Bases de Données NoSQL - TP2

Prénom: Abdelillah

Nom: SELHI

N°Étudiant: 12206456

Groupe: INFOA3

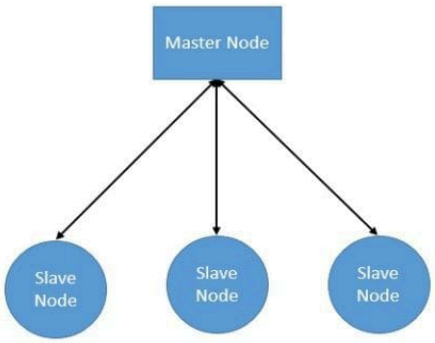
1. C’est quoi MongoDB?

MongoDB est un système de gestion de base de données **NoSQL orienté document**. Il stocke les données sous forme de documents BSON (Binary JSON), qui sont des structures de données flexibles (similaires à des objets JSON) regroupées dans des "collections". **Contrairement aux bases de données relationnelles (SQL)** qui utilisent des tables, le modèle orienté document de MongoDB permet des schémas dynamiques, ce qui facilite l'évolution des applications.

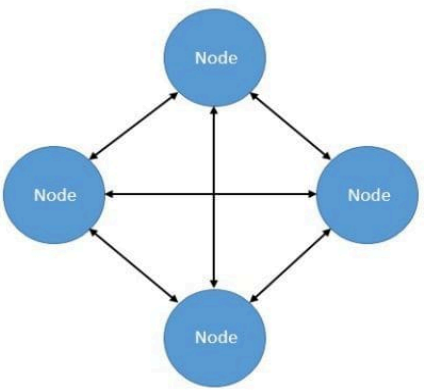
2. Architectures de Base de Données

Architecture Maître-Esclave vs Multi-nœuds

Caractéristique	**Maître-Esclave (Master-Slave)**	**Multi-Nœuds (Peer-to-Peer model)**
Rôles des Nœuds	Un seul nœud principal (Maître) gère les écritures. Les autres sont des Esclaves (Slaves) qui gèrent les lectures et répliquent les données.	Tous les nœuds peuvent potentiellement accepter des écritures.
Tolérance aux Pannes	Si le Maître tombe, le système d'écriture est interrompu jusqu'à l'élection d'un nouveau Maître.	Plus résilient. Si un nœud tombe, d'autres peuvent continuer à accepter des écritures.
Montée en Charge	Les lectures peuvent être distribuées, mais les écritures sont limitées par la capacité du Maître unique.	Meilleure distribution de la charge d'écriture sur tous les nœuds.



Master-Slave Model



Peer-to-Peer Model

Quel modèle utilise MongoDB?

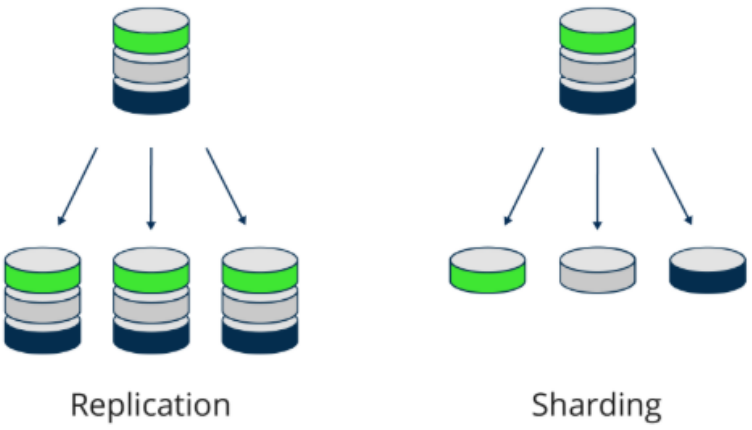
MongoDB utilise un modèle basé sur la **Réplication par Set de Répliques** (Replica Set). Un Replica Set est composé de plusieurs instances :

- **Primaire (équivalent du Maître) :** Un seul nœud accepte toutes les opérations d'écriture.
- **Secondaires (équivalents des Esclaves) :** Répliquent les données du Primaire et peuvent gérer les requêtes de lecture.
- **Élection :** Si le nœud Primaire tombe, les Secondaires élisent automatiquement un nouveau Primaire.

3. Stratégies de Montée en Charge

Réplication vs. Distribution (Sharding)

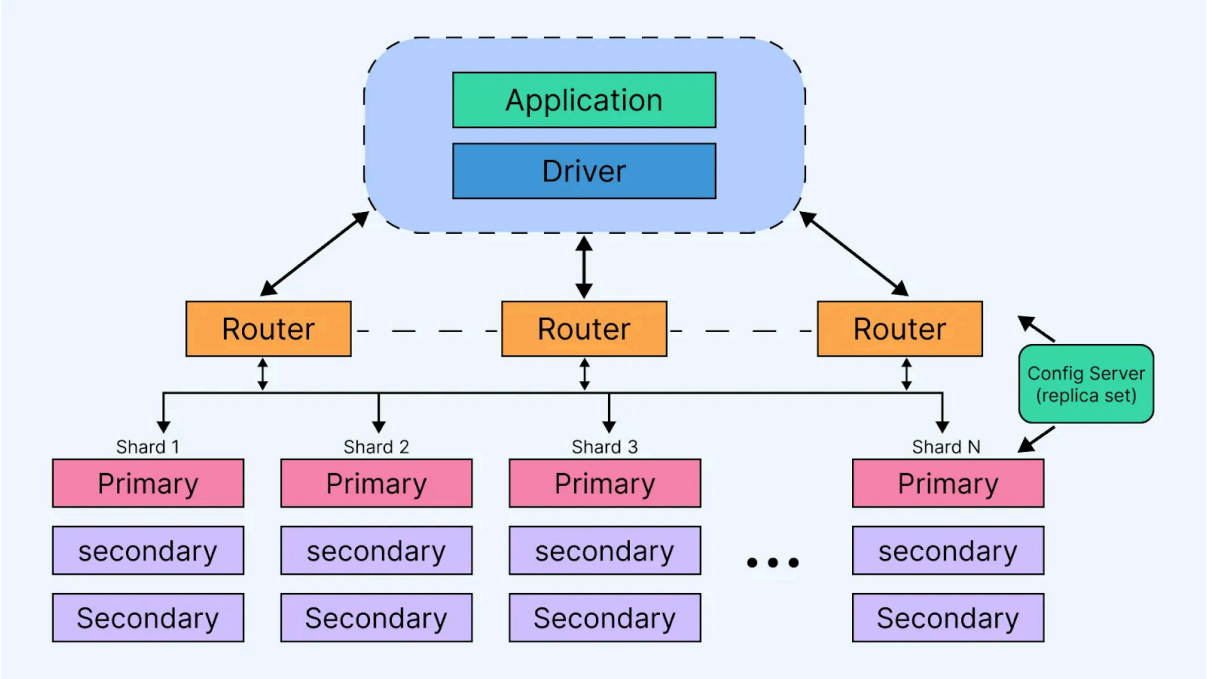
Stratégie	Définition	Objectif	Risques/Défis
Réplication	Copie des mêmes données sur plusieurs serveurs (nœuds).	**Haute Disponibilité** et **Tolérance aux Pannes** ; **Équilibrage de Charge** (distribution des lectures).	Coût en espace de stockage (duplication) ; Latence potentielle de la réplication (consistance éventuelle).
Distribution (Sharding)	Partitionnement des données en sous-ensembles (shards) stockés sur des serveurs distincts.	**Augmentation de la Capacité de Stockage** et **Amélioration des Performances d'Écriture/Lecture**.	Complexité de l'administration et de la distribution des données ; Jonctions de données plus complexes.



Quel modèle utilise MongoDB?

MongoDB utilise les deux stratégies pour la montée en charge :

1. **Réplication** via les **Replica Sets** (pour la haute disponibilité).
2. **Distribution** via le **Sharding** (pour l'évolutivité horizontale de la capacité et de la charge). Chaque "shard" est lui-même un Replica Set.

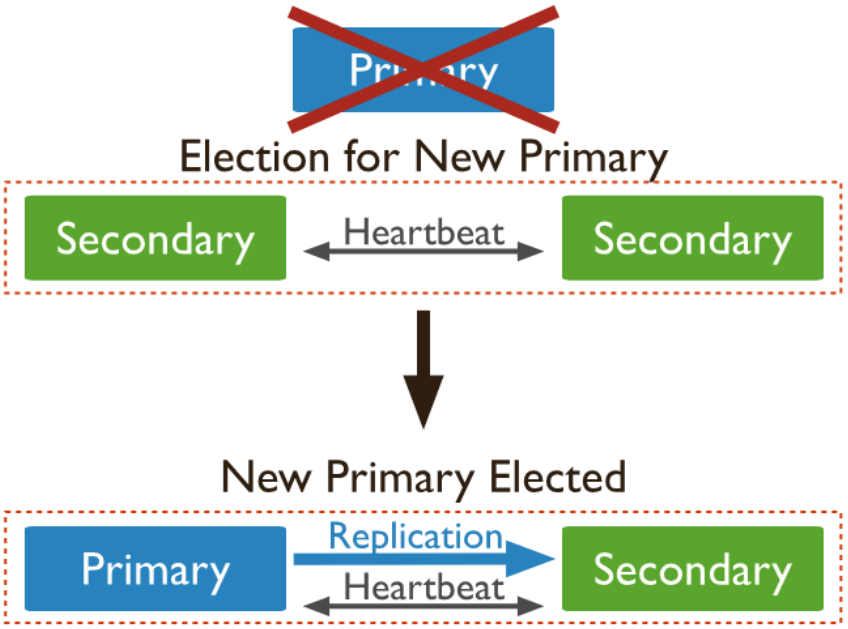


4. Tolérance aux Pannes

SI le maître tombe? (procédure d'élection)

Dans un Replica Set MongoDB, si le nœud **Primaire** tombe, les nœuds **Secondaires** initient immédiatement une **procédure d'élection** pour choisir un nouveau Primaire.

1. **Détection :** Les Secondaires détectent la panne du Primaire (absence de "heartbeats").
2. **Vote :** Les Secondaires éligibles se portent candidats et votent.
3. **Élection :** Le nœud qui obtient la majorité des voix devient le nouveau Primaire et commence à accepter les écritures.
4. **Basculement (Failover) :** L'application cliente est redirigée vers le nouveau Primaire.



5. Configuration de la Réplication

POUR configurer la réplication sur mongodb ?

La configuration implique de démarrer les instances de MongoDB avec l'option *Replica Set* activée, puis d'initialiser le set.

Un replica set MongoDB est un groupe de processus **mongod** qui maintiennent la même copie de données, avec un nœud primaire (master) qui accepte les écritures et un ou plusieurs nœuds secondaires (slave) qui répliquent ces données.

a. Démarrer les 3 nœuds MongoDB avec Docker

Paramètres nécessaires : Nom du Replica Set (**--replSet rs0**), Ports d'écoute (**27018, 27019, 27020**) et Répertoires de données (**disque1, disque2, disque3**).

Exemple de commande Docker:

```
abdelillah@abdelillah-HP-250-G9:~$ docker network create mongo-net
0ac1febb6aa3bc87bafffdf0b23c7dd3b47c8362d24ded971031c4c61eb0756c
abdelillah@abdelillah-HP-250-G9:~$ docker run -d --name mongo1 --network mon
go-net \
-p 27018:27017 \
-v $(pwd)/disque1:/data/db \
mongo --replSet rs0 --port 27017
47f1891323be6df303e277aaf3a321ef9d153ee573ef0348fe616c520676fcd5
abdelillah@abdelillah-HP-250-G9:~$ docker run -d --name mongo2 --network mon
go-net \
-p 27019:27017 \
-v $(pwd)/disque2:/data/db \
mongo --replSet rs0 --port 27017
90ee2d41d14ba01ad656abfd43e3d00505333403795ed67c53dafa3f6b73e73c
abdelillah@abdelillah-HP-250-G9:~$ docker run -d --name mongo3 --network mon
go-net \
-p 27020:27017 \
-v $(pwd)/disque3:/data/db \
mongo --replSet rs0 --port 27017
bfd2a65bbfd7281b32cccfa01318781df35dc7b17788996b9f13f33b05f03f64
```

b. Initialiser le Replica Set

- **Connexion au premier nœud :**
  docker exec -it mongo1 mongosh
- **Initialisation et ajout des autres nœuds :**

  rs.intiate()

  rs.add(“mongo2:27017”)

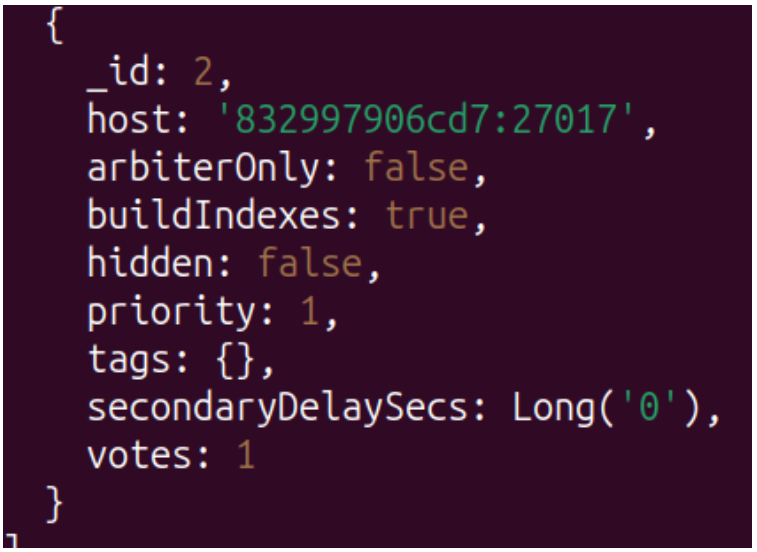
  rs.add(“mongo3:27017”)

6. Analyse de Configuration

Explique les champs… rs.config()

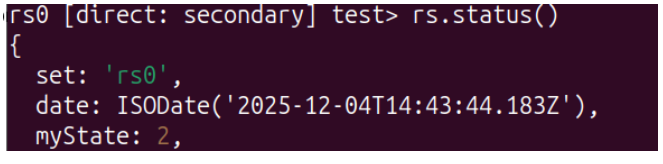
La commande rs.config() retourne le document de configuration du Replica Set. Les champs clés dans la section members sont :

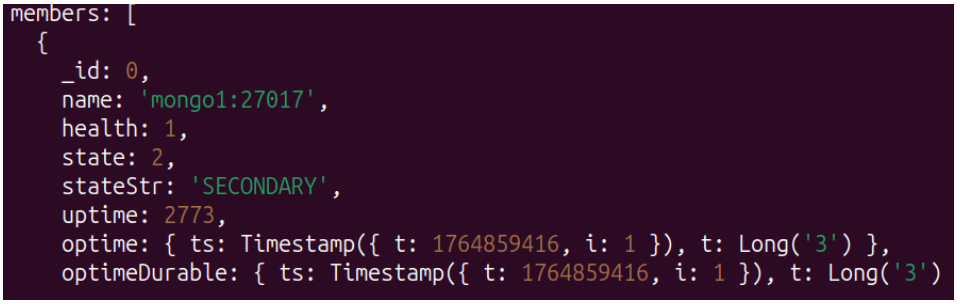
- **_id** : ID unique du membre au sein du set.
- **host** : Adresse réseau du membre (hostname:port).
- **priority** : Priorité du membre dans l'élection (plus élevée = plus de chance d'être Primaire).
- **votes** : Indique si le membre a une voix lors d'une élection (généralement 1).
- **arbiterOnly** : Si true, le membre participe au vote, mais ne stocke pas de données.
- **hidden** : Si true, le membre est invisible aux applications clientes.



Explique les champs… rs.config()

Cette commande fournit un aperçu de l'état actuel et opérationnel de tous les membres du Replica Set. Les champs clés comprennent :





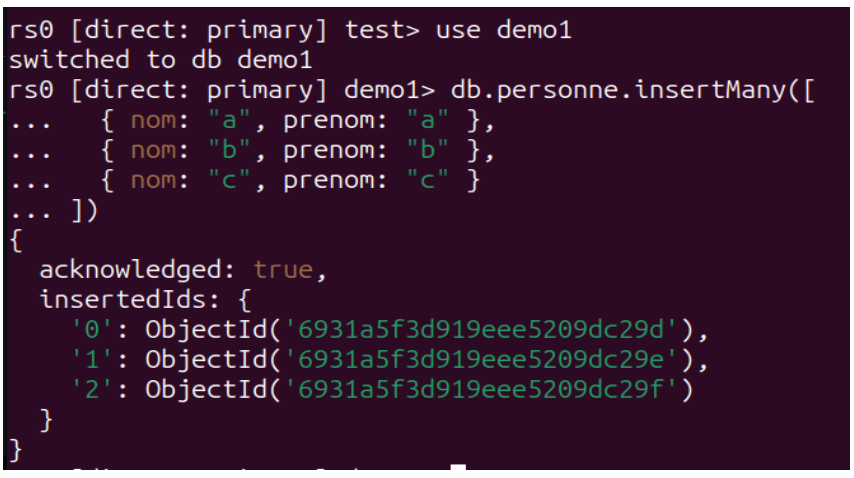
- **set**: Le nom du Replica Set.
- **myState:** L'état actuel du nœud à partir duquel la commande a été exécutée.
- **members:** Un tableau qui détaille l'état de chaque membre du Replica Set.
    - **name: L**'adresse réseau du membre.
    - **stateStr:** L'état du membre (PRIMARY / SECONDARY)
    - **health:** Un indicateur de santé.
    - **uptime:** Le temps écoulé (en secondes) depuis le démarrage du processus.
    - **optimeDate:** Le timestamp de la dernière opération de réplication appliquée par ce membre. Ce champ est crucial pour calculer le lag de réplication
    - **electionTime:**La date de la dernière élection remportée.

7. Démonstration de la réplication

Dans cette section, on illustre le fonctionnement du **Replica Set** en effectuant une écriture sur le nœud primaire **mongo1**, puis en vérifiant que les données sont bien répliquées sur les nœuds secondaires **mongo2** et **mongo3**.

1. **Connexion au nœud primaire**
    - Connexion au conteneur primaire :
      docker exec -it mongo1 mongosh
    - Sélection de la base de données :
      use demo1
2. **Création de la collection et insertion de documents**
    - Création de la collection personne et insertion de trois documents :

```
db.personne.insertMany([
  { nom: "a", prenom: "a" },
  { nom: "b", prenom: "b" },
  { nom: "c", prenom: "c" }
])
```



3. **Vérification sur le nœud secondaire mongo2**
    - Connexion : **docker exec -it mongo2 mongosh**
    - Passage en mode lecture sur un secondaire (si nécessaire): **rs.secondaryOk()**
    - Sélection de la base : **use demo1**
    - Vérification des données : **db.personne.find()**

```
rs0 [direct: secondary] demo1> use demo1
already on db demo1
rs0 [direct: secondary] demo1> db.personne.find()
[
  { _id: ObjectId('6931a5f3d919eee5209dc29d'), nom: 'a', prenom: 'a' },
  { _id: ObjectId('6931a5f3d919eee5209dc29e'), nom: 'b', prenom: 'b' },
  { _id: ObjectId('6931a5f3d919eee5209dc29f'), nom: 'c', prenom: 'c' }
]
rs0 [direct: secondary] demo1>
```

4. **Vérification sur le nœud secondaire mongo3**
    - Connexion : **docker exec -it mongo3 mongosh**
    - Passage en mode lecture (si nécessaire) : **rs.secondaryOk()**
    - Sélection de la base : **use demo1**
    - Vérification des données : **db.personne.find()**

```
rs0 [direct: secondary] test> use demo1
switched to db demo1
rs0 [direct: secondary] demo1> db.personne.find()
[
  { _id: ObjectId('6931a5f3d919eee5209dc29d'), nom: 'a', prenom: 'a' },
  { _id: ObjectId('6931a5f3d919eee5209dc29e'), nom: 'b', prenom: 'b' },
  { _id: ObjectId('6931a5f3d919eee5209dc29f'), nom: 'c', prenom: 'c' }
]
rs0 [direct: secondary] demo1>
```

Explication:

- l'insertion des documents dans la collection **personne** est effectuée uniquement depuis le nœud **primaire (mongo 1)** du replica set.
- Dès que le primaire confirme l'écriture, il enregistre l'opération dans son journal des opérations. Les nœuds secondaires (**mongo 2** et **mongo 3**) lisent en continu ce journal, appliquent les mêmes opérations localement, puis mettent à jour leurs propres données pour rester synchronisés avec le primaire.
- Ainsi, lorsque l'on exécute la commande find() sur les secondaires, on observe les mêmes documents que ceux insérés sur le primaire.

Questions / Réponses

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

Un Replica Set est un groupe de processus **mongod** qui maintiennent le même jeu de données. Il est composé d'un nœud **Primaire** unique et d'un ou plusieurs nœuds **Secondaires**. Il fournit une haute disponibilité et une redondance des données.

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

Le nœud **Primaire** est le seul membre du Replica Set qui accepte toutes les opérations d'écriture (inserts, updates, deletes). Il enregistre ces opérations dans son journal des opérations pour la réplication. Il peut également gérer les opérations de lecture.

3. Quel est le rôle essentiel des Secondaries?

Les nœuds **Secondaires** répliquent en continu les données du Primaire en lisant son journal des opérations. Leur rôle essentiel est de fournir une redondance des données et d'assurer la tolérance aux pannes en participant à l'élection d'un nouveau Primaire si le nœud actuel tombe. Ils peuvent également gérer les requêtes de lecture.

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

MongoDB applique la règle qu'il ne peut y avoir qu'un seul Primaire à la fois pour garantir la **cohérence forte** des écritures. Autoriser les écritures sur un Secondaire créerait un risque de conflits de données.

5. Qu'est-ce qu'une cohérence forte dans le contexte MongoDB ?

La cohérence forte garantit qu'une opération de lecture renvoie la donnée la plus récemment écrite, quelle que soit l'instance lue. Dans un Replica Set, lire sur le Primaire (readPreference: "primary") garantit cette cohérence.

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

- **"primary"**: Toutes les opérations de lecture sont dirigées vers le nœud Primaire. C'est le réglage par défaut et il garantit les lectures les plus à jour (cohérence forte).
- **"secondary"**: Les opérations de lecture sont dirigées vers les nœuds Secondaires. Cela permet de répartir la charge de lecture et peut réduire la latence pour les clients géographiquement proches, au prix d'une potentielle lecture de données obsolètes.

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

On souhaite lire sur un Secondaire principalement pour :

- **Équilibrer la charge de lecture** : Soulager le Primaire, surtout pour les applications à forte intensité de lecture.
- **Réduire la latence** : Si le Secondaire est plus proche géographiquement de l'application cliente.

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

La commande utilisée est rs.initiate(), exécutée depuis l'interface mongosh du nœud destiné à être le primaire initial.

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

Pour ajouter un membre, utilisez la commande rs.add("<hostname>:<port>").

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

La commande rs.status() permet d'afficher l'état opérationnel actuel de l'ensemble des membres du Replica Set (rôles, santé, etc.).

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

Le rôle est indiqué dans la sortie de rs.status(), via le champ **stateStr** du membre concerné. Il peut être PRIMARY, SECONDARY, ou ARBITER.

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

La commande rs.stepDown() exécutée sur le nœud Primaire force celui-ci à se démettre de son rôle, déclenchant ainsi une élection parmi les Secondaires pour choisir un nouveau Primaire.

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

Un nœud est désigné comme Arbitre en utilisant la commande rs.addArbiter("<hostname>:<port>") ou en ajoutant un membre avec l'option arbiterOnly: true dans la configuration. Le faire permet de garantir une majorité électorale.

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

Pour configurer un délai, on utilise rs.reconfig(<config_document>), en modifiant le document de configuration et en y ajoutant le champ slaveDelay (en secondes) pour le membre secondaire souhaité.

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

Si le Primaire tombe et que le nombre de membres restants et joignables ne constitue pas une majorité stricte (>50%) des membres votants, **aucune élection ne peut avoir lieu**. Le Replica Set devient en lecture seule, et **aucune nouvelle écriture n'est possible** tant qu'une majorité n'est pas restaurée.

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

Un nouveau Primaire est choisi par un processus d'**élection**. Les critères pour l'emporter sont:

- **Disponibilité**: Le nœud doit être en bonne santé et joignable.
- **Priorité**: Les nœuds ayant une priorité plus élevée (valeur > 0) ont plus de chances d'être élus.
- **État des données** : Le nœud candidat doit posséder les données les plus récentes.

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

Une élection est le processus par lequel les membres Secondaires d'un Replica Set choisissent de manière consensuelle un nouveau nœud Primaire pour reprendre les opérations d'écriture après la défaillance du Primaire actuel.

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

L'auto-dégradation survient lorsque le Replica Set perd sa majorité de membres votants. Il passe alors en mode dégradé ou lecture seule, car il ne peut plus garantir la cohérence des écritures ni élire un nouveau Primaire en cas de besoin.

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

Avoir un nombre impair de nœuds est conseillé pour garantir une majorité simple lors des élections. Pour un set de N membres, la majorité est N/2+1.

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

Une partition réseau peut diviser le Replica Set en deux groupes. Le groupe qui conserve la **majorité** des nœuds votants peut continuer à fonctionner et à élire/maintenir un Primaire. Le groupe minoritaire ne peut pas maintenir le Primaire.

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

Si le Primaire (27017) tombe, il reste deux nœuds votants : le Secondaire (27018) et l'Arbitre (27019). Ces deux nœuds constituent une majorité (2/3). Ils vont immédiatement lancer une élection, et le Secondaire (27018), étant le seul à stocker des données et à être éligible au rôle de Primaire, sera élu comme nouveau Primaire.

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 ?

L'slaveDelay force le Secondaire à appliquer les opérations de réplication avec un retard minimal spécifié (ici, 120 secondes). Son utilité principale est de servir de **sauvegarde de secours** pour récupérer des données après une erreur humaine. Par exemple, si une suppression accidentelle est faite sur le Primaire, le Secondaire retardé offre une fenêtre de 120 secondes pour l'arrêter avant qu'il ne réplique l'erreur, permettant de restaurer les données à partir de ce nœud.

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

- **writeConcern** : { w: "majority" } pour garantir que l'écriture est appliquée sur la majorité des nœuds avant de confirmer au client.
- **readConcern** : { level: "majority" } pour garantir que la lecture renvoie uniquement les données qui ont été appliquées sur la majorité des nœuds.

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 ?

Vous devez utiliser writeConcern: { w: 2 }. Cela garantit que l'opération d'écriture ne renverra le succès au client qu'après avoir été appliquée par le Primaire et au moins un Secondaire.

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

- **Pourquoi :** L'étudiant a lu depuis un Secondaire. Le Secondaire n'avait pas encore eu le temps d'appliquer la dernière opération d'écriture du Primaire (lag de réplication), ce qui a conduit à une lecture obsolète.
- **Comment éviter :** Utiliser readConcern: { level: "majority" } ou passer la readPreference à "primary" (pour lire uniquement sur le nœud le plus à jour).

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

La commande la plus simple est d'exécuter rs.isMaster() depuis n'importe quel membre du set. La réponse contient le champ "ismaster" : true si vous êtes sur le Primary, ou le champ "primary" : "<hostname>:<port>" qui indique l'adresse du Primary.

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

La méthode recommandée est d'utiliser la commande rs.stepDown() sur le nœud Primaire actuel. Cela force le Primaire à se déclasser immédiatement en Secondaire. Les autres membres lancent alors une élection pour choisir un nouveau Primaire, ce qui minimise l'interruption côté application.

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

1. **Préparation :** Démarrer le nouveau processus mongod avec le même nom de Replica Set (--replSet <rsName>).
2. **Connexion :** Se connecter au Primaire actuel en utilisant mongosh.
3. **Ajout :** Exécuter la commande rs.add("<nouveau_nœud>:<port>").
   Le nouveau nœud est ajouté à la configuration et commence immédiatement la synchronisation des données.

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

Pour retirer un nœud, connectez-vous au Primaire et utilisez la commande rs.remove("<nœud_à_retirer>:<port>").

30. Comment configurer un nœud secondaire pour qu'il soit caché (non visible aux clients) ? Pourquoi ferait-on cela ?

Lors de la configuration du Replica Set, on modifie le document de configuration avec rs.reconfig() en ajoutant ou en modifiant le champ hidden: true.

Utilité : Un nœud caché participe à la réplication et aux élections, mais il est ignoré par les *drivers* clients lors de la distribution des lectures, ce qui le réserve à des usages administratifs.

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

Pour modifier la priorité, on utilise rs.reconfig() :

1. Obtenir la configuration : cfg = rs.config()
2. Modifier la priorité du membre ciblé (par exemple, membre _id: 0).
3. Appliquer la nouvelle configuration : rs.reconfig(cfg)

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

Le délai de réplication est vérifié en comparant l'optimeDate du Primaire avec l'optimeDate de chaque Secondaire. La différence de temps entre ces deux champs indique le délai.

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

La commande rs.freeze(<seconds>) empêche le membre sur lequel elle est exécutée de participer aux élections pendant le nombre de secondes spécifié. Elle est utile dans les scénarios de maintenance ou d'administration, par exemple pour isoler temporairement un nœud ou pour s'assurer qu'il ne prendra pas le rôle de Primaire pendant une intervention.

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

La configuration est stockée de manière persistante sur les membres du set. Pour redémarrer un Replica Set sans perdre la configuration, redémarrez chaque processus mongod un par un, en veillant à ce que le Primaire actuel ne soit pas redémarré en premier (sauf s'il est isolé ou déclassé manuellement) pour éviter une interruption des écritures.

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

- **Commandes Shell :** rs.status() fournit un instantané de l'état, et rs.printReplicationInfo() donne un aperçu du délai de réplication.
- **Logs :** Les logs MongoDB (mongod.log) contiennent des informations détaillées sur l'activité de réplication.

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 a un rôle de vote dans les élections mais ne stocke **aucune donnée** et ne peut jamais devenir Primaire.

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

La latence est vérifiée en comparant l'optimeDate du Primaire avec l'optimeDate de chaque Secondaire.

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

La commande rs.printReplicationInfo() est souvent utilisée pour un résumé rapide du "lag" de réplication.

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

- **Réplication Synchrone :** L'opération d'écriture n'est considérée comme réussie et confirmée au client qu'une fois que la donnée a été appliquée sur **tous** les membres spécifiés.
- **Réplication Asynchrone :** L'opération d'écriture est confirmée au client après avoir été appliquée sur le Primaire, les Secondaires se synchronisant plus tard.
- **MongoDB** utilise par défaut la **réplication asynchrone**.

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

Oui, la configuration peut être modifiée à chaud sans redémarrer les serveurs en utilisant la commande rs.reconfig(cfg) sur le Primaire.

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

Si un Secondaire est en retard de plusieurs minutes, il pourrait ne pas être éligible pour devenir Primaire lors d'une élection. De plus, son journal risque de déborder.

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

MongoDB utilise un mécanisme de réplication basé sur le **journal des opérations**. Les Secondaires appliquent les opérations dans le même ordre que le Primaire. **Il n'y a pas de conflit de données à gérer**.

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

Non, il n'est **pas possible** d'avoir plusieurs Primaires simultanément et MongoDB est conçu pour le prévenir grâce au mécanisme d'élection par majorité.

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

Les Secondaires ne sont pas autorisés à recevoir des écritures. Les Secondaires existent uniquement pour la redondance et la distribution des lectures.

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

Un réseau instable peut entraîner :

- **Augmentation du lag de réplication**.
- **Élections fréquentes ou inutiles**.
- **Perte de majorité temporaire**.