TP2 – Introduction à la réplication sur MongoDB

Base de Données NoSQL

NICOLAS Ethan

G5SI2
2025–2026

Table des matières

1 Initialisation du mode Replica Set sur MongoDB

1.1 MongoDB Replica Set, c'est quoi ?

L'objectif d'un système de gestion de base de données comme MongoDB est de conserver les données de manière fiable, durable et disponible. Les *Replica Sets* sont le mécanisme natif de haute disponibilité et de tolérance aux pannes de MongoDB.

Un Replica Set est un groupe de processus `mongod` qui maintiennent la **même** base de données en répliquant un journal d'opérations (*oplog*). Il contient :

— un **nœud primaire** (*Primary*) qui reçoit toutes les écritures ;

— un ou plusieurs **nœuds secondaires** (*Secondaries*) qui répliquent les opérations du Primary ;

— éventuellement un ou plusieurs **arbitres** (*Arbiters*) qui ne stockent pas de données mais participent au vote.

On recommande d'avoir un **nombre impair de membres votants** (nœuds de données + arbitres) afin de faciliter l'obtention d'une majorité lors des élections.

1.2 Fonctionnement général d'un Replica Set

Un Replica Set MongoDB se compose donc :

— d'un nœud **Primary**, point d'entrée pour toutes les opérations d'écriture ;

— de nœuds **Secondaries** qui maintiennent une copie de la base en appliquant l'oplog du Primary de manière *asynchrone*.

Schéma simplifié :

— les clients envoient les écritures au Primary ;

— le Primary écrit dans ses collections **et** dans l'oplog ;

— chaque Secondary lit l'oplog du Primary et rejoue les opérations dans le même ordre.

En cas de panne du Primary (crash, maintenance, coupure réseau), les membres restants organisent une **élection** : si une majorité de votes est atteinte, un Secondary à jour devient le nouveau Primary et continue à accepter les écritures. Quand l'ancien Primary revient, il se resynchronise et redevient Secondary.

En pratique, un Replica Set de production contient au minimum **trois membres votants** (par exemple 2 nœuds de données + 1 arbitre) pour garantir une majorité même si un nœud tombe.

1.3 Avantages principaux d'un Replica Set

Les Replica Sets apporte plusieurs garanties importantes :

— **Haute disponibilité** : si le Primary tombe, un Secondary à jour peut être élu Primary automatiquement.

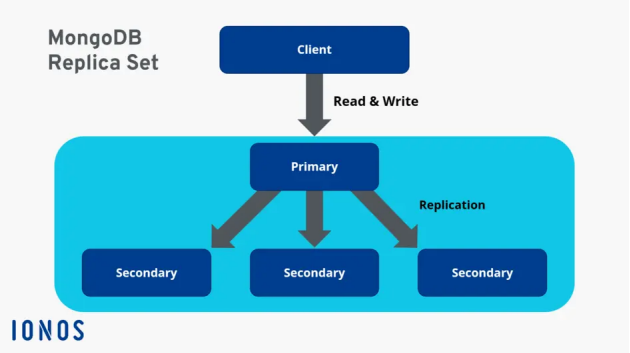

FIGURE 1 – Schéma simplifié d'un Replica Set MongoDB

— **Tolérance aux pannes** : les données sont stockées sur plusieurs nœuds ; la perte d'un nœud ne signifie pas perte des données.

— **Maintenance à chaud** : on peut redémarrer ou mettre à jour un nœud secondaire sans arrêter le service global.

— **Répartition de charge en lecture** : pour certains cas, on peut déléguer des lectures à des Secondaries (rapports, analytics, sauvegardes, etc.).

Ces mécanismes réduisent fortement le temps d'indisponibilité et le risque de perte de données dû à une panne matérielle ou logicielle.

1.4 Installation de MongoDB en mode Replica Set via Docker

Les commandes ci-dessous sont inspirées de la documentation officielle de MongoDB. L'idée est de lancer 3 conteneurs `mongo` dans un même réseau Docker et de les joindre dans un Replica Set nommé `rs0`.

```
# Création du réseau virtuel pour le cluster
docker network create mongo-cluster

# Lancement de 3 instances MongoDB dans ce réseau virtuel, avec des ports différents
# --replSet rs0 indique que l'instance fera partie du Replica Set "rs0"

docker run -d --name mongo1 --net mongo-cluster \
  -p 27017:27017 \
  mongo \
  --replSet rs0 --bind_ip_all

docker run -d --name mongo2 --net mongo-cluster \
  -p 27018:27017 \
  mongo \
  --replSet rs0 --bind_ip_all

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

```
  --replSet rs0 --bind_ip_all

# Vérification du lancement des conteneurs
docker ps

CONTAINER ID   IMAGE     COMMAND                  CREATED       STATUS       PORTS
573d38a75a45   mongo     "docker-entrypoint.s..."   5 hours ago   Up 5 hours   0.0.0.0:27019->2
cb0485400286   mongo     "docker-entrypoint.s..."   5 hours ago   Up 4 hours   0.0.0.0:27018->2
34a3055a1b18   mongo     "docker-entrypoint.s..."   5 hours ago   Up 3 hours   0.0.0.0:27017->2

# Récupération de l'IP hôte qui sera vue par les trois conteneurs
# (localhost ne convient pas car chaque conteneur a son propre localhost)
ifconfig | grep "inet "   # sur MacOS par exemple

# Puis initialisation du Replica Set depuis le noeud qui jouera le rôle de Primary (mongo1)
docker exec -it mongo1 mongosh

# Dans mongosh : initier le Replica Set rs0 avec ip:port de chaque instance
rs.initiate({
  _id: "rs0",
  members: [
    { _id: 0, host: "<host_ip>:27017" },
    { _id: 1, host: "<host_ip>:27018" },
    { _id: 2, host: "<host_ip>:27019" }
  ]
})

# En cas de changement d'IP, on peut reconfigurer la configuration
cfg = rs.conf()
cfg.members[0].host = "<new_host_ip>:27017"
cfg.members[1].host = "<new_host_ip>:27018"
cfg.members[2].host = "<new_host_ip>:27019"
rs.reconfig(cfg, { force: true })

# Pour vérifier l'état des membres du Replica Set
rs.status().members.map(m => ({ name: m.name, stateStr: m.stateStr }))

# Exemple de résultat
[
  { name: '192.0.0.2:27017', stateStr: 'PRIMARY' },
  { name: '192.0.0.2:27018', stateStr: 'SECONDARY' },
  { name: '192.0.0.2:27019', stateStr: 'SECONDARY' }
]
```

1.5 Connexion au Replica Set

On peut se connecter au cluster de deux manières :

```
# Connexion directe à un noeud (ici le Primary présumé sur 27017)
mongosh "mongodb://<host_ip>:27017/?directConnection=true"

# Connexion au Replica Set complet rs0 (recommandé en production)
mongosh "mongodb://<host_ip>:27017,<host_ip>:27018,<host_ip>:27019/?replicaSet=rs0"

# Exemple sur une machine donnée
mongosh "mongodb://192.0.0.2:27017,192.0.0.2:27018,192.0.0.2:27019/?replicaSet=rs0"
```

Le driver MongoDB ou `mongosh` détecte alors automatiquement le Primary et envoie les écritures vers ce nœud. En cas de bascule, il se reconnecte au nouveau Primary sans changer la chaîne de connexion.

2 Questions / Réponses sur la réplication MongoDB

Partie 1 — Compréhension de base

1. Qu'est-ce qu'un Replica Set dans MongoDB ? Un Replica Set est un groupe de processus `mongod` qui maintiennent la même base de données en répliquant un journal d'opérations (*oplog*). Il fournit redondance des données et haute disponibilité grâce à un Primary, un ou plusieurs Secondaries et éventuellement des Arbitres.

2. Quel est le rôle du Primary dans un Replica Set ? Le Primary reçoit **toutes les opérations d'écriture**. Il applique ces opérations aux données, les enregistre dans l'oplog puis les Secondaries les rejouent. Par défaut, les lectures des clients sont également envoyées au Primary (readPreference `"primary"`).

3. Quel est le rôle essentiel des Secondaries ? Les Secondaries répliquent l'oplog du Primary et maintiennent une copie quasi à jour des données. Ils peuvent :

- être promus Primary lors d'une élection ;
- servir des lectures si la `readPreference` du client le permet (`"secondary"`, `"secondaryPreferred"`, etc.).

4. Pourquoi MongoDB n'autorise-t-il pas les écritures sur un Secondary ? Pour garantir une **source de vérité unique** et éviter les conflits de mise à jour. Si plusieurs nœuds acceptaient des écritures en parallèle, il faudrait un mécanisme complexe de résolution de conflits. MongoDB impose donc que toutes les écritures passent par le Primary, qui est ensuite répliqué de façon unidirectionnelle vers les Secondaries.

5. Qu'est-ce que la cohérence forte dans le contexte MongoDB ? Dans ce contexte, on parle de cohérence forte quand un client lit une donnée qui reflète les dernières écritures confirmées :

- lecture sur le **Primary** ;
- avec un **readConcern** approprié (`"majority"` ou `"linearizable"`) combiné à un **writeConcern** adapté (par exemple `"majority"`).

Ainsi, une lecture après une écriture confirmée renvoie une vue à jour des données.

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

- `"primary"` : toutes les lectures vont au Primary. Cohérence maximale, mais charge concentrée sur un seul nœud.
- `"secondary"` : les lectures vont uniquement vers les Secondaries. On décharge le Primary, mais on accepte de possibles **données légèrement en retard** à cause de la réplication asynchrone.

7. Dans quel cas lire sur un Secondary malgré les risques ? Typiquement quand une légère obsolescence est acceptable :

- rapports, statistiques, dashboards analytiques ;
- exports, sauvegardes, traitements batch ;
- lectures géolocalisées sur un Secondary proche de l'utilisateur dans un Replica Set distribué géographiquement.

Partie 2 — Commandes & configuration

8. Quelle commande permet d'initialiser un Replica Set ?

```
docker exec -it mongo1 mongosh

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

9. Comment ajouter un nœud à un Replica Set après son initialisation ? Ajout d'un nouveau conteneur `mongo4` :

```
# 1) Lancer un nouveau conteneur dans le même réseau Docker et un nouveau port
docker run -d --name mongo4 --net mongo-cluster \
  -p 27020:27017 \
  mongo \
  --replSet rs0 --bind_ip_all

# 2) Depuis le Primary (mongo1 par exemple), l'ajouter au Replica Set
docker exec -it mongo1 mongosh

# Dans mongosh :
rs.add("<host_ip>:27017")
```

10. Quelle commande permet d'afficher l'état actuel du Replica Set ? `rs.status()` dans `mongosh` affiche l'état détaillé de chaque membre.

Version Docker :

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

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

— Sur un nœud donné : `db.isMaster()` (ou `rs.isMaster()`) indique `"ismaster": true` pour le Primary et `"secondary": true` pour un Secondary.

— Globalement : `rs.status().members` contient le champ `stateStr` (`"PRIMARY"`, `"SECONDARY"`, `"ARBITER"`).

```
docker exec -it mongo1 mongosh --eval "db.isMaster()"
docker exec -it mongo2 mongosh --eval "db.isMaster()"
```

12. Quelle commande permet de forcer le basculement du Primary ?

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

13. Comment désigner un nœud comme Arbitre ? Pourquoi le faire ?

```
# 1) Lancer un conteneur pour l'arbitre
docker run -d --name mongo-arb --net mongo-cluster \
  mongo \
  --replSet rs0 --bind_ip_all

# 2) Depuis le Primary, déclarer l'arbitre
docker exec -it mongo1 mongosh

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

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

```
# 1) Lancer le conteneur
docker run -d --name mongo-delayed --net mongo-cluster \
  -p 27021:27017 \
  mongo \
  --replSet rs0 --bind_ip_all

# 2) Configurer dans le Replica Set
docker exec -it mongo1 mongosh

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

Partie 3 — Résilience et tolérance aux pannes

15. Que se passe-t-il si le Primary tombe en panne et qu'il n'y a pas de majorité ? Sans majorité de votes, aucun nouveau Primary ne peut être élu. Le Replica Set se retrouve **sans Primary** :

— les écritures échouent ;

— seules des lectures sur des Secondaries sont possibles (si le client l'autorise).

16. Comment MongoDB choisit-il un nouveau Primary ? Critères utilisés ? Lors d'une élection :

— seuls les membres dans la **majorité de votes** sont éligibles ;

— parmi eux, priorité à ceux dont la **priority** est la plus élevée ;

— à priorité égale, le choix se fait en fonction de l'**oplog le plus à jour**.

17. Qu'est-ce qu'une élection dans MongoDB ? C'est le protocole automatique par lequel les membres votants décident quel nœud sera Primary. Un candidat se présente, demande les votes des autres, et devient Primary s'il obtient la majorité.

18. Que signifie auto-dégradation du Replica Set ? Dans quel cas cela survient-il ? Auto-dégradation = un nœud Primary se **dégrade** lui-même en Secondary quand il détecte qu'il a perdu la majorité (partition réseau). Il se met en mode lecture seule pour éviter un *split-brain*.

19. Pourquoi est-il conseillé d'avoir un nombre impair de nœuds dans un Replica Set ? Pour obtenir plus facilement une **majorité absolue** lors des votes et éviter les blocages.

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

— la partition avec la majorité garde ou élit un Primary ;

— les partitions minoritaires deviennent read-only (pas de Primary).

Partie 4 — Scénarios pratiques

21. 3 nœuds : 27017 (Primary), 27018 (Secondary), 27019 (Arbitre). Que se passe-t-il si le Primary devient injoignable ? Il reste 2 votes sur 3 : le Secondary et l'Arbitre. Ils forment une majorité et peuvent élire le Secondary comme nouveau Primary.

22. Secondary avec `slaveDelay = 120` secondes : utilité et usages ? Ce Secondary applique les opérations avec 120 s de retard. Utilité : rattrapage d'erreurs humaines, audit, analyses « dans le passé proche ». À ne pas utiliser pour des lectures critiques.

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

— `writeConcern: "majority"` ;

— `readPreference: "primary"` ;

— `readConcern: "majority"` ou `"linearizable"`.

24. Garantir que l'écriture est confirmée par au moins deux nœuds. Quel `writeConcern` ? `writeConcern: { w: 2 }` ou `"majority"` (dans un Replica Set à 3 membres).

25. Un étudiant lit depuis un Secondary et récupère une donnée obsolète. Pourquoi, et comment éviter ça ? Réplication asynchrone, retard possible du Secondary. Éviter : lire sur le Primary, `readConcern: "majority"`, surveiller le lag.

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

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

27. Forcer une bascule manuelle du Primary sans interruption majeure

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

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

```
# 1) Lancer le nouveau conteneur
docker run -d --name mongo4 --net mongo-cluster \
  -p 27020:27017 \
  mongo \
  --replSet rs0 --bind_ip_all

# 2) Depuis le Primary, l'ajouter au Replica Set
docker exec -it mongo1 mongosh

# Dans mongosh :
rs.add("mongo4:27017")
```

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

```
# Retirer du Replica Set
docker exec -it mongo1 mongosh --eval 'rs.remove("mongo4:27017")'

# Arrêter et supprimer le conteneur
docker stop mongo4
docker rm mongo4
```

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

```
docker exec -it mongo1 mongosh

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

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

```
docker exec -it mongo1 mongosh

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

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

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

33. Que fait la commande `rs.freeze()` et dans quel scénario est-elle utile ? `rs.freeze(N)` empêche un nœud de devenir Primary pendant `N` secondes.

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

34. Comment redémarrer un Replica Set sans perdre la configuration ? La configuration est stockée dans la base `local`. Tant que le `dbPath` est conservé et que `-replSet` reste le même, la configuration est conservée.

```
# Redémarrage sans perte de configuration grâce aux volumes
docker restart mongo1
docker restart mongo2
docker restart mongo3
```

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

— `rs.status()`, `rs.printSlaveReplicationInfo()`, `db.printReplicationInfo()` ;

— lecture des logs.

```
# Logs du Primary
docker logs -f mongo1

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

Questions complémentaires

37. Qu'est-ce qu'un Arbitre (Arbiter) et pourquoi ne stocke-t-il pas de données ?
Un Arbitre est un membre du Replica Set qui ne stocke pas de données, ne peut pas devenir Primary, et ne sert qu'à voter.

```
docker run -d --name mongo-arb --net mongo-cluster \
  mongo \
  --replSet rs0 --bind_ip_all

docker exec -it mongo1 mongosh --eval 'rs.addArb("mongo-arb:27017")'
```

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

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

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

```
rs.printSlaveReplicationInfo()
```

40. Différence entre réplication asynchrone et synchrone ? Quel type utilise MongoDB ? MongoDB utilise une réplication **asynchrone** avec contrôle via `writeConcern`.

41. Peut-on modifier la configuration d'un Replica Set sans redémarrer les serveurs ? Oui, via `rs.conf()` et `rs.reconfig()`.

42. Que se passe-t-il si un nœud Secondary est en retard de plusieurs minutes ?
Retard important, impossibilité temporaire d'être élu Primary, possible resynchronisation complète si l'oplog n'est plus suffisant.

43. Comment MongoDB gère-t-il les conflits de données lors de la réplication ?
Un seul nœud accepte les écritures (Primary). Les conflits sont évités par conception, les cas extrêmes sont gérés par rollback.

44. Est-il possible d'avoir plusieurs Primary simultanément dans un Replica Set ? Pourquoi ? En fonctionnement normal non. Les mécanismes d'élection et de majorité empêchent le *split-brain*.

45. Pourquoi est-il déconseillé d'utiliser un Secondary pour des opérations d'écriture même en lecture préférée secondaire ? Un Secondary ne doit pas recevoir d'écritures applicatives ; les drivers standard l'interdisent. Les données pourraient être écrasées par la réplication.

46. Quelles sont les conséquences d'un réseau instable sur un Replica Set ? Élections fréquentes, step-down répétés, erreurs d'écriture, augmentation du lag, risque de rollback.