Katell GOUZERH ING3A INFO
2025-2026

TP3 : Le partitionnement sous MongoDB

Le but de ce TP est de déployer un cluster MongoDB shardé et de comprendre ce qui se passe quand on insère beaucoup de documents dans une collection shardée.

Partie 1 : Manipulations

Dans ce TP, on travaille en local avec :

- 1 replica set pour les config servers : replicaconfig (ici, un seul nœud pour le TP)
- 2 shards : replicashard1 et replicashard2
- 1 routeur mongos

On commence par créer les répertoires de données :

```
mkdir -p configsvrdb
mkdir -p serv1
mkdir -p serv2
mkdir -p logs
```

Ensuite on démarre le config server (port 27019) et on initialise le replica set :

```
mongod --configsvr --replSet replicaconfig --dbpath configsvrdb --port 27019 --logpath logs/configsvr.log –fork
mongosh --port 27019
rs.initiate({
  _id: "replicaconfig",
  configsvr: true,
  members: [{ _id: 0, host: "localhost:27019" }]
})
rs.status() //verification
```

Ensuite on démare les shards (ports 200004 et 20005).

Shard 1 :

```
mongod --shardsvr --replSet replicashard1 --dbpath serv1 --port 20004 --logpath logs/shard1.log –fork
mongosh --port 20004
rs.initiate({
  _id: "replicashard1",
  members: [{ _id: 0, host: "localhost:20004" }]
})
```

Shard 2 :

```
mongod --shardsvr --replSet replicashard2 --dbpath serv2 --port 20005 --logpath logs/shard2.log –fork
mongosh --port 20005
rs.initiate({
  _id: "replicashard2",
```

```
  members: [{ _id: 0, host: "localhost:20005" }]
})
```

On démarre le routeur mongos, on se connecte au routeur et on ajoute des shards :

```
mongos --configdb replicaconfig/localhost:27019 --port 27017 --logpath logs/mongos.log –fork
mongosh --port 27017 //connexion au routeur
sh.addShard("replicashard1/localhost:20004") //ajout des shards
sh.addShard("replicashard2/localhost:20005")
sh.status() //verification
```

Maintenant on active le sharding sur la base :

```
sh.enableSharding("mabasefilms") //sur la base
sh.shardCollection("mabasefilms.films", { titre: 1 }) //sur la collection avec une shard key
```

Ensuite on utilise le script python. On l’exécute avec la commande : *python3 monappunparun.py*.

Ensuite on regarde l’état du cluster :

```
sh.status()
```

On vérifie la distribution des données :

```
use mabasefilms
db.films.getShardDistribution()
```

On observe les chunks :

```
use config
db.collections.find({ _id: "mabasefilms.films" }).pretty()
db.chunks.find({ ns: "mabasefilms.films" }).pretty()
```

Puis on surveille le balancer :

```
sh.getBalancerState()
sh.isBalancerRunning()
```

Au départ, la collection shardée démarre avec un chunk “initial” localisé sur un shard. Quand les données grossissent, MongoDB déclenche des splits : un chunk devient plusieurs chunks plus petits. Le balancer s’occupe ensuite de limiter les déséquilibres : si un shard a “trop” de chunks (ou si la distribution devient mauvaise), il lance des migrations de chunks vers d’autres shards. Si la shard key distribue mal les écritures (ex. clé monotone), on obtient un hot shard : un shard reçoit presque toute la charge d’écriture, ce qui annule l’intérêt du sharding.

Partie 2 : Questions

1. Qu'est-ce que le sharding dans MongoDB et pourquoi est-il utilisé ?

Le sharding dans MongoDB est une technique de partitionnement horizontal qui permet de répartir les documents d'une collection sur plusieurs serveurs appelés shards. Cette approche est utilisée lorsque les données deviennent trop volumineuses pour être stockées ou traitées efficacement par un seul serveur. En divisant les données et en les distribuant sur plusieurs machines, le système peut évoluer de manière horizontale, augmenter sa capacité et améliorer ses performances. Le sharding permet ainsi de gérer de très grands volumes de données, de plusieurs téraoctets à plusieurs pétaoctets, tout en réduisant la charge sur chaque serveur. Grâce à cette répartition, les opérations de lecture et d'écriture sont partagées entre plusieurs machines, ce qui améliore le débit global et diminue la latence, car les requêtes peuvent être traitées en parallèle sur différents shards.

2. Quelle est la différence entre le sharding et la réplication dans MongoDB ?

Dans MongoDB, le sharding et la réplication sont deux mécanismes complémentaires mais qui poursuivent des objectifs différents. La réplication, réalisée au moyen d'un replica set, vise principalement la haute disponibilité et la tolérance aux pannes. Elle repose sur la création de copies identiques des mêmes données sur plusieurs serveurs, ce qui permet au système de continuer à fonctionner même si l'un d'eux tombe en panne, puisqu'un autre membre du replica set prend automatiquement le relais. Les données y sont donc entièrement redondantes. Le sharding, en revanche, a pour objectif le passage à l'échelle et la gestion de très gros volumes de données. Il consiste à répartir des parties différentes des données sur plusieurs serveurs, chacun contenant un sous-ensemble unique de la collection. Les données ne sont donc plus dupliquées, mais fragmentées. En pratique, ces deux approches sont souvent combinées, chaque shard étant lui-même constitué d'un replica set, ce qui permet de bénéficier à la fois de la scalabilité du sharding et de la haute disponibilité offerte par la réplication.

3. Quels sont les composants d'une architecture shardée (mongos, config servers, shards) ?

Une architecture shardée MongoDB repose sur trois composants principaux qui assurent ensemble la distribution et le fonctionnement du cluster. Les shards constituent la partie où les données sont réellement stockées. Chaque shard contient un sous-ensemble des documents d'une collection et, en environnement de production, chaque shard est généralement organisé sous la forme d'un replica set afin de garantir la haute disponibilité. Les config servers, regroupés dans un Config Server Replica Set, jouent un rôle central puisqu'ils conservent toutes les métadonnées du cluster, notamment la localisation de chaque portion de données, appelée chunk. Ils maintiennent en permanence la carte de distribution des données et sont indispensables au fonctionnement du cluster, car leur indisponibilité rendrait l'ensemble du système inopérant. Pour cette raison, ils doivent toujours être répliqués, le plus souvent avec trois instances. Enfin, les mongos agissent comme des routeurs et servent de point d'entrée pour les applications clientes. Ils ne stockent aucune donnée, mais analysent les requêtes et les dirigent vers les shards concernés. Pour savoir où se trouvent les données, ils consultent les config servers. Il peut y avoir plusieurs mongos afin d'assurer une bonne répartition de la charge et une haute disponibilité du routage.

4. Quelles sont les responsabilités des config servers (CSRS) dans un cluster shardé ?

Dans un cluster shardé, les config servers ont un rôle essentiel car ils stockent toutes les métadonnées nécessaires au fonctionnement du système, comme la répartition des plages de clés entre les shards, la liste des shards du cluster, les informations sur les bases et collections shardées ainsi que l'état des chunks. Ils assurent également la coordination du cluster en gérant les

migrations de chunks, en suivant l'activité du balancer et en conservant l'historique des mouvements de données. Les mongos s'appuient constamment sur eux pour savoir où se trouvent les données et router correctement les requêtes. Comme toute modification de la topologie dépend d'eux, ils doivent impérativement être répliqués, car la perte d'un config server empêcherait entièrement le cluster de fonctionner.

5. Quel est le rôle du mongos router ?
Le mongos joue le rôle de routeur dans un cluster shardé. Il reçoit les requêtes des applications clientes, détermine quels shards contiennent les données nécessaires, leur transmet les opérations et, si besoin, agrège les résultats lorsqu'une requête implique plusieurs shards. Pour l'application, ce fonctionnement est totalement transparent, car elle se connecte au mongos comme à un serveur MongoDB classique sans connaître la structure du cluster. Le mongos s'appuie sur les config servers pour obtenir la carte de répartition des données, qu'il met ensuite en cache afin d'optimiser le routage. Il peut cibler directement un shard lorsque la requête contient la clé de sharding ou, à défaut, interroger l'ensemble des shards lorsque cette information n'est pas disponible.

6. Comment MongoDB décide-t-il sur quel shard stocker un document ?
MongoDB choisit le shard sur lequel stocker un document en fonction de la clé de sharding. Selon le type de sharding utilisé, la valeur de cette clé est soit prise directement, comme dans le range sharding, soit transformée par une fonction de hachage dans le cas du hashed sharding. MongoDB détermine ensuite dans quel chunk se situe cette valeur, puis consulte les config servers pour savoir quel shard héberge ce chunk. Le document est alors inséré sur ce shard. Par exemple, avec un sharding par plage sur le champ titre, un document dont le titre commence par A peut être dirigé vers un chunk correspondant à la plage A–D et donc vers un shard particulier, tandis qu'un titre commençant par T ira vers une autre plage et un autre shard. Avec un sharding basé sur le hachage, c'est la valeur hachée du champ qui définit le chunk de destination. Il est important de noter qu'une fois la clé de sharding définie pour un document, elle ne peut plus être modifiée.

7. Qu'est-ce qu'une clé de sharding et pourquoi est-elle essentielle ?
La clé de sharding est un champ, ou un ensemble de champs, utilisé par MongoDB pour décider comment répartir les documents entre les différents shards. Elle est essentielle car elle conditionne directement la distribution des données dans le cluster, ce qui influence l'équilibre de charge et les performances globales. Elle permet également au mongos de router efficacement les requêtes : lorsqu'une requête contient la clé de sharding, le mongos peut cibler uniquement le shard concerné, tandis qu'en son absence il est contraint d'interroger tous les shards, ce qui ralentit l'exécution. Une bonne clé garantit donc une répartition homogène des données et des requêtes rapides, alors qu'une clé mal choisie peut entraîner un fort déséquilibre du cluster. La clé de sharding est immuable, ce qui signifie qu'elle ne peut pas être modifiée une fois la collection créée, d'où l'importance de la choisir avec soin dès le départ. MongoDB crée par ailleurs automatiquement un index sur cette clé pour optimiser les recherches.

8. Quels sont les critères de choix d'une bonne clé de sharding ?
Le choix d'une bonne clé de sharding repose avant tout sur trois critères essentiels. Elle doit d'abord présenter une forte cardinalité, c'est-à-dire offrir un grand nombre de valeurs distinctes afin de permettre une répartition efficace des documents entre les shards. Elle doit ensuite être utilisée avec une fréquence équilibrée pour éviter que certaines valeurs trop fréquentes ne concentrent une grande partie des données sur un seul shard, ce qui créerait un déséquilibre. Enfin, elle ne doit pas

suivre une progression monotone, comme un timestamp ou un identifiant auto-incrémenté, afin d'empêcher que toutes les nouvelles insertions se retrouvent toujours sur le même shard. En complément, une bonne clé doit correspondre aux schémas de requêtes les plus courants, permettre d'éviter les requêtes broadcast et favoriser une distribution uniforme des données. Ce choix conditionne directement les performances et l'équilibre du cluster.

9. Qu'est-ce qu'un chunk dans MongoDB ?

Dans MongoDB, un chunk représente une plage continue de valeurs définie à partir de la clé de sharding. Il s'agit de l'unité logique utilisée pour répartir et déplacer les données dans un cluster shardé. Chaque chunk correspond à une portion de données comprise entre une valeur minimale et une valeur maximale de la clé, et il est toujours stocké sur un seul shard, même si un shard peut en contenir plusieurs. La taille d'un chunk est par défaut de 64 Mo, mais elle peut être ajustée selon les besoins. Ces informations, incluant les bornes min et max ainsi que le shard qui héberge le chunk, sont enregistrées dans les config servers. Lors du balancing, ce sont les chunks qui sont déplacés d'un shard à un autre afin de maintenir une distribution équilibrée des données. Par exemple, avec un sharding par plages sur un champ comme titre, un chunk peut regrouper les valeurs de A à M tandis qu'un autre couvrira celles de M à Z.

10. Comment fonctionne le splitting des chunks ?

Le splitting est le mécanisme par lequel MongoDB divise un chunk devenu trop volumineux en deux chunks plus petits. Lorsque la taille maximale configurée est dépassée, MongoDB détecte automatiquement le dépassement, calcule un point de coupure en se basant sur la répartition des valeurs de la clé de sharding, puis crée deux chunks correspondant chacun à une moitié de la plage initiale. Les config servers mettent alors à jour les métadonnées pour enregistrer ces nouvelles limites. Le split ne provoque aucune migration : les deux chunks restent sur le même shard, et ce n'est que si un déséquilibre apparaît que le balancer décidera éventuellement de déplacer l'un d'eux. Le splitting peut aussi être lancé manuellement lorsque cela est nécessaire.

11. Que fait le balancer dans un cluster shardé ?

Le splitting est le mécanisme par lequel MongoDB divise un chunk devenu trop volumineux en deux chunks plus petits. Lorsque la taille maximale configurée est dépassée, MongoDB détecte automatiquement le dépassement, calcule un point de coupure en se basant sur la répartition des valeurs de la clé de sharding, puis crée deux chunks correspondant chacun à une moitié de la plage initiale. Les config servers mettent alors à jour les métadonnées pour enregistrer ces nouvelles limites. Le split ne provoque aucune migration : les deux chunks restent sur le même shard, et ce n'est que si un déséquilibre apparaît que le balancer décidera éventuellement de déplacer l'un d'eux. Le splitting peut aussi être lancé manuellement lorsque cela est nécessaire.

12. Quand et comment le balancer déplace-t-il des chunks ?

Le balancer déplace des chunks dès qu'il détecte un déséquilibre notable entre les shards, en comparant leur nombre de chunks à l'aide de seuils prédéfinis. Lorsqu'un shard en possède significativement plus qu'un autre, une migration est lancée. Le processus consiste d'abord à sélectionner un chunk sur le shard surchargé, puis à en copier les données vers le shard cible. Pendant cette copie, toutes les nouvelles écritures sur ce chunk sont enregistrées afin d'être appliquées ensuite, ce qui permet de maintenir la cohérence. Une fois la copie terminée et rattrapée, les config servers mettent à jour les métadonnées pour indiquer le nouveau shard propriétaire, puis l'ancienne copie est supprimée. Cette migration est transparente pour l'application, n'interrompt ni

les lectures ni les écritures, et peut être lente si le chunk est volumineux. Par défaut, une seule migration est effectuée à la fois.

13. Qu'est-ce qu'un hot shard et comment l'éviter ?
Un hot shard est un shard qui reçoit beaucoup plus de requêtes ou d'écritures que les autres, ce qui crée une surcharge, ralentit tout le cluster et peut saturer le serveur concerné. Ce phénomène apparaît souvent lorsque la clé de sharding est monotone, comme un identifiant croissant ou un timestamp, car toutes les nouvelles insertions se retrouvent dirigées vers le dernier chunk et donc vers le même shard. Il peut aussi survenir lorsque certaines valeurs de la clé sont beaucoup plus fréquentes que d'autres, ou lorsque la cardinalité de la clé est trop faible pour permettre une répartition efficace. Pour éviter ce déséquilibre, on peut utiliser une clé hashée qui uniformise naturellement la distribution, choisir une clé composée qui combine un champ métier avec un identifiant unique, sélectionner une clé conforme aux schémas de requêtes, mettre en place du zone sharding ou surveiller en continu la distribution et le trafic afin de détecter les dérives.

14. Quels problèmes une clé de sharding monotone peut-elle engendrer ?
Une clé de sharding monotone, dont les valeurs ne cessent d'augmenter, entraîne plusieurs problèmes majeurs dans un cluster MongoDB. Comme toutes les nouvelles insertions se situent toujours dans la plage de valeurs la plus récente, elles finissent dans le même chunk et donc sur le même shard, ce qui crée un hot shard recevant l'ensemble des écritures tandis que les autres restent sous-utilisés. Le dernier chunk grossit alors très rapidement jusqu'à dépasser sa taille maximale, ce qui peut mener à la formation de jumbo chunks difficiles à diviser et impossibles à migrer correctement. Cette situation annule l'intérêt du sharding puisque la charge n'est plus distribuée, et le balancer tente en vain de rééquilibrer les données : il migre un chunk qui continue aussitôt de croître sur son nouveau shard, générant des migrations coûteuses et inefficaces. À terme, il devient presque impossible de gagner en performance en ajoutant de nouveaux shards, car ils restent vides tant que la clé reste monotone. Pour éviter ces dérives, il est préférable d'utiliser une clé hashée, une clé composée ou encore d'introduire un préfixe aléatoire afin de casser la monotonie des valeurs.

15. Comment activer le sharding sur une base de données et sur une collection ?
L'activation du sharding dans MongoDB se déroule en deux étapes. Il faut d'abord activer le sharding au niveau de la base de données afin de permettre la création de collections shardées, ce qui se fait avec la commande enableSharding. Ensuite, le sharding doit être activé sur une collection précise en indiquant la clé de sharding à utiliser, qu'il s'agisse d'un sharding par plages, d'un sharding hashé ou d'une clé composée. MongoDB crée automatiquement l'index correspondant si celui-ci n'existe pas encore. La collection peut être vide ou déjà contenir des données et, une fois shardée, sa clé de sharding devient définitive. Depuis un mongos, l'administrateur active d'abord le sharding sur la base, puis sur la collection, avant de vérifier la configuration via sh.status.

16. Comment ajouter un nouveau shard à un cluster MongoDB ?
Pour ajouter un nouveau shard à un cluster MongoDB, il faut d'abord démarrer le serveur qui constituera ce shard, généralement sous la forme d'un replica set, puis l'initialiser. Une fois ce shard opérationnel, on se connecte au mongos et on l'intègre au cluster avec la commande addShard, en indiquant le nom du replica set et les hôtes qui le composent. Dès son ajout, le shard devient immédiatement membre du cluster et le balancer commence progressivement à lui transférer des chunks afin de rééquilibrer la distribution des données. L'opération peut être réalisée à chaud et n'interrompt pas le fonctionnement du cluster, et l'on peut vérifier l'intégration du nouveau shard

via sh.status. Ce mécanisme permet d’augmenter la capacité du cluster à mesure que les besoins grandissent.

17. Comment vérifier l'état du cluster shardé (commandes usuelles) ?
L’état d’un cluster shardé peut être vérifié grâce à plusieurs commandes d’administration. La commande sh.status fournit une vue d’ensemble du cluster, incluant les shards disponibles, les bases et collections shardées, la clé utilisée et la distribution des chunks. Pour obtenir des informations plus précises sur les shards eux-mêmes, la commande listShards renvoie la liste des shards avec leurs adresses et leur état. La distribution détaillée d’une collection peut être observée via getShardDistribution, qui indique le nombre de documents, la taille des données et le nombre de chunks par shard. L’activité du balancer se vérifie à l’aide de getBalancerState ou isBalancerRunning. Les métadonnées d’une collection shardée, notamment sa version et sa configuration, peuvent être consultées avec getShardVersion, tandis que le contenu réel de la table des chunks est accessible dans la base config en interrogeant db.chunks, ce qui permet d’observer les plages min et max ainsi que l’hébergement de chaque chunk. L’historique des événements du balancer, incluant splits, migrations et éventuelles erreurs, est consultable dans config.changelog. Enfin, des commandes comme db.stats ou un sh.status en mode verbose donnent des informations complémentaires lorsque des zones ou d’autres mécanismes avancés sont utilisés.

18. Dans quels cas faut-il envisager d'utiliser un hashed sharding key ?
Le hashed sharding est particulièrement adapté lorsque la distribution uniforme des données et des écritures est prioritaire. Il est utile dans les situations où la clé de sharding est monotone, comme un identifiant croissant, un timestamp ou un compteur, car le hachage permet alors de répartir les insertions sur tous les shards et d’éviter la formation d’un hot shard. Il convient aussi lorsque les requêtes ne reposent pas sur des recherches par plage mais sur des correspondances exactes, par exemple lorsqu’on interroge un userId précis. Cette approche est idéale pour les charges d’écriture élevées, les flux continus d’événements ou les jeux de données dont la distribution naturelle est très déséquilibrée, car le hachage produit une répartition presque parfaite et facilite un scaling linéaire. En contrepartie, elle rend les requêtes par plage inefficaces et supprime la localité des données, ce qui peut entraîner des broadcasts lorsque la clé exacte n’est pas fournie dans la requête.

19. Dans quels cas faut-il privilégier un ranged sharding key ?
Le ranged sharding est à privilégier lorsque les requêtes par plage sont fréquentes, car il permet au mongos de cibler directement les shards concernés au lieu de diffuser la requête partout comme avec un hachage. Il est également adapté lorsque la localité des données est importante, par exemple pour regrouper des informations liées sur un même shard et faciliter les agrégations ou les traitements analytiques. Cette approche fonctionne particulièrement bien si la clé choisie présente une distribution naturelle équilibrée, comme un email ou un identifiant aléatoire, et elle est très efficace pour les tris puisqu’elle conserve l’ordre des valeurs. Le ranged sharding est aussi indispensable dans les architectures nécessitant du zone sharding, notamment pour des contraintes géographiques ou réglementaires. Enfin, il convient aux clés composites qui reflètent la logique métier, comme l’association d’un tenantId et d’une date de création. En revanche, il exige une bonne compréhension des patterns d’accès et peut entraîner des hot shards si la clé n’est pas choisie avec soin.

20. Qu'est-ce que le zone sharding et quel est son intérêt ?

Le zone sharding consiste à associer certaines plages de valeurs de la clé de sharding à des shards déterminés, de sorte que le balancer place les données dans les zones définies plutôt que de les répartir librement. On crée des zones, on assigne des shards à ces zones, puis on indique quelles plages de données doivent s'y trouver, et le cluster maintient cette organisation lors des migrations. Cette approche est particulièrement utile pour satisfaire des contraintes réglementaires, par exemple en garantissant que les données européennes restent stockées sur des serveurs situés en Europe. Elle permet aussi de réduire la latence en rapprochant géographiquement les données des utilisateurs, d'isoler certaines charges comme des clients sensibles ou des environnements spécifiques, et d'organiser les données selon leur fréquence d'utilisation, en séparant les données récentes sur des shards rapides et les archives sur des infrastructures moins coûteuses. Enfin, elle s'avère adaptée aux environnements multi-tenant en permettant d'attribuer des plages dédiées à chaque client.

21. Comment MongoDB gère-t-il les requêtes multi-shards ?

Lorsqu'une requête doit être exécutée sur plusieurs shards, elle est d'abord analysée par le mongos, qui détermine si elle peut être ciblée vers un shard précis ou si elle doit être diffusée à l'ensemble des shards faute de disposer de la clé de sharding. Les shards concernés exécutent alors la requête indépendamment et renvoient leurs résultats partiels au mongos, qui se charge de les fusionner, d'appliquer les opérations globales comme le tri, les limites ou les sauts, puis de renvoyer la réponse complète au client. Dans les pipelines d'agrégation, MongoDB exécute autant d'étapes que possible sur les shards, tandis que les opérations nécessitant une vue globale, comme un tri ou un groupement final, sont effectuées par le mongos. De la même manière, les opérations comme count ou distinct sont décomposées en calculs locaux sur chaque shard puis agrégées par le routeur. Les requêtes qui ne peuvent pas être ciblées, c'est-à-dire celles qui ne contiennent pas la clé de sharding, sont plus coûteuses car elles sollicitent tous les shards en parallèle.

22. Comment optimiser les performances de requêtes dans un environnement shardé ?

L'optimisation des requêtes dans un environnement shardé repose avant tout sur la capacité à cibler les shards plutôt que d'interroger tout le cluster. Pour cela, il est essentiel que les requêtes incluent la clé de sharding, ce qui permet au mongos de diriger l'opération vers un seul shard et de réduire considérablement les coûts. Des index adaptés aux filtres, tris et jointures fréquents doivent être présents sur chaque shard, et il peut être pertinent de créer des index composés combinant la clé de sharding avec d'autres champs ciblés par l'application. Les projections permettent de limiter la quantité de données transférées, tandis que les limites appliquées dès les premières étapes réduisent le volume traité par chaque shard. Les tris doivent être soutenus par un index pour éviter des opérations coûteuses en parallèle, et les pipelines d'agrégation doivent commencer par des filtrages efficaces afin d'être exécutés au maximum sur les shards. Les préférences de lecture peuvent répartir la charge entre les nœuds, les opérations en batch optimisent les écritures et les covered queries améliorent encore les performances en utilisant uniquement l'index. L'analyse régulière des requêtes via explain permet enfin d'identifier les points de blocage, mais la meilleure optimisation reste de choisir une clé de sharding cohérente avec les patterns d'accès de l'application.

23. Que se passe-t-il lorsqu'un shard devient indisponible ?

Lorsqu'un shard devient indisponible, l'impact dépend de son architecture. Un shard isolé sans replica set rend immédiatement toutes ses données inaccessibles, ce qui peut provoquer des erreurs dans les requêtes qui les concernent et entraîner un risque réel de perte de données, raison pour laquelle cette configuration n'est jamais recommandée en production. Avec un shard configuré en replica set, la situation est beaucoup plus résiliente. Si un secondary tombe, le fonctionnement du

cluster n’est pas affecté et les opérations continuent normalement. Si le primary devient indisponible, le replica set déclenche une élection automatique afin de nommer un nouveau primary, ce qui entraîne une courte interruption des écritures mais permet une reprise rapide grâce à la redondance des données. En revanche, si tous les membres d’un shard sont indisponibles, les données de ce shard ne peuvent plus être consultées et les requêtes broadcast ne renvoient que des résultats partiels, tandis que les requêtes dirigées spécifiquement vers ce shard échouent. Le balancer suspend alors toute migration liée à ce shard et ne reprend ses activités qu’après son rétablissement. MongoDB ne redistribue jamais automatiquement les données d’un shard définitivement perdu, ce qui impose une intervention manuelle. Pour limiter les risques, il est indispensable d’utiliser des replica sets, de surveiller en continu la santé du cluster, d’effectuer des sauvegardes régulières et de tester les mécanismes de failover.

24. Comment migrer une collection existante vers un schéma shardé ?
Pour migrer une collection existante vers un schéma shardé, il faut d’abord préparer la transition en analysant la collection, ses index et surtout les patterns de requêtes, afin de choisir une clé de sharding adaptée en termes de cardinalité, de fréquence d’apparition et de mode d’accès aux données. Une fois cette clé déterminée, on commence par créer l’index correspondant sur la collection, car MongoDB exige qu’un index existe sur la clé de sharding avant d’autoriser le sharding. Ensuite, on se connecte au mongos, on active le sharding sur la base de données avec la commande d’activation globale, puis on shard la collection en spécifiant la clé retenue, qu’il s’agisse d’un sharding par plage ou hashé. Le balancer peut être laissé actif ou configuré pour ne migrer les chunks que dans une plage horaire de maintenance, afin de limiter l’impact sur la production. Pendant la migration, la collection reste disponible, mais des migrations de chunks en arrière-plan peuvent dégrader légèrement les performances, surtout si le volume de données est très important. Il est donc conseillé de surveiller la distribution des données et l’activité du balancer grâce aux commandes d’état du sharding et aux collections de la base config, puis d’ajuster si nécessaire des paramètres comme la taille des chunks. Pour les collections très volumineuses ou en cas de chargement massif, on peut recourir au pré-split, en créant manuellement des splits sur la clé de sharding avant que le balancer ne commence ses migrations, ce qui permet de mieux contrôler la répartition initiale des données et de réduire la charge liée aux splits automatiques. Une fois la migration stabilisée, la clé de sharding devient définitive et sert de base à la scalabilité future de la collection.

25. Quels outils ou métriques utiliser pour diagnostiquer les problèmes de sharding ?
Pour diagnostiquer les problèmes de sharding, MongoDB met à disposition plusieurs outils permettant d’observer la répartition des données, l’activité du cluster et le comportement des requêtes. La commande sh.status fournit une vue d’ensemble sur les shards, les collections shardées, la distribution des chunks et l’état du balancer. L’utilisation d’explain permet d’analyser le plan d’exécution d’une requête et d’identifier le nombre de documents examinés, le temps total d’exécution ou encore les shards sollicités, ce qui aide à repérer les requêtes non ciblées ou les index manquants. La distribution d’une collection peut être examinée avec getShardDistribution, tandis que currentOp donne de la visibilité sur les migrations en cours ou les requêtes lentes. Le profiler enregistre les opérations coûteuses et permet de repérer les filtres et tris problématiques. La base config offre une vision fine des métadonnées du cluster, notamment les chunks, l’historique des migrations et l’état des shards. Des outils système comme mongostat et mongotop permettent de suivre en temps réel l’activité du cluster et la charge par collection. MongoDB Compass, Ops Manager ou Atlas fournissent une visualisation plus approfondie de la santé du cluster, des latences

ou de l’équilibre du sharding. Les logs complètent ce diagnostic en permettant d’identifier anomalies et erreurs. Les métriques importantes à surveiller incluent la latence des requêtes, la distribution des chunks, l’équilibre des shards, l’activité du balancer, ainsi que les ressources système telles que CPU, mémoire, disque et latence réseau. L’ensemble de ces outils permet d’isoler les problèmes courants, qu’il s’agisse de requêtes lentes, de déséquilibres entre shards ou de hot shards, et d’orienter les actions correctives.