set sans redémarrer les serveurs ?

Réponse : Oui, c'est possible. On récupère la configuration avec `rs.conf()`, on la modifie, puis on applique les changements avec `rs.reconfig()` sans devoir arrêter les instances.

Question 42 : Que se passe-t-il si un nœud Secondary est en retard de plusieurs minutes ?

Réponse : Un important retard signifie que ce Secondary ne peut pas devenir Primary tant qu'il n'a pas rattrapé les autres. Si l'oplog sur le Primary ne contient plus les opérations manquantes, le nœud devra éventuellement se resynchroniser complètement, ce qui peut être coûteux.

Question 43 : Comment MongoDB gère-t-il les conflits de données lors de la réplication ?

Réponse : Comme seul le Primary accepte les écritures, la conception même du Replica Set évite les conflits. Dans des cas extrêmes (par exemple lors de rollbacks), MongoDB peut annuler certaines opérations pour revenir à un état cohérent, mais il n'y a pas d'édition concurrente sur plusieurs nœuds.

Question 44 : Est-il possible d'avoir plusieurs Primary simultanément dans un Replica Set ? Pourquoi ?

Réponse : En fonctionnement normal, non. Le protocole d'élection et la notion de majorité sont précisément conçus pour empêcher la présence de deux Primary actifs en même temps, afin d'éviter le phénomène de *split-brain* et des divergences irréconciliables.

Question 45 : Pourquoi est-il déconseillé d'utiliser un Secondary pour des opérations d'écriture même en lecture préférée secondaire ?

Réponse : Les Secondaries ne sont pas destinés à recevoir des écritures applicatives et les pilotes MongoDB standard l'interdisent. Toute écriture directe sur un Secondary risquerait d'être écrasée par la réplication en provenance du Primary, ce qui rendrait l'état des données incohérent.

Question 46 : Quelles sont les conséquences d'un réseau instable sur un Replica Set ?

Réponse : Un réseau de mauvaise qualité peut provoquer :

— des élections fréquentes et des changements répétés de Primary (*step-down*) ;

— des erreurs lors des écritures si le Primary perd la majorité ;

— une augmentation du retard de réplication et, dans certains cas, des rollbacks.