Réplication et reprise sur panne (Partie 2)

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, permettant la **redondance**, la **tolérance aux pannes** et la **haute disponibilité** grâce à la réplication automatique.

---

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

Le Primary est le nœud qui reçoit toutes les **écritures** et les **lectures par défaut**.
Il réplique ces données vers les Secondaries.

---

3. Quel est le rôle essentiel des Secondaries ?

Les Secondaries répliquent les données du Primary et peuvent servir :

- de nœuds de secours en cas de panne ;
- de nœuds pour des lectures (si configuré).

---

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

Parce que cela créerait des **conflits de données**.
MongoDB utilise une réplication **asynchrone primaire→secondaire**, donc seuls les Primary peuvent écrire pour garantir la cohérence.

---

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

C’est le fait que **toutes les lectures se font sur le Primary**, garantissant des données **toujours à jour**.

---

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

- **Primary** : lecture toujours à jour, cohérente.
- **secondary** : lecture possible sur les nœuds secondaires, mais risque de données **en retard**.

---

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

- Pour **équilibrer la charge** (read scaling)
- Pour des **rapports**, **analytics**, tâches non critiques
- Pour réduire la latence géographique

---

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

```
rs.initiate()
```

---

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

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

---

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

```
rs.status()
```

---

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

En utilisant :

```
rs.status()
```

---

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

```
rs.stepDown()
```

Partie 3 — Résilience et tolérance aux pannes

---

13. Comment peut-on désigner un nœud comme arbitre ? Pourquoi le faire ?

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

Un arbitre participe aux élections mais **ne stocke pas de données**.
Il est utile pour **obtenir une majorité** à coût réduit.

---

14. Donnez la commande pour configurer un nœud secondaire avec un délai de réplication (`slaveDelay`).

```
cfg = rs.conf()
cfg.members[1].slaveDelay = 120
rs.reconfig(cfg)
```

---

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

Aucun nouveau Primary n'est élu.
Le Replica Set passe en **lecture seule** jusqu'à récupération de la majorité.

---

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

Critères :

- Priorité (`priority`) la plus élevée
- Nœud à jour (petit oplog lag)
- Réplication fonctionnelle
- Vote des membres (majorité)

---

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

Processus automatique qui choisit un nouveau Primary lorsqu'un Primary devient indisponible.

---

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

Quand les nœuds détectent qu'ils ne sont plus majoritaires →
Ils se mettent en **SECONDARY** et refusent les écritures.

---

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

Pour faciliter l'**obtention de la majorité**, condition pour élire un Primary.

---

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

- Le groupe minoritaire perd le Primary

- Éventuel split-brain évité (pas 2 Primary)
- Blocage des écritures dans la partie minoritaire

****21. Vous avez 3 nœuds : 27017 (Primary), 27018 (Secondary) et 27019 (Arbitre).**

Que se passe-t-il si le Primary devient injoignable ?**
27018 + 27019 forment la majorité →
Le Secondary devient **Primary**.

****22. Vous avez configuré un Secondary avec un slaveDelay de 120 secondes.**

Quelle est son utilité ?**
Il maintient une copie **retardée** de 2 min.
Utilité :

- récupération en cas d'erreur utilisateur (delete accidentel)
- audit
- analyses historiques

****23. Un client exige une lecture toujours à jour, même en cas de bascule.**

Quelles options recommander ?**

- `readConcern: "majority"`
- `writeConcern: { w: "majority" }`

****24. Garantir que l'écriture est confirmée par au moins deux nœuds.**

Quel writeConcern utiliser ?**

```
{ w: 2 }
```

---

25. Un étudiant lit depuis un Secondary et récupère une donnée obsolète. Pourquoi ? Comment éviter cela ?

Parce que la réplication est **asynchrone** → retard possible.
Éviter :

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

---

26. Commande pour vérifier quel nœud est actuellement Primary ?

```
rs.status()
```

---

27. Comment forcer une bascule manuelle du Primary sans interruption majeure ?

Sur le Primary :

```
rs.stepDown()
```

---

28. Procédure pour ajouter un nouveau nœud secondaire à chaud :

1. Ajouter dans la conf :

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

2. Vérifier la réplication :

```
rs.status()
```

---

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

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

---

30. Comment configurer un nœud secondaire pour qu'il soit caché ? Pourquoi ?

Modifier la config :

```
cfg = rs.conf()
cfg.members[1].hidden = true
cfg.members[1].priority = 0
rs.reconfig(cfg)
```

Usage : nœuds dédiés à l'analyse, sauvegarde, reporting.

---

31. Modifier la priorité d'un nœud pour qu'il devienne Primary préféré :

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

---

32. Vérifier le délai de réplication d'un Secondary :

Avec :

```
rs.printSlaveReplicationInfo()
```

Questions complémentaires

---

33. Que fait `rs.freeze()` ? Pourquoi est-elle utile ?

Elle empêche un Secondary de devenir Primary pendant une durée donnée.
Utile pour maintenance.

---

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

La configuration est stockée dans les données du Replica Set.
Il suffit de **redémarrer les mongod** : rien n'est perdu.

---

35. Comment surveiller la réplication en temps réel ?

- Logs du serveur
- Commandes :

```
rs.printSlaveReplicationInfo()
rs.status()
```

---

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

Un Arbitre vote aux élections mais n'a pas de données → coût faible.
Il sert uniquement pour la **majorité**.

---

38. Vérifier la latence de réplication entre Primary et Secondaries ?

Avec :

```
rs.printSlaveReplicationInfo()
```

---

39. Commande affichant le retard de réplication des secondaries ?

```
rs.printSlaveReplicationInfo()
```

---

40. Différence entre réplication synchrone et asynchrone ? Quel type MongoDB utilise-t-il ?

- Synchrone : tous les nœuds doivent écrire avant validation.
- Asynchrone : propagation différée.
  MongoDB utilise **asynchrone**, sauf si `writeConcern` impose une attente.

---

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

Oui :

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

---

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

Il se **resynchronise** si le Primary a encore les données dans l'oplog.
Sinon, il doit faire un **initial sync** complet.

---

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

Le Primary est la **source de vérité**.
Les Secondaries appliquent les opérations dans le même ordre (oplog).

---

44. Peut-on avoir plusieurs Primary simultanément ? Pourquoi ?

Non.
Le protocole de vote empêche le **split-brain**.

---

45. Pourquoi ne pas écrire sur un Secondary même en readPreference "secondary" ?

Parce que MongoDB **n’autorise jamais** les écritures sur les secondaries → risque de divergence.

---

46. Conséquences d’un réseau instable sur un Replica Set ?

- Chutes de Primary
- Élections fréquentes
- Risque de lectures obsolètes
- Blocage des écritures (pas de majorité)