Bases de données NoSQL

Réplication

Nom: HAMRANI
Prénom: Amziane

I. Rappel:

1. Qu'est-ce qu'une base de données NoSQL ?

Les bases de données NoSQL (Not Only SQL) désignent une famille de systèmes de gestion de données qui se distinguent des SGBD relationnels classiques par leur modèle de données, leurs mécanismes de stockage et leur scalabilité.
Elles sont apparues pour répondre aux limitations des bases relationnelles face à la croissance massive des données, à la distribution géographique des systèmes et aux exigences de disponibilité des applications modernes.

Contrairement au modèle relationnel, NoSQL ne s'appuie pas sur un schéma fixe et privilégie souvent un schéma flexible, permettant de stocker des données hétérogènes et évolutives. Les SGBD NoSQL sont généralement conçus pour offrir :

- Scalabilité horizontale : ajout de nœuds pour absorber la charge.
- Tolérance aux pannes renforcée grâce à la réplication.
- Performances élevées sur des volumes importants de données.
- Modèles de données variés, adaptés à différents usages : documents, colonnes, graphes, clés-valeurs.

2. MongoDB : rôle et positionnement dans l'écosystème NoSQL

MongoDB est un système NoSQL orienté documents. Il stocke les données sous forme de documents BSON (Binary JSON), une représentation binaire de JSON permettant d'intégrer des types supplémentaires et d'optimiser le stockage.
Ses principaux objectifs sont : proposer un modèle flexible, riche et proche des structures manipulées dans les applications, offrir des performances élevées pour les opérations de lecture/écriture, assurer une forte disponibilité grâce à la **réplication**.
MongoDB se distingue par :

- un langage de requêtes puissant, proche de JSON ;
- un modèle semi-structuré adapté aux données hétérogènes ;
- la possibilité de faire évoluer les schémas très facilement ;
- des mécanismes avancés de réplication (**Replica Sets**) et de distribution (**Sharded Clusters**).

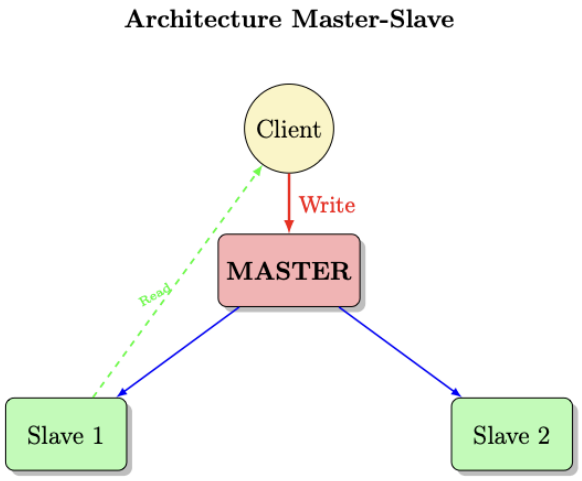
3. Systèmes distribués : définitions et modèles de réplication

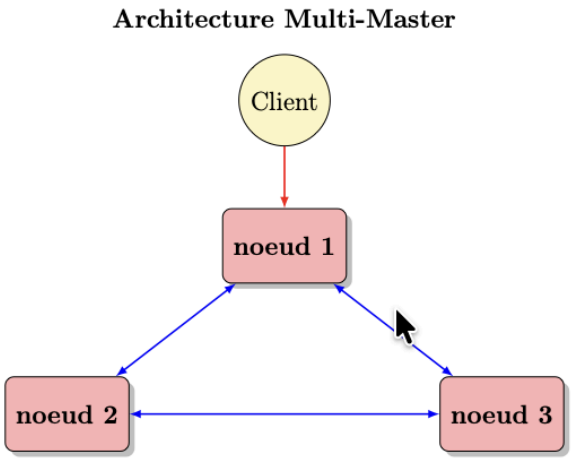
Une base de données distribuée repose sur un ensemble de nœuds interconnectés coopérant pour stocker et servir les données. Les objectifs principaux d'un tel système sont :

- **Disponibilité** : la base doit continuer à fonctionner même si une partie des nœuds échoue.
- **Tolérance aux pannes** : récupération automatique après une défaillance.
- **Scalabilité** : capacité à augmenter les performances en ajoutant des machines.
- **Répartition géographique** : distribution des données sur plusieurs sites.

Dans le contexte des bases distribuées, on distingue souvent deux grands types d'organisation :

- **Architecture Master–Slave:** Dans ce modèle, un nœud est désigné **master** qui reçoit les écritures, plusieurs nœuds **slaves** répliquent les données issues du master, les lectures peuvent être servies par le master ou les slaves.
- **Architecture sans master (Masterless / Peer-to-Peer):** Dans un cluster masterless, tous les nœuds jouent un rôle homogène. Les écritures et lectures peuvent être envoyées à n'importe quel nœud, qui réplique ensuite les données aux autres.

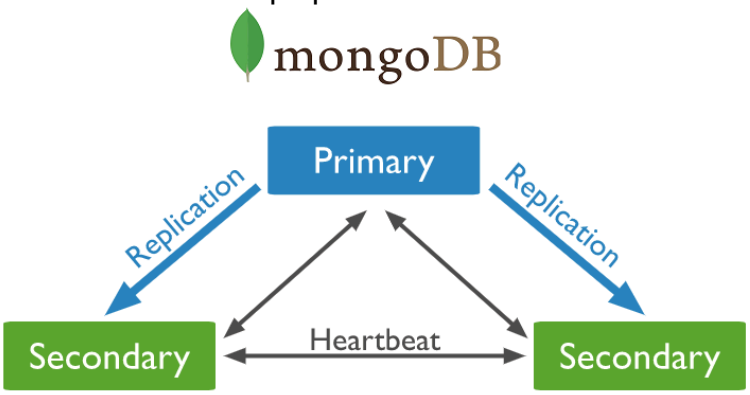




II. Mongodb: mise en échelle:

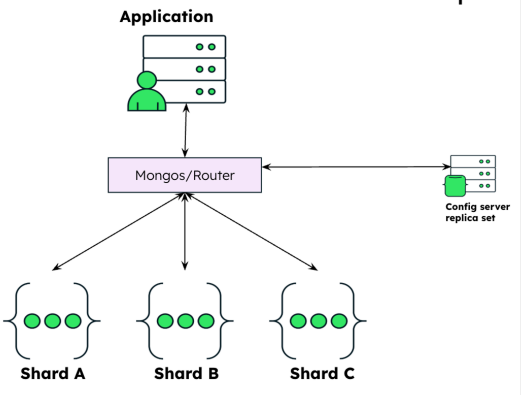
MongoDB assure sa mise à l'échelle et sa haute disponibilité en s'appuyant sur deux mécanismes fondamentaux : la **Réplication** et le **Sharding**.
1. La Réplication

- **Qu'est-ce que c'est et pourquoi ?** La réplication est le processus qui consiste à copier les données sur plusieurs serveurs (ou nœuds) au sein d'un même cluster. L'objectif principal est d'assurer la **haute disponibilité** (le système continue de fonctionner même si un serveur tombe en panne) et la **tolérance aux pannes** (les données ne sont pas perdues). Elle permet également d'améliorer les performances de lecture en distribuant la charge sur les différents serveurs répliqués.



2. Le Sharding

- **Qu'est-ce que c'est et pourquoi ?** Le sharding (ou partitionnement) est une méthode pour distribuer les données sur plusieurs machines indépendantes, appelées "shards" (fragments). L'objectif principal est d'assurer la **scalabilité horizontale** en divisant la base de données en ensembles plus petits et plus gérables. Cela permet de répondre aux défis de stockage et de performance lorsque le volume de données devient trop important pour un seul serveur. Chaque shard contient un sous-ensemble unique des données.



3. Comparaison entre Réplication et Sharding

Caractéristique	Réplication (Replica Set)	Sharding (Cluster Shardé)
Objectif Principal	Haute disponibilité et tolérance aux pannes.	Scalabilité horizontale (stockage et écriture).
Données	Chaque nœud possède une **copie complète et identique** de l'ensemble des données.	Chaque *shard* possède un **sous-ensemble unique** des données.
Limitation de Taille	Limitée par la capacité du plus grand nœud.	La capacité totale est la somme de la capacité de tous les *shards*.
Performance	Améliore les performances de **lecture**.	Améliore les performances d'**écriture** et de **lecture** sur de très grands volumes de données.

1. La Réplication : Le fondement de la Haute Disponibilité avec le Replica Set

L'architecture NoSQL de MongoDB est intrinsèquement liée à la notion de systèmes distribués, ce qui rend la réplication indispensable. Pour concrétiser les objectifs de **haute disponibilité** et de **tolérance aux pannes**, MongoDB utilise un mécanisme spécifique et robuste : le **Replica Set**. Cette section détaille les composants de cette structure et les principes de cohérence qu'elle met en œuvre.**1. Définition et Structure d'un Replica Set**

- **Qu'est-ce qu'un Replica Set dans MongoDB ?**
  Un **Replica Set** est un groupe de processus
  mongod
  qui gère de manière collaborative le même ensemble de données. Il est l'unité de base de la réplication de MongoDB. Sa structure minimale et essentielle repose sur la distinction des rôles entre les membres : un nœud **Primaire (*Primary*)** et un ou plusieurs nœuds **Secondaires (*Secondaries*)**.

2. Les Rôles Clés dans la Réplication

Objectif : Établir une source unique de vérité pour garantir l'intégrité des données tout en permettant la distribution de la charge de travail.

Rôle du Nœud	Description Détaillée
Le Primaire	**Unique responsable des écritures.** Il est le seul membre à accepter les opérations d'insertion, de mise à jour et de suppression. Il enregistre toutes ces modifications dans un journal interne appelé l'**oplog** (opération log). Le nœud primaire est la source de vérité du cluster.
Les Secondaires	**Réplication et Distribution des lectures.** Les secondaires se connectent au primaire et copient en continu l'oplog. Ils appliquent les opérations pour maintenir leur propre copie des données à jour. **Pourquoi ?** Ils servent à distribuer la charge des requêtes de **lecture** et sont prêts à prendre le relais du primaire en cas de défaillance.

3. Le Principe de Cohérence et la Séparation des Écritures

- **Pourquoi l'écriture n'est-elle pas autorisée sur les Secondaires ?**
  L'interdiction des écritures sur les nœuds secondaires est une mesure de conception fondamentale pour prévenir les **conflits de mise à jour** et garantir la **cohérence des données**. Si plusieurs nœuds pouvaient accepter des écritures simultanément, cela mènerait inévitablement à des divergences (état incohérent) qui seraient extrêmement complexes à résoudre. En forçant toutes les écritures à passer par le nœud primaire, on maintient un ordre d'opérations clair et non ambigu.
- **Qu'est-ce que la Cohérence Forte (*Strong Consistency*) ?**
  La cohérence forte est le modèle idéal où le système garantit qu'une fois qu'une écriture est validée, toute lecture ultérieure, effectuée sur n'importe quel nœud, retourne la dernière valeur écrite. Dans un Replica Set de MongoDB, le nœud **Primaire** offre naturellement cette garantie pour ses propres lectures. Les lectures sur les secondaires offrent par défaut une **cohérence éventuelle** (*Eventual Consistency*).

4. Modèles de Réplication : Synchrone vs Asynchrone

Objectif : Choisir entre une latence d'écriture faible et une garantie de cohérence maximale.

Caractéristique	Réplication Synchrone	Réplication Asynchrone
Principe	L'opération d'écriture est confirmée au client uniquement après avoir été répliquée et confirmée par une majorité de secondaires.	L'opération d'écriture est confirmée au client dès qu'elle est validée par le Primaire. La réplication vers les secondaires se fait en arrière-plan.

Avantages	Cohérence forte garantie immédiatement sur tout le cluster.	Faible latence d'écriture (performances rapides).
Inconvénients	Latence d'écriture plus élevée, car elle attend la confirmation des secondaires.	Introduit un léger décalage (*lag*) où les secondaires peuvent avoir des données légèrement obsolètes (cohérence éventuelle).
MongoDB utilise...	Par défaut, MongoDB utilise une réplication **Asynchrone**. Cependant, le développeur peut exiger une **"write concern"** pour simuler un comportement synchrone (exiger la confirmation par N membres) en contrepartie d'une latence accrue.	

2. Mise en place d'un replicaSet

Le but ici est de simuler un cluster mongodb avec 1 primaire et 2 secondaire avec Docker. L'idée est de créer donc 3 conteneur mongodb et de les joindre dans un replicaSet.

Installation:

Dans un fichier `docker_compose.yaml` (il faut d'abord creer les répertoir Master, Slave1 et Slave2 pour le stockage des données)

```yaml
version: '3.8'
services:
 mongo1:
   image: mongo:6
   container_name: mongoMaster
   ports:
     - "27017:27017"
   command: ["mongod", "--replSet", "Myreplicas", "--bind_ip_all"]
   volumes:
     - ./Master:/data/db

 mongo2:
   image: mongo:6
   container_name: mongoSlave1
   command: ["mongod", "--replSet", "Myreplicas", "--bind_ip_all"]
   volumes:
     - ./Slave1:/data/db
```

```
mongo3:
  image: mongo:6
  container_name: mongoSlave2
  command: ["mongod", "--replSet", "Myreplicas", "--bind_ip_all"]
  volumes:
    - ./Slave2:/data/db
```

Puis lancer la commande `docker compose up -d`

Pour vérifier le lancement des conteneur: `docker ps`

```
amzianehamrani@MacBook-Air-de-Amziane-2 NoSQL % docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED      STATUS          PORTS                     NAMES
196f5ba8349b   mongo:6   "docker-entrypoint.s…"   3 days ago   Up 26 seconds   27017/tcp                 mongoSlave1
62ae5482b7b6   mongo:6   "docker-entrypoint.s…"   3 days ago   Up 9 seconds    27017/tcp                 mongoSlave2
8cbbe50ff955   mongo:6   "docker-entrypoint.s…"   3 days ago   Up 40 seconds   0.0.0.0:27017->27017/tcp  mongoMaster
```

Création du replicaSet

Se connecter au container qui joura le rôle du master

`docker exec -it <Container_ID> mongosh`

Initialiser le replicatSet “MyReplicas”

```
rs.initiate({
_id: "Myreplicas",
members: [
   { _id: 0, host: "mongoMaster:27017" },
   { _id: 1, host: "mongoSlave1:27017" },
   { _id: 2, host: "mongoSlave2:27017" }
 ]
})
```

Vérifier l’état du RS:

`rs.config()`

`rs.status()`

Insertion de données:

Télécharger le fichier films.json et le placer dans le répertoire Master précédemment créé, ce qui permettra de placer ce fichier dans /data/db du container mongoMaster.

Insérer les données dans la base test de mongoMaster:

```
amzianehamrani@MacBook-Air-de-Amziane-2 NoSQL % docker exec -it 8cbbe50ff955 mongoimport --db test --collection films --file /data/db/films.j
son --jsonArray
2025-12-07T17:47:23.378+0000    connected to: mongodb://localhost/
2025-12-07T17:47:23.419+0000    278 document(s) imported successfully. 0 document(s) failed to import.
```

Se connecter à mongoMaster pour vérifier le nombre de documents insérés:

```
docker exec -it <Container_ID_Master> mongosh
```

```
Myreplicas [direct: primary] test> db.films.count()
DeprecationWarning: Collection.count() is deprecated. Use countDocuments or estimatedDocumentCount.
278
Myreplicas [direct: primary] test>
```

Vérifier la réplication sur les secondaires:

```
docker exec -it <Container_ID_Slave1> mongosh
```

```
Myreplicas [direct: secondary] test> db.films.count()
DeprecationWarning: Collection.count() is deprecated. Use countDocuments or estimatedDocumentCount.
278
Myreplicas [direct: secondary] test>
```

3. Simulation d'une panne: Master KO

On va supprimer le container mongoMaster

```
docker kill <Container_ID_Master>
```

```
amzianehamrani@MacBook-Air-de-Amziane-2 NoSQL % docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED      STATUS          PORTS                     NAMES
196f5ba8349b   mongo:6   "docker-entrypoint.s…"   3 days ago   Up 27 minutes   27017/tcp                 mongoSlave1
62ae5482b7b6   mongo:6   "docker-entrypoint.s…"   3 days ago   Up 27 minutes   27017/tcp                 mongoSlave2
8cbbe50ff955   mongo:6   "docker-entrypoint.s…"   3 days ago   Up 27 minutes   0.0.0.0:27017->27017/tcp  mongoMaster
amzianehamrani@MacBook-Air-de-Amziane-2 NoSQL % docker kill 8cbbe50ff955
8cbbe50ff955
amzianehamrani@MacBook-Air-de-Amziane-2 NoSQL % docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED      STATUS          PORTS       NAMES
196f5ba8349b   mongo:6   "docker-entrypoint.s…"   3 days ago   Up 28 minutes   27017/tcp   mongoSlave1
62ae5482b7b6   mongo:6   "docker-entrypoint.s…"   3 days ago   Up 27 minutes   27017/tcp   mongoSlave2
amzianehamrani@MacBook-Air-de-Amziane-2 NoSQL %
```

On se connectant à l'un des slave et en exécutant `rs.config()` on pourra identifier le nouveau master du cluster.

```
Myreplicas [direct: primary] test> rs.isMaster()
{
  topologyVersion: {
    processId: ObjectId('6935b9dbe87c7821c522133b'),
    counter: Long('7')
  },
  hosts: [ 'mongo1:27017', 'mongo2:27017', 'mongo3:27017' ],
  setName: 'Myreplicas',
  setVersion: 1,
  ismaster: true,
  secondary: false,
  primary: 'mongo2:27017',
  me: 'mongo2:27017',
```

Imaginons maintenant que l'ex mongoMaster soit rétablie et rejoigne le replicaSet, quel sera son rôle?

```
docker compose up -d mongoMaster
 ✔ Container mongoMaster  Started
```

On remarque plutôt qu'il a été réintégré en tant secondary

III. Questions / Réponses:

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 le même jeu de données avec un Primary et un ou plusieurs Secondaries, afin d'assurer haute disponibilité et tolérance aux pannes.
2. **Quel est le rôle du Primary dans un Replica Set ?**
   Le Primary est le seul nœud qui accepte les opérations d'écriture, enregistre ces opérations dans l'`oplog` et sert de source de vérité pour la réplication vers les Secondaries.
3. **Quel est le rôle essentiel des Secondaries ?**
   Les Secondaries répliquent en continu l'`oplog` du Primary pour maintenir une copie à jour des données et peuvent aussi prendre le relais comme Primary en cas de bascule.
4. **Pourquoi MongoDB n'autorise-t-il pas les écritures sur un Secondary ?**
   Les écritures sont interdites sur les Secondaries pour éviter les conflits de mise à jour et garantir un ordre global unique des opérations en centralisant toutes les écritures sur le Primary.
5. **Qu'est-ce que la cohérence forte dans le contexte MongoDB ?**
   La cohérence forte signifie qu'une fois une écriture validée, toute lecture suivante retourne la dernière valeur, ce que garantit naturellement le Primary pour ses propres lectures.
6. **Quelle est la différence entre readPreference: "primary" et "secondary" ?**
   `readPreference: "primary"` force toutes les lectures vers le Primary avec cohérence forte, alors que `"secondary"` permet de lire sur les Secondaries avec une cohérence seulement éventuelle et un risque de données légèrement en retard.
7. **Dans quel cas pourrait-on souhaiter lire sur un Secondary malgré les risques ?**
   On lit sur un Secondary pour décharger le Primary, pour des rapports ou traitements analytiques où un léger retard des données est acceptable.

Partie 2 — Commandes & configuration

1. **Quelle commande permet d'initialiser un Replica Set ?**
   On initialise un Replica Set avec la commande `rs.initiate({...})` exécutée dans le shell MongoDB sur un nœud.
2. **Comment ajouter un nœud à un Replica Set après son initialisation ?**
   Après l'initialisation, on ajoute un nœud avec `rs.add("hostname:port")` depuis le Primary.
3. **Quelle commande permet d'afficher l'état actuel du Replica Set ?**
   L'état du Replica Set s'affiche avec `rs.status()` dans le shell MongoDB.
4. **Comment identifier le rôle actuel (Primary / Secondary / Arbitre) d'un nœud ?**
   On identifie le rôle via `rs.status()` (champ `stateStr`) ou via `db.hello()` / `db.isMaster()` qui indiquent si le nœud est Primary ou Secondary.
5. **Quelle commande permet de forcer le basculement du Primary ?**
   On force le basculement en exécutant `rs.stepDown()` sur le Primary, ce qui le fait renoncer à son rôle et déclenche une élection.
6. **Comment peut-on désigner un nœud comme Arbitre ? Pourquoi le faire ?**
   On désigne un arbitre avec `rs.addArb("hostname:port")`; on le fait pour ajouter un vote supplémentaire sans stocker de données, afin de conserver une majorité pour les élections.
7. **Donnez la commande pour configurer un nœud secondaire avec un délai de réplication (slaveDelay).**
   On modifie la configuration : `cfg = rs.conf();` `cfg.members[i].slaveDelay = X; cfg.members[i].priority = 0;` `rs.reconfig(cfg);` où X est le délai en secondes.

Partie 3 — Résilience et tolérance aux pannes

1. **Que se passe-t-il si le Primary tombe en panne et qu'il n'y a pas de majorité ?**
   Aucun nouveau Primary ne peut être élu, le Replica Set reste sans Primary et n'accepte plus d'écritures mais les lectures restent possibles sur les membres disponibles.
2. **Comment MongoDB choisit-il un nouveau Primary ? Quels critères utilise-t-il ?**
   MongoDB choisit un nouveau Primary par élection en fonction de la majorité de votes, des priorités configurées des nœuds et de leur fraîcheur de données (`oplog` le plus à jour).
3. **Qu'est-ce qu'une élection dans MongoDB ?**
   Une élection est le processus par lequel les membres votent pour désigner quel Secondary deviendra Primary après une défaillance ou un `stepDown` du Primary.
4. **Que signifie auto-dégradation du Replica Set ? Dans quel cas cela survient-il ?**
   L'auto-dégradation signifie que le Replica Set se met volontairement en mode sans Primary lorsqu'il perd la majorité ou détecte un risque pour la cohérence, afin de bloquer les écritures.
5. **Pourquoi est-il conseillé d'avoir un nombre impair de nœuds dans un Replica Set ?**
   Un nombre impair limite les égalités de votes et facilite l'obtention d'une majorité

claire lors des élections.

6. **Quelles conséquences a une partition réseau sur le fonctionnement du cluster ?**
   Une partition réseau peut séparer les membres en groupes; seul le groupe qui possède la majorité de votes peut élire un Primary, les autres restent sans Primary et en lecture seule.

Partie 4 — Scénarios pratiques

1. **Vous avez 3 nœuds : 27017 (Primary), 27018 (Secondary), et 27019 (Arbitre). Que se passe-t-il si le Primary devient injoignable ?**
   Le Secondary et l'Arbitre forment une majorité et déclenchent une élection, le Secondary 27018 devient alors le nouveau Primary si ses données sont suffisamment à jour.
2. **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 ?**
   Ce Secondary maintient une copie retardée de 2 minutes, utile pour se protéger contre des erreurs applicatives récentes et pour revenir rapidement à un état antérieur en cas de suppression accidentelle.
3. **Un client exige une lecture toujours à jour, même en cas de bascule. Quelles options de readConcern et writeConcern recommanderiez-vous ?**
   On recommande `writeConcern: { w: "majority" }` pour que les écritures soient validées par une majorité, et `readConcern: "majority"` pour ne lire que des données confirmées par la majorité.
4. **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 }` (ou `"majority"` si le Replica Set comporte au moins trois membres).
5. **Un étudiant a lu depuis un Secondary et récupéré une donnée obsolète. Expliquez pourquoi et comment éviter cela.**
   Le Secondary peut être en retard sur le Primary et ne pas avoir appliqué la dernière écriture, ce qui explique la donnée obsolète; pour éviter cela, on lit sur le Primary (`readPreference: "primary"`) et on peut utiliser `readConcern: "majority"`.
6. **Montrez la commande pour vérifier quel nœud est actuellement Primary dans votre Replica Set.**
   On exécute `rs.status()` et on regarde le membre dont l'état est `PRIMARY`, ou `db.hello()` sur un nœud pour voir s'il indique `isWritablePrimary: true`.
7. **Expliquez comment forcer une bascule manuelle du Primary sans interruption majeure.**
   On s'assure d'abord qu'un Secondary est à jour, puis on exécute `rs.stepDown()` sur le Primary pour le faire passer en Secondary et déclencher une élection rapide vers un autre nœud.

8. **Décrivez la procédure pour ajouter un nouveau nœud secondaire dans un Replica Set en fonctionnement.**
   On démarre un nouveau `mongod` avec le même nom de Replica Set (`--replSet`), puis depuis le Primary on lance `rs.add("nouveauHost:port")` et on laisse la synchronisation se faire.
9. **Quelle commande permet de retirer un nœud défectueux d'un Replica Set ?**
   On retire un membre avec `rs.remove("hostname:port")` exécuté sur le Primary.
10. **Comment configurer un nœud secondaire pour qu'il soit caché (non visible aux clients) ? Pourquoi ferait-on cela ?**
    On le configure avec `hidden: true` et `priority: 0` dans la configuration (`rs.conf` / `rs.reconfig`), afin qu'il ne soit pas ciblé par les clients pour les lectures et qu'il serve par exemple à la sauvegarde ou à l'analytique interne.
11. **Montrez comment modifier la priorité d'un nœud afin qu'il devienne le Primary préféré.**
    On modifie la config : `cfg = rs.conf(); cfg.members[i].priority = 2; rs.reconfig(cfg);` (avec une priorité plus élevée que les autres membres pour en faire le Primary préféré).
12. **Expliquez comment vérifier le délai de réplication d'un Secondary par rapport au Primary.**
    On utilise `rs.printSlaveReplicationInfo()` ou `rs.printReplicationInfo()` pour comparer les timestamps de l'`oplog` Primary/Secondaries et en déduire le `lag` en secondes.
13. **Que fait la commande rs.freeze() et dans quel scénario est-elle utile ?**
    `rs.freeze(seconds)` rend le membre inéligible pour devenir Primary pendant un certain temps, ce qui est utile pour empêcher un Secondary particulier d'être élu lors d'opérations de maintenance.
14. **Comment redémarrer un Replica Set sans perdre la configuration ?**
    La configuration du Replica Set est stockée dans la base `local`; tant qu'on redémarre chaque `mongod` avec les mêmes données et le même nom de Replica Set, la configuration est conservée automatiquement.
15. **Expliquez comment surveiller en temps réel la réplication via les logs MongoDB ou commandes shell.**
    On peut surveiller la réplication en consultant les logs (`repl/heartbeat`) ou en utilisant régulièrement `rs.status()`, `rs.printSlaveReplicationInfo()` et des outils de monitoring pour suivre le `lag` et l'état des membres.

Questions complémentaires

1. **Qu'est-ce qu'un Arbitre (Arbiter) et pourquoi ne stocke-t-il pas de données ?**
   Un Arbitre est un membre spécial d'un Replica Set qui participe uniquement au vote lors des élections et ne stocke pas de données pour limiter la consommation de ressources et les risques de sécurité.

2. **Comment vérifier la latence de réplication entre le Primary et les Secondaries ?**
   On vérifie la latence avec `rs.printSlaveReplicationInfo()` ou `rs.status()`, qui indiquent le décalage temporel entre les points d'`oplog` du Primary et des Secondaries.
3. **Quelle commande MongoDB permet d'afficher le retard de réplication des membres secondaires ?**
   La commande `rs.printSlaveReplicationInfo()` affiche précisément le retard de réplication des Secondaries par rapport au Primary.
4. **Quelle est la différence entre la réplication asynchrone et synchrone ? Quel type utilise MongoDB ?**
   En réplication synchrone, l'écriture n'est confirmée qu'après propagation sur plusieurs nœuds, alors qu'en réplication asynchrone elle est confirmée dès qu'elle est validée sur le Primary; MongoDB utilise la réplication asynchrone par défaut, avec la possibilité d'approcher un comportement synchrone via `writeConcern`.
5. **Peut-on modifier la configuration d'un Replica Set sans redémarrer les serveurs ?**
   Oui, la configuration peut être modifiée à chaud via `rs.reconfig(cfg)` sans redémarrer les `mongod`.
6. **Que se passe-t-il si un nœud Secondary est en retard de plusieurs minutes ?**
   Le Secondary continue à rattraper les opérations depuis l'`oplog` du Primary, mais les lectures qu'il sert peuvent être très obsolètes tant que le retard n'est pas résorbé.
7. **Comment MongoDB gère-t-il les conflits de données lors de la réplication ?**
   MongoDB applique un modèle "Primary unique" et un journal d'opérations ordonné pour éviter les conflits; en cas de changement de Primary, les opérations sont résolues selon l'ordre des `optime` de l'`oplog`, ce qui peut entraîner le `rollback` de certaines écritures.
8. **Est-il possible d'avoir plusieurs Primarys simultanément dans un Replica Set ? Pourquoi ?**
   Non, un Replica Set ne peut avoir qu'un seul Primary à la fois afin de garantir un ordre global unique des écritures et éviter les divergences de données.
9. **Pourquoi est-il déconseillé d'utiliser un Secondary pour des opérations d'écriture même en lecture préférée secondaire ?**
   Les Secondaries ne sont pas conçus pour accepter des écritures directes; les forcer à le faire casserait le protocole de réplication et conduirait à des données incohérentes par rapport au Primary.
10. **Quelles sont les conséquences d'un réseau instable sur un Replica Set ?**
    Un réseau instable provoque des bascules fréquentes, des partitions, du retard de réplication et potentiellement des `rollbacks`, ce qui dégrade la cohérence apparente et la disponibilité des écritures.