TP 3 : Partitionnement avec MongoDB :

L'objectif de ce TP est de mettre en place un cluster MongoDB shardé, afin de comprendre :

- comment MongoDB répartit les données sur plusieurs serveurs,
- comment il passe à l'échelle horizontalement,
- et comment il équilibre la charge automatiquement.

Le cluster mis en place est composé de :

- 1 config server (stocke les métadonnées du sharding),
- 1 routeur mongos (point d'entrée des clients),
- 2 shards, chacun étant un replica set.

Réplication : tolérance aux pannes (copies des mêmes données)
Sharding : montée en charge (répartition des données)

Les deux sont complémentaires, pas concurrents.

Je démarre mon cluster :

```
PS C:\Users\Fei3\downloads\tp-sharding-mongo> docker compose up -d
```

J'initialise le config server :

```
PS C:\Users\Fei3\downloads\tp-sharding-mongo> docker exec -it configsvr mong
osh --port 27019
```

```
test> rs.initiate()
```

J'initialise le shard 1 :

```
PS C:\Users\Fei3\downloads\tp-sharding-mongo> docker exec -it shard1 mongosh
 --port 27018
```

```
test> rs.initiate()
```

J'initialise le shard 2 :

```
PS C:\Users\Fei3\downloads\tp-sharding-mongo> docker exec -it shard2 mongosh
 --port 27020
```

```
test> rs.initiate()
```

Je me connecte au routeur mongos :

```
PS C:\Users\Fei3\downloads\tp-sharding-mongo> docker exec -it mongos mongosh
 --port 27017
```

J’ajoute les shards :

```
[direct: mongos] test> sh.addShard("replicashard1/shard1:27018")
```

```
[direct: mongos] test> sh.addShard("replicashard2/shard2:27020")
```

J’active le sharding sur la base :

```
[direct: mongos] test> sh.enableSharding("lesfilms")
```

Je crée l’index :

```
[direct: mongos] test> use lesfilms
switched to db lesfilms
[direct: mongos] lesfilms> db.films.createIndex({ titre: 1 })
titre_1
```

Je shard la collection :

```
[direct: mongos] lesfilms> sh.shardCollection("mabasefilms.films", { titre:
1 })
```

Insérer des films :

```
[direct: mongos] lesfilms> for (let i = 0; i < 100000; i++) {
...   db.films.insertOne({
...     titre: "Film_" + i,
...     annee: 2000 + (i % 20),
...     genre: ["Genre_" + (i % 5)]
...   })
... }
```

Je vérifie le sharding :

Extrait :

```
[direct: mongos] lesfilms> sh.status()
shardingVersion
{ _id: 1, clusterId: ObjectId('693eb61bf6cd4c051aed8fef') }
---
shards
[
  {
    _id: 'replicashard1',
    host: 'replicashard1/shard1:27018',
    state: 1,
    topologyTime: Timestamp({ t: 1765717939, i: 2 })
  },
  {
    _id: 'replicashard2',
    host: 'replicashard2/shard2:27020',
    state: 1,
    topologyTime: Timestamp({ t: 1765718228, i: 2 })
  }
]
```

Je vérifie la répartition :

```
[direct: mongos] lesfilms> db.films.getShardDistribution()
Shard replicashard2 at replicashard2/shard2:27020
{
  data: '10.66MiB',
  docs: 136699,
  chunks: 1,
  'estimated data per chunk': '10.66MiB',
  'estimated docs per chunk': 136699
}
---
Totals
{
  data: '10.66MiB',
  docs: 136699,
  chunks: 1,
  'Shard replicashard2': [
    '100 % data',
    '100 % docs in cluster',
    '81B avg obj size on shard'
  ]
}
```

Questions :

1/ Le sharding est un mécanisme de partitionnement horizontal des données consistant à répartir les documents d'une collection sur plusieurs serveurs appelés shards. Il est utilisé

afin d'améliorer la scalabilité, de répartir la charge et de permettre à MongoDB de gérer de très grands volumes de données et de requêtes.

2/ La réplication vise la haute disponibilité en copiant les mêmes données sur plusieurs nœuds, tandis que le sharding vise la scalabilité horizontale en répartissant les données entre plusieurs serveurs distincts. Les deux mécanismes sont complémentaires.

3/ Une architecture shardée comprend :

- les shards : serveurs stockant les données,
- les config servers (CSRS) : stockent les métadonnées du sharding,
- le mongos : routeur recevant les requêtes des clients et les redirigeant.

4/ Les config servers stockent :

- les métadonnées de partitionnement,
- la répartition des chunks,
- les informations nécessaires au routage des requêtes.
  Ils sont indispensables au bon fonctionnement du cluster shardé.

5/ Le mongos agit comme un routeur intelligent. Il reçoit les requêtes des clients, consulte les config servers, puis redirige chaque requête vers le ou les shards concernés.

6/ MongoDB utilise la clé de sharding du document. En fonction de la valeur de cette clé et des plages de chunks définies, MongoDB détermine sur quel shard stocker le document.

7/ La clé de sharding est un champ (ou un ensemble de champs) utilisé pour répartir les documents entre les shards. Elle est essentielle car elle influence directement la distribution des données, les performances et l'équilibrage du cluster.

8/ Une bonne clé de sharding doit :

- avoir une forte cardinalité,
- assurer une distribution uniforme,
- être fréquemment utilisée dans les requêtes,
- éviter les valeurs monotones.

9/ Un chunk est une plage de valeurs de la clé de sharding regroupant un ensemble de documents. Les chunks constituent l'unité de base de distribution des données entre les shards.

10/ Lorsqu'un chunk dépasse une taille prédéfinie, MongoDB le divise automatiquement en plusieurs chunks plus petits afin de maintenir une répartition équilibrée.

11/ Le balancer est un processus automatique qui :

- surveille la distribution des chunks,
- déplace les chunks entre les shards,
- assure un équilibrage de la charge.

12/ Le balancer déplace des chunks lorsque certains shards sont surchargés. Il migre les chunks vers des shards moins chargés, généralement en arrière-plan.

13/ Un hot shard est un shard recevant une charge excessive. Il peut être évité en choisissant une clé de sharding bien distribuée, ou en utilisant une clé hashée.

14/ Une clé monotone (ex. date croissante) entraîne :

- des insertions concentrées sur un seul shard,
- un déséquilibre de charge,
- la création de hot shards.

15/ Activer le sharding sur la base :

sh.enableSharding("mabasefilms")

Sharder la collection :

sh.shardCollection("mabasefilms.films", { titre: 1 })

16/ sh.addShard("replicaSet/host:port")

17/ sh.status()

db.collection.getShardDistribution()

18/ Une clé hashée est utilisée lorsque :

- les insertions sont très fréquentes,
- on souhaite une distribution uniforme,
- l’ordre des données n’est pas important.

19/ Une clé ranged est préférable lorsque :

- les requêtes utilisent des plages de valeurs,
- l’ordre naturel des données est important.

20/ Le zone sharding permet d’associer des plages de valeurs à des shards spécifiques. Il est utile pour respecter des contraintes géographiques ou organisationnelles.

21/ Le mongos envoie la requête à tous les shards concernés, agrège les résultats, puis renvoie une réponse unique au client.

22/ On peut optimiser les performances en :

- utilisant la clé de sharding dans les requêtes,
- créant des index appropriés,
- limitant les requêtes multi-shards.

23/ Si un shard est indisponible, MongoDB s’appuie sur la réplication pour maintenir l’accès aux données. Sinon, les données hébergées sur ce shard deviennent temporairement inaccessibles.

24/ Il faut :

1. activer le sharding sur la base,
2. définir une clé de sharding,
3. exécuter sh.shardCollection,
   MongoDB répartit ensuite progressivement les données existantes.

25/ On utilise :

- sh.status(),
- db.chunks,
- getShardDistribution(),
- les logs MongoDB,
- les métriques de charge CPU et mémoire.