Réplication et tolérance aux pannes avec MongoDB

Résumé:

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

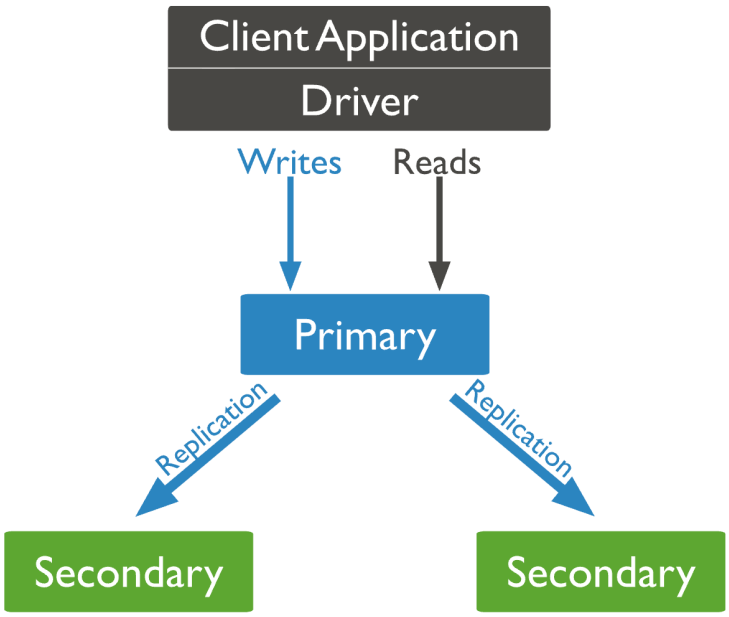
Un Replica Set est un ensemble de serveurs MongoDB qui contiennent la même base de données copiée automatiquement.

C'est comme avoir plusieurs clones du même serveur, toujours synchronisés.

Un Replica Set contient :

- 1 Primary → le serveur principal
- 1 ou plusieurs Secondaries → serveurs de secours, qui reçoivent une copie des données
- (optionnel) 1 Arbitre → participe aux votes mais ne stocke rien

L'objectif est simple :
Si un serveur tombe, un autre prend sa place automatiquement.



2. Pourquoi utilise-t-on un Replica Set ?

On utilise un Replica Set pour trois raisons principales :

a) Tolérance aux pannes

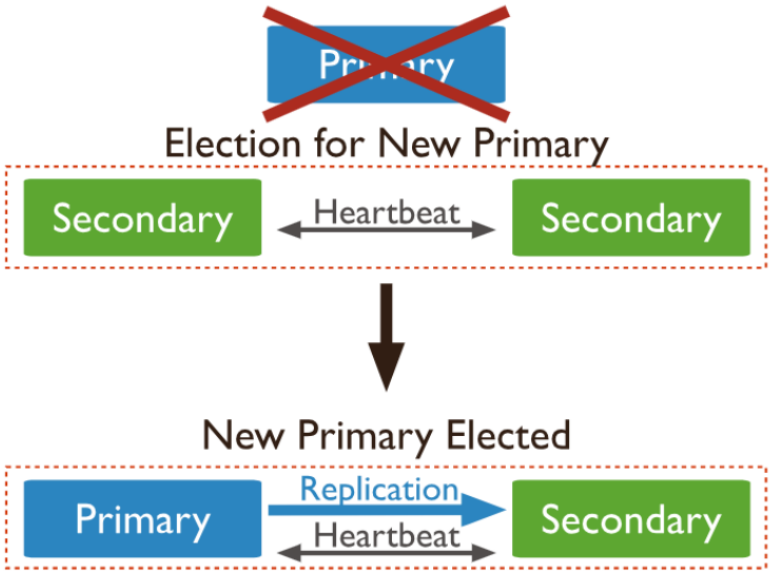
Si le Primary tombe, un Secondary devient automatiquement le nouveau Primary.
L'application continue de fonctionner sans interruption.

b) Haute disponibilité

Les données sont toujours accessibles même si une machine tombe en panne.

c) Sécurité des données

Comme les données sont copiées automatiquement, on ne perd rien si un serveur est corrompu ou cassé.



---

3. Comment fonctionne un Replica Set ?

Voici le fonctionnement en 4 étapes simples :

1/Le Primary reçoit toutes les écritures

On écrit uniquement sur le Primary pour éviter les conflits.

Exemple :Tu insères un document → il est écrit dans le Primary en premier.

2/Les Secondaries répliquent les données

Le Primary copie les données vers les Secondaries grâce à un journal de réplication (oplog).

=>La réplication est asynchrone :les Secondary peuvent avoir un peu de retard.

3/Élection automatique en cas de panne

Si le Primary ne répond plus, les autres nœuds organisent une élection pour élire un nouveau Primary.

Pour être élu, il faut :

- être à jour
- être en bonne santé
- appartenir au groupe majoritaire

4/ L'Arbitre aide à atteindre la majorité

Si on a un nombre pair de nœuds, on ajoute un arbitre pour éviter les égalités.

=>L'Arbitre vote mais ne stocke pas de données.

Partie 1 — Compréhension de base

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

   Un Replica Set est un ensemble de serveurs MongoDB connectés entre eux qui se répliquent mutuellement les données.
   Il permet d'assurer la tolérance aux pannes : si un serveur tombe, un autre peut prendre le relais.

   les nœuds échangent constamment des messages pour rester cohérents.

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

   Le Primary est le serveur principal du Replica Set.
   C'est le seul nœud qui accepte les écritures, et par défaut toutes les lectures passent aussi par lui.
   Il réplique ensuite les données vers les Secondaries.

3. Quel est le rôle essentiel des **Secondaries** ?

   Les Secondaries reçoivent les données envoyées par le Primary.
   Ils servent principalement à :

   répliquer les données,

   assurer la continuité du service si le Primary tombe en panne (élection),

   répondre aux lectures, si on l'autorise.

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

   MongoDB interdit les écritures sur un Secondary pour éviter les conflits et garantir une cohérence forte.

Si plusieurs nœuds pouvaient écrire en même temps, les données risqueraient d'être incohérentes.
Le système simplifie cela en centralisant toutes les écritures sur le Primary.

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

La cohérence forte signifie que toutes les opérations (lecture et écriture) passent par le Primary.
Ainsi, on est sûr de lire la version la plus récente des données.
C'est le mode par défaut de MongoDB.

6. Quelle est la différence entre **readPreference : "primary"** et **"secondary"** ?

- readPreference: "primary"
  → On lit toujours depuis le Primary.
  → Les données sont à jour et cohérentes.
- readPreference: "secondary"
  → On lit depuis un Secondary.
  → Cela allège le Primary, mais il existe un risque de lire une donnée pas encore mise à jour, car la réplication est asynchrone.

7. Dans quel cas pourrait-on souhaiter lire sur un Secondary malgré les risques ?

On peut choisir de lire sur un Secondary pour :répartir la charge (améliorer les performances),

préserver le Primary lorsqu'il est très sollicité,lire des données dans des contextes où une légère latence n'est pas critique.

⇨ C'est utile dans des cas non sensibles où la performance est prioritaire sur la cohérence instantanée.

Partie 2 — Commandes & configuration

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

Pour initialiser un Replica Set, on utilise la commande :rs.initiate()

```
test> rs.initiate()
{
  info2: 'no configuration specified. Using a default configuration for the set'
,
  me: '6d0358b821fa:27017',
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1764855965, i: 1 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1764855965, i: 1 })
}
```

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

On ajoute un nouveau nœud avec :rs.add("Nom:PORT")

```
rs0 [direct: primary] test> rs.add("6d0358b821fa:27019")
{
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1764856104, i: 1 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1764856104, i: 1 })
}
```

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

Pour afficher l'état en temps réel des nœuds (primary, secondary, santé, délais…), on utilise :rs.status()

```
rs0 [direct: primary] test> rs.status()
{
  set: 'rs0',
  date: ISODate('2025-12-04T14:07:45.942Z'),
  myState: 1,
  term: Long('1'),
  syncSourceHost: '',
  syncSourceId: -1,
  heartbeatIntervalMillis: Long('2000'),
  majorityVoteCount: 1,
  writeMajorityCount: 1,
  votingMembersCount: 1,
  writableVotingMembersCount: 1,
  optimes: {
    lastCommittedOpTime: { ts: Timestamp({ t: 1764857263, i: 1 }), t: Long('1')
}
```

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

On peut utiliser :rs.isMaster()

Cette commande indique si le nœud est Primary ou Secondary, et affiche aussi les membres du Replica Set.

```
rs0 [direct: primary] test> rs.isMaster()
{
  topologyVersion: {
    processId: ObjectId('69318f35be418d094a49120d'),
    counter: Long('8')
  },
  hosts: [ '6d0358b821fa:27017' ],
  passives: [ '6d0358b821fa:27019', '6d0358b821fa:27020' ],
  setName: 'rs0',
  setVersion: 3,
  ismaster: true,
  secondary: false,
  primary: '6d0358b821fa:27017',
  me: '6d0358b821fa:27017',
  electionId: ObjectId('7fffffff0000000000000001'),
  lastWrite: {
    opTime: { ts: Timestamp({ t: 1764856235, i: 1 }), t: Long('1') },
    lastWriteDate: ISODate('2025-12-04T13:50:35.000Z'),
    majorityOpTime: { ts: Timestamp({ t: 1764856235, i: 1 }), t: Long('1') },
```

12. Quelle commande permet de forcer le basculement du Primary ?
13. Comment peut-on désigner un nœud comme Arbitre ? Pourquoi le faire ?

l'arbitre est ajouté avec : rs.addArb("localhost:PORT")

Un arbitre sert à :participer au vote pour l'élection d'un Primary,sans stocker de données (plus léger).

⇨ Il est utilisé lorsqu'on a un nombre pair de nœuds pour éviter une situation où aucun groupe n'a la majorité.

14. Donnez la commande pour configurer un nœud secondaire avec un **délai de réplication**

cfg.members[1].slaveDelay = 10

Cela applique un délai de réplication au nœud secondaire ciblé.

```
rs0 [direct: primary] test> cfg.members[1].slaveDelay = 10
10
```

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 ?

S'il n'existe aucune majorité de nœuds disponibles, MongoDB ne peut pas élire un nouveau Primary. Dans ce cas, le Replica Set passe en mode dégradé :
→ aucune écriture n'est possible,
→ seules les lectures forcées sur les Secondaries restent envisageables (avec risque d'obsolescence).

16. Comment MongoDB choisit-il un nouveau Primary ? Quels critères utilise-t-il ?

Lorsqu'un Primary tombe, les nœuds restants organisent une élection.
MongoDB choisit comme nouveau Primary un nœud qui :

1. appartient au groupe majoritaire,
2. est en ligne (health = 1),
3. possède généralement la priorité la plus élevée,
4. a les données les plus à jour.

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

Une élection est le processus automatique par lequel les nœuds du Replica Set se mettent d'accord pour choisir un nouveau Primary lorsque l'ancien tombe en panne.

Elle se produit lorsque :

- le Primary ne répond plus,
- un nœud détecte une anomalie,
- une partition réseau isole un groupe majoritaire.

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

L'auto-dégradation (mode dégradé) signifie que le Replica Set cesse de fonctionner normalement, principalement :

- aucun Primary n'est élu,
- les écritures sont bloquées,
- seul un fonctionnement partiel est possible.

Cela se produit lorsque :

- aucun groupe de nœuds n'a la majorité,
- typiquement après une partition réseau ou une double panne

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

Un nombre impair permet :

- d'éviter les égalités,
- d'assurer qu'une majorité est toujours possible,
- d'empêcher le cluster de tomber en mode dégradé lors d'une panne ou partition.

⇨ C'est la raison pour laquelle MongoDB recommande parfois d'ajouter un arbitre, qui existe justement pour atteindre la majorité lors d'élections.

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

Une partition réseau coupe le Replica Set en deux groupes qui ne peuvent plus communiquer.
Conséquences :

1. Chaque groupe peut croire que l'autre est mort.
2. Chaque groupe pourrait tenter d'élire un Primary, ce qui est inacceptable (deux maîtres).
3. MongoDB empêche cela en bloquant le groupe minoritaire.
4. Seul le groupe majoritaire continue à fonctionner normalement.

Partie 4 — Scénarios pratiques

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

    Si le Primary tombe, le Secondary et l'Arbitre forment la majorité (2 sur 3).
    Ils organisent alors une élection automatique pour choisir un nouveau Primary.

    L'Arbitre ne contient pas de données, mais il participe au vote

22. Vous avez configuré un Secondary avec un `slaveDelay` de 120 secondes.
    Quelle est son utilité ? Quels usages peut-on en faire dans la vraie vie ?

    Le *slaveDelay* crée un nœud secondaire retardé volontairement dans la réplication.

    Utilités dans la vraie vie :

    - Récupérer des données après une erreur humaine (ex : suppression accidentelle il y a 2 minutes).
    - Analyser l'historique récent des modifications.
    - Éviter que les erreurs logiques se propagent immédiatement.

23. Un client exige une lecture toujours à jour, même en cas de bascule.
    Quelles options de `readConcern` et `writeConcern` recommanderiez-vous ?

    Pour garantir une donnée 100% à jour, il faut :

    - readConcern : "majority"
      → la lecture attend que la majorité ait confirmé l'écriture.
    - writeConcern : "majority"
      → l'écriture est validée seulement si au moins la majorité des nœuds l'ont enregistrée.

    C'est le seul moyen d'éviter toute lecture obsolète lors d'un basculement.

24. Dans une application critique, vous voulez garantir que l'écriture est confirmée par **au moins deux nœuds**.
    Quelle option de `writeConcern` devez-vous utiliser ?

Il faut utiliser :writeConcern: { w: 2 }

Cela garantit que l'écriture est enregistrée sur au moins deux membres du Replica Set avant d'être considérée comme validée.

25. Un étudiant a lu depuis un Secondary et récupéré une donnée obsolète. Expliquez pourquoi et comment éviter cela.

Parce que MongoDB utilise une réplication asynchrone :Donc, le Secondary n'avait pas encore reçu la dernière écriture.

Comment éviter cela ?

Lire uniquement depuis le Primary (readPreference : "primary")

Ou exiger une cohérence : readConcern : "majority"

Ou attendre que les Secondaries soient synchronisés (rs.status)

26. Montrez la commande pour vérifier quel nœud est actuellement Primary dans votre Replica Set.

rs.isMaster permet de savoir si le nœud est le primary ou un secondary

27. Expliquez comment forcer une bascule manuelle du Primary sans interruption majeure.

Arrêter temporairement le Primary, ce qui déclenche une élection :

rs.stepDown()

Cela demande au Primary d'abandonner volontairement son rôle.

28. Décrivez la procédure pour ajouter un nouveau nœud secondaire dans un Replica Set en fonctionnement.

rs.add("localhost:PORT")

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

rs.remove("localhost:PORT")

30. Comment configurer un nœud secondaire pour qu'il soit **caché** (non visible aux clients) ?

Pourquoi ferait-on cela ?
Un Secondary caché :

- ne reçoit aucune requête client,
- mais continue de répliquer les données.

Utile pour :

- opérations d'analyse,
- backups,
- traitements lourds sans impacter les clients.

Configuration (dans rs.conf) :

cfg.members[1].hidden = true

cfg.members[1].priority = 0

31. Montrez comment modifier la priorité d'un nœud afin qu'il devienne le Primary préféré.

cfg.members[1].priority = 10

32. Expliquez comment vérifier le délai de réplication d'un Secondary par rapport au Primary.

rs.status()

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

rs.freeze() empêche un nœud de participer à l'élection pendant un temps donné.
Utile quand :

- on veut empêcher un Secondary de devenir Primary (ex : maintenance),
- ou quand on veut favoriser un autre nœud plus puissant.

34. Comment redémarrer un Replica Set sans perdre la configuration ?

La configuration est stockée dans les données du Replica Set, pas en RAM.
Donc :

- arrêter les serveurs,
- les relancer normalement.

35. Expliquez comment surveiller en temps réel la réplication via les logs MongoDB ou commandes shell.

- rs.status() → état du cluster, synchronisation, retards
- rs.config() → configuration actuelle
- messages de réplication affichés dans les logs du terminal

Questions complémentaires

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

Un Arbitre est un nœud spécial du Replica Set qui ne stocke aucune donnée, mais participe uniquement au vote lors de l'élection d'un Primary.

Il ne stocke pas de données pour rester léger, car son rôle est uniquement de stabiliser les élections, surtout lorsqu'on a un nombre pair de nœuds.

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

    On vérifie la latence via :rs.status()

    Cette commande affiche notamment :

    - l'état de chaque nœud,
    - la dernière opération répliquée (optime),
    - le décalage temporel entre le Primary et les Secondaries.

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

    La commande utilisée est :rs.status()

    C'est là que MongoDB montre :

    - le retard de réplication,
    - les timestamps optime du Primary et des Secondaries.

40. Quelle est la différence entre la réplication **asynchrone** et **synchrone** ? Quel type utilise MongoDB ?

    Réplication synchrone :

    - Chaque écriture doit être confirmée par tous les nœuds avant de revenir au client.
    - Très cohérente, mais lente.

    Réplication asynchrone :

    - Le Primary confirme l'écriture **avant** que les Secondaries ne l'aient répliquée.
    - Plus rapide, mais un Secondary peut être en retard → risque de lire une donnée obsolète.

    => MongoDB utilise une réplication **asynchrone**

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

    MongoDB permet de **modifier dynamiquement** la configuration avec :

```
cfg = rs.conf()
(cfg modifications…)
rs.reconfig(cfg)
```

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

    S'il est très en retard :

    - il peut fournir des données obsolètes si on lit dessus,
    - certains mécanismes de synchronisation peuvent prendre plus de temps,
    - en cas de bascule, il ne pourra pas devenir Primary car ses données ne sont pas à jour.

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

MongoDB évite les conflits grâce à sa règle fondamentale :

- Toutes les écritures se font uniquement sur le Primary.
- Les Secondaries ne modifient jamais directement les données.

⇨ Donc, il n'y a pas de conflits d'écriture possibles.

44. Est-il possible d'avoir plusieurs Primarys simultanément dans un Replica Set ? Pourquoi ?

Non, ce n'est jamais possible.

MongoDB empêche cela grâce au mécanisme de majorité :

Si aucun groupe n'a la majorité, MongoDB bloque les élections pour éviter l'apparition de deux Primarys.

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

Parce qu'un Secondary ne peut pas accepter les écritures : MongoDB les rejette systématiquement.

Même en lecture préférée secondary, ce nœud reste en lecture seule, car autoriser l'écriture provoquerait des conflits et briserait la cohérence.

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

Un réseau instable peut provoquer :

1. Des partitions réseau → le cluster se divise en groupes.
2. L'impossibilité d'élire un Primary s'il n'y a pas de majorité → le système passe en mode dégradé.
3. Risque de retard de réplication sur les Secondaries.
4. Basculement fréquent si le Primary devient régulièrement injoignable.