Rapport TP2 - partie 2

Réplication et tolérance aux pannes avec MongoDB

Addou Aicha Amira

Bombay Marc

3ème Année Ingénieur - INFO - Apprentissage

Introduction

Ce rapport présente la mise en œuvre d'un **Replica Set** dans MongoDB, un mécanisme essentiel pour assurer la haute disponibilité, la tolérance aux pannes et la réplication des données dans les systèmes distribués.
Après avoir reproduit les manipulations vues dans la vidéo du TP, nous répondons ici de manière détaillée aux questions concernant la configuration, le fonctionnement, la résilience et les scénarios pratiques liés aux Replica Sets.

Partie 1 - Compréhension de base

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

Un Replica Set est un groupe de serveurs MongoDB contenant les mêmes données, où un nœud est **Primary** (gère les écritures), et les autres **Secondaries** (répliquent les données). Il assure la haute disponibilité grâce à l'élection automatique d'un nouveau Primary en cas de panne.

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

Le Primary reçoit **toutes les écritures** et sert de source de vérité pour la réplication. Il applique les écritures dans son oplog, que les Secondaries utilisent pour se synchroniser.

3. Quel est le rôle essentiel des Secondaries ?

Ils répliquent les données du Primary et prennent le relai en cas de défaillance du Primary grâce au mécanisme d'élection.

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

Pour éviter les conflits de données et maintenir une cohérence stricte du journal d'opérations (oplog).
MongoDB utilise une **réplication asynchrone** basée sur un flux d'écritures ordonné.

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

Lire sur le Primary garantit d'obtenir la **dernière version commitée**.
C'est le mode le plus sûr pour éviter la lecture de données obsolètes.

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

- `"primary"` : lecture toujours à jour, cohérente.
- `"secondary"` : lecture potentiellement plus rapide mais **risque de données périmées** (lag de réplication).

7. Quand pourrait-on lire sur un Secondary ?

Pour **décharger le Primary** lors de forte charge en lecture ou pour des analyses non critiques (reporting, BI).

Partie 2 - Commandes & configuration

8. Commande pour initialiser un Replica Set

```
rs.initiate()
```

Crée la configuration initiale permettant aux nœuds de fonctionner en cluster.

9. Ajouter un nœud après initialisation

```
rs.add("hostname:port")
```

10. Afficher l'état du Replica Set

```
rs.status()
```

11. Identifier le rôle d'un nœud

```
rs.isMaster()

// ou depuis MongoDB 4.2 : rs.hello()
```

12. Forcer le basculement du Primary vers Secondary

```
rs.stepDown()
```

Partie 3 - Résilience et tolérance aux pannes

13. Comment désigner un Arbitre ? Pourquoi ?

```
rs.addArb("hostname:port")
```

L'Arbitre participe aux élections mais **ne stocke pas de données**, permettant d'obtenir une majorité à moindre coût.

14. Configurer un Secondary avec délai de réplication (slaveDelay)

```
cfg = rs.conf()
```

```
cfg.members[i].slaveDelay = 120

rs.reconfig(cfg)
```

```
cfg.members[i].priority = 0 // requis pour secondaryDelaySecs

cfg.members[i].secondaryDelaySecs = 7200
```

Un Secondary retardé permet de conserver un historique plus ancien pour restauration.

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

Aucun nœud ne devient Primary.
Le Replica Set passe en mode **lecture seule**.

16. Critères de choix d'un nouveau Primary

MongoDB utilise :

- priorité (`priority`),
- dernière synchronisation (oplog à jour),
- disponibilité,
- latence réseau.

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

Un processus où les nœuds votent pour désigner un nouveau Primary après défaillance ou stepDown.

18. Auto-dégradation du Replica Set

La dégradation survient lorsqu'un nœud perd la majorité : un Primary se met alors **automatiquement en Secondary** pour éviter le split-brain.

19. Pourquoi un nombre impair de nœuds ?

Pour faciliter l'obtention d'une **majorité électorale**.

20. Conséquences d'une partition réseau

Un segment sans majorité devient **read-only**.

Le segment avec majorité élit un Primary.

Partie 4 — Scénarios pratiques

21. Scénario avec Primary (27017), Secondary (27018), Arbiter (27019)

Si le Primary devient injoignable :

- 27018 + 27019 = 2 votes → **élection réussie**, 27018 devient Primary.

22. Utilité d'un Secondary avec slaveDelay = 120s

Il maintient une version "ancienne" de la base.

Usages :

- récupération d'erreurs utilisateur,
- rollback manuel,
- audit.

23. Configuration pour lecture toujours à jour même en cas de bascule

`readConcern: "majority"`

`writeConcern: { w: "majority" }`

24. Option writeConcern garantissant que 2 nœuds valident l'écriture

`{ w: 2 }`
configuration statique, il faut que 2 secondaires aient envoyé l'acquitement pour que l'écriture soit validée au niveau du primaire

25. Pourquoi une donnée obsolète peut-elle être lue sur un Secondary ?

A cause du **lag de réplication**.

Pour éviter cela :

- readPreference `"primary"`,
- ou `readConcern: "majority"`.

26. Commande pour savoir quel nœud est Primary

`rs.isMaster()`

-> lire à "primary:"

```
rs0 [direct: primary] test> rs.isMaster()
{
  topologyVersion: {
    processId: ObjectId('693729438294e40971d27936'),
    counter: Long('7')
  },
  hosts: [ 'localhost:27018', 'localhost:27020' ],
  arbiters: [ 'localhost:27021' ],
  setName: 'rs0',
  setVersion: 9,
  ismaster: true,
  secondary: false,
  primary: 'localhost:27018',
  me: 'localhost:27018',
```

dans cet exemple, le primary est localhost

27. Forcer une bascule manuelle sans interruption majeure

```
rs.stepDown(60)
```

Optionnel :

```
rs.freeze(120)
```

pour empêcher ce nœud de redevenir Primary trop vite.

28. Procédure pour ajouter un Secondary en fonctionnement

1. Démarrer une instance MongoDB vide.
2. L'ajouter :

```
rs.add("hostname:port")
```

29. Retirer un nœud défectueux

```
rs.remove("hostname:port")
```

30. Configurer un Secondary caché (hidden)

```
cfg = rs.conf()

cfg.members[i].hidden = true

cfg.members[i].priority = 0

rs.reconfig(cfg)
```

Utilité : nœud réservé pour analytics, sauvegardes, etc.

31. Modifier la priorité d'un nœud pour devenir Primary préféré

```
cfg = rs.conf()

cfg.members[i].priority = 2

rs.reconfig(cfg)
```

Utile pour définir un Primary "préféré" plus performant.

32. Vérifier le délai de réplication

```
rs.printSlaveReplicationInfo()

déprécié,

rs.printSecondaryReplicationInfo()
```

33. Fonction de `rs.freeze()`

Empêche un nœud "secondary" de participer aux élections pour un temps donné. Utile pour la maintenance.

34. Redémarrer un Replica Set sans perdre la configuration

La configuration est stockée dans la base `local`.
Il suffit de redémarrer les instances MongoDB normalement.

35. Surveiller la réplication en temps réel

- Logs (`mongod.log`)

- Commande :

  `rs.printSecondaryReplicationInfo()`

Questions complémentaires

37. Qu’est-ce qu’un Arbitre et pourquoi ne stocke-t-il pas de données ?

Nœud sans stockage utilisé uniquement pour **les votes**.

Coût réduit, mais pas de réplication possible.

38. Vérifier la latence de réplication

`rs.printSecondaryReplicationInfo()`

Permet d'évaluer la santé du cluster.

39. Commande affichant le retard de réplication

La même :

`rs.printSecondaryReplicationInfo()`

Detecte les problèmes de synchronisation.

40. Différence réplication asynchrone / synchrone — que fait MongoDB ?

- Synchrone : nœuds mettent à jour en même temps.
- Asynchrone : Secondaries rattrapent via oplog → **MongoDB utilise l'asynchrone**.

  - > L'asynchronisme améliore les performances au prix d'un léger retard.

41. Modifier la configuration sans redémarrer les serveurs ?

Oui, via :

```
rs.reconfig(cfg)
```

42. Si un Secondary a plusieurs minutes de retard ?

Il risque de lire des données obsolètes et peut nécessiter un resync complet si l'oplog n'a plus l'historique.

43. Gestion des conflits de données

Les écritures n'arrivent que sur le Primary → pas de conflit.
En cas de divergence rare : le journal d'opérations (oplog) tranche.

44. Peut-on avoir plusieurs Primarys ?

Non, jamais.
MongoDB impose une **élection unique** pour éviter le split-brain (écritures concurrentes divergentes)

45. Pourquoi ne pas écrire sur un Secondary ?

Risque :

- données incohérentes,
- perte d'historique,
- violation du modèle de réplication.

46. Conséquences d’un réseau instable

- basculements fréquents
- latence accrue,
- incohérences de lecture,
- risques de perte de majorité et donc **blocage des écritures**.

Conclusion

Ce rapport présente une vision complète des mécanismes de réplication dans MongoDB, incluant le fonctionnement des Replica Sets, leur configuration, leurs comportements en cas de panne et les meilleures pratiques pour garantir une haute disponibilité.
Ces concepts sont essentiels pour comprendre l’architecture des systèmes distribués modernes.