I- Mise en place du cluster shardé (config server, shards, mongos)

Dans ce TP, on met en place un cluster shardé en local composé de :

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

1) Préparer l'arborescence des données et des logs

On commence par créer les dossiers nécessaires au stockage des données et aux fichiers de logs :

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

2) Lancer le Config Server et initialiser son Replica Set

On démarre le config server (port 27019), puis on initialise le replica set replicaconfig :

```
mongod --configsvr --replSet replicaconfig --dbpath configsvrdb --port 27019 --logpath logs/configsvr.log --fork
```

Connexion au config server :

```
mongosh --port 27019
```

Initialisation du replica set et vérification :

```
rs.initiate({
 _id: "replicaconfig",
 configsvr: true,
 members: [{ _id: 0, host: "localhost:27019" }]
})
rs.status() //verification
```

3) Démarrer les shards et initialiser leurs Replica Sets

On démarre ensuite les deux shards (ports 20004 et 20005), puis on initialise leurs replica sets respectifs.

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" }]
})
```

4) Démarrer mongos et rattacher les shards au cluster

On lance le routeur mongos, on s'y connecte, puis on ajoute les deux shards au cluster :

```
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
```

5) Activer le sharding (base + collection)

On active le sharding sur la base, puis sur la collection films avec la shard key titre :

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

6) Charger les données et observer le cluster

On exécute ensuite le script Python d'insertion :

```
python3 monappunparun.py
```

Suivi de l'état du cluster :

```
sh.status()
```

Vérification de la distribution des données :

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

Observation des chunks :

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

Vérification du balancer :

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

7) Interprétation (chunks, split, balancing)

Au début, la collection shardée a souvent un seul chunk sur un shard. Quand on ajoute beaucoup de données, MongoDB coupe ce chunk en plusieurs petits chunks (split). Ensuite, le balancer déplace des chunks vers d'autres shards pour que les données soient mieux réparties. Si la shard key répartit mal les insertions (par exemple une clé qui augmente toujours), un seul shard reçoit presque toutes les écritures (hot shard), et le sharding devient moins utile.

II- Questions

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

Le sharding consiste à répartir les données d'une collection sur plusieurs serveurs (shards). On l'utilise quand une seule machine ne suffit plus, pour augmenter la capacité de stockage et mieux gérer la charge.

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

La réplication duplique les mêmes données sur plusieurs serveurs pour la disponibilité et la tolérance aux pannes. Le sharding, lui, découpe les données et les répartit entre serveurs pour monter en charge.

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

Les shards stockent les données. Les config servers gardent les informations de répartition. Le mongos est le routeur : c'est lui qui reçoit les requêtes et les envoie au bon endroit.

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

Ils stockent la configuration du sharding : quelle collection est shardée, la shard key, les chunks, et sur quels shards ils se trouvent. Sans ces infos, mongos ne peut pas router correctement.

5. Quel est le rôle du mongos router ?

Il sert de point d'entrée pour les applications. Il analyse la requête, consulte les métadonnées et contacte uniquement le(s) shard(s) nécessaires, puis renvoie le résultat.

6. Comment MongoDB décide-t-il sur quel shard stocker un document ?

MongoDB regarde la valeur de la shard key du document, trouve le chunk correspondant, puis écrit le document sur le shard qui possède ce chunk.

7. Qu'est-ce qu'une clé de sharding et pourquoi est-elle essentielle ?

La clé de sharding (shard key) est le champ qui sert à découper la collection en morceaux. Elle est essentielle car elle détermine la répartition des données et l'efficacité du routage des requêtes.

8. Quels sont les critères de choix d'une bonne clé de sharding ?

Elle doit bien répartir les données (éviter que tout tombe sur un seul shard), avoir beaucoup de valeurs différentes, et idéalement être utilisée dans les requêtes fréquentes. Elle doit aussi limiter les "hotspots" d'écriture.

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

Un chunk est une "tranche" de données (une plage de shard key) stockée sur un shard. C'est l'unité que MongoDB déplace lors du rééquilibrage.

10. Comment fonctionne le splitting des chunks ?

Quand un chunk devient trop gros, MongoDB le divise en plusieurs chunks plus petits. Cela facilite ensuite la répartition et les migrations.

11. Que fait le balancer dans un cluster shardé ?

Le balancer surveille la répartition des chunks et cherche à équilibrer les shards. Il évite qu'un shard soit beaucoup plus chargé que les autres.

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

Quand il détecte un déséquilibre, il migre un ou plusieurs chunks d'un shard trop chargé vers un shard moins chargé. La migration copie les données puis met à jour les métadonnées.

13. Qu'est-ce qu'un hot shard et comment l'éviter ?

Un hot shard est un shard qui reçoit presque toute la charge (souvent les écritures). Pour l'éviter, il faut une shard key qui répartit bien les insertions (parfois une shard key hashed).

14. Quels problèmes une clé de sharding monotone peut-elle engendrer ?

Si la clé augmente toujours (date, compteur…), les nouvelles insertions arrivent au même endroit, ce qui crée un hotspot. Résultat : un shard travaille beaucoup plus que les autres.

15. Comment activer le sharding sur une base de données et sur une collection ?

On active d'abord sur la base avec sh.enableSharding("db"). Ensuite on shard la collection avec sh.shardCollection("db.collection", { champ: 1 }) (souvent après avoir créé l'index sur ce champ).

16. Comment ajouter un nouveau shard à un cluster MongoDB ?

On démarre un nouveau shard (souvent un replica set) puis, via mongos, on exécute sh.addShard("replica/host:port"). Le balancer pourra ensuite redistribuer des chunks.

17. Comment vérifier l'état du cluster shardé (commandes usuelles) ?

On utilise surtout sh.status() pour la vue globale. On peut aussi vérifier la distribution avec db.collection.getShardDistribution() et consulter config.chunks pour voir les chunks.

18. Dans quels cas faut-il envisager d'utiliser un hashed sharding key ?

Quand on a beaucoup d'écritures et qu'on veut répartir uniformément les insertions sur les shards. C'est utile pour éviter les hotspots, mais moins adapté aux requêtes par intervalle.

19. Dans quels cas faut-il privilégier un ranged sharding key ?

Quand on fait souvent des requêtes par plages (ex : dates entre deux valeurs). Cela permet de cibler une partie des shards, mais il faut éviter les clés trop monotones.

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

Le zone sharding permet d'imposer que certaines plages de shard key aillent sur certains shards. C'est pratique pour séparer des données par région, client, ou contrainte de localisation.

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

Si la requête ne contient pas la shard key (ou pas assez), mongos envoie la requête à plusieurs shards, récupère les résultats et les regroupe avant de répondre.

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

Il faut des requêtes qui filtrent sur la shard key, des index adaptés, et éviter les opérations qui forcent un passage sur tous les shards (gros scans, tris globaux, etc.).

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

Si le shard est en replica set et qu'il reste un nœud disponible, le service peut continuer après élection. Si tout le shard est indisponible, les données qui y sont stockées ne sont plus accessibles.

24. Comment migrer une collection existante vers un schéma shardé ?

On active le sharding sur la base, on crée l'index sur la shard key, puis on lance sh.shardCollection(...). MongoDB gère ensuite la création des chunks et la redistribution via le balancer.

25. Quels outils ou métriques utiliser pour diagnostiquer les problèmes de sharding ?

sh.status() et getShardDistribution() donnent une première vue. Les collections de la base config (chunks, shards) aident à comprendre la répartition, et explain() permet de voir si une requête cible bien un shard ou tout le cluster.