Bases de Données NoSQL - TP3

Prénom: Abdelillah

Nom: SELHI

N°Étudiant: 12206456

Groupe: INFOA3

1. C’est quoi MongoDB?

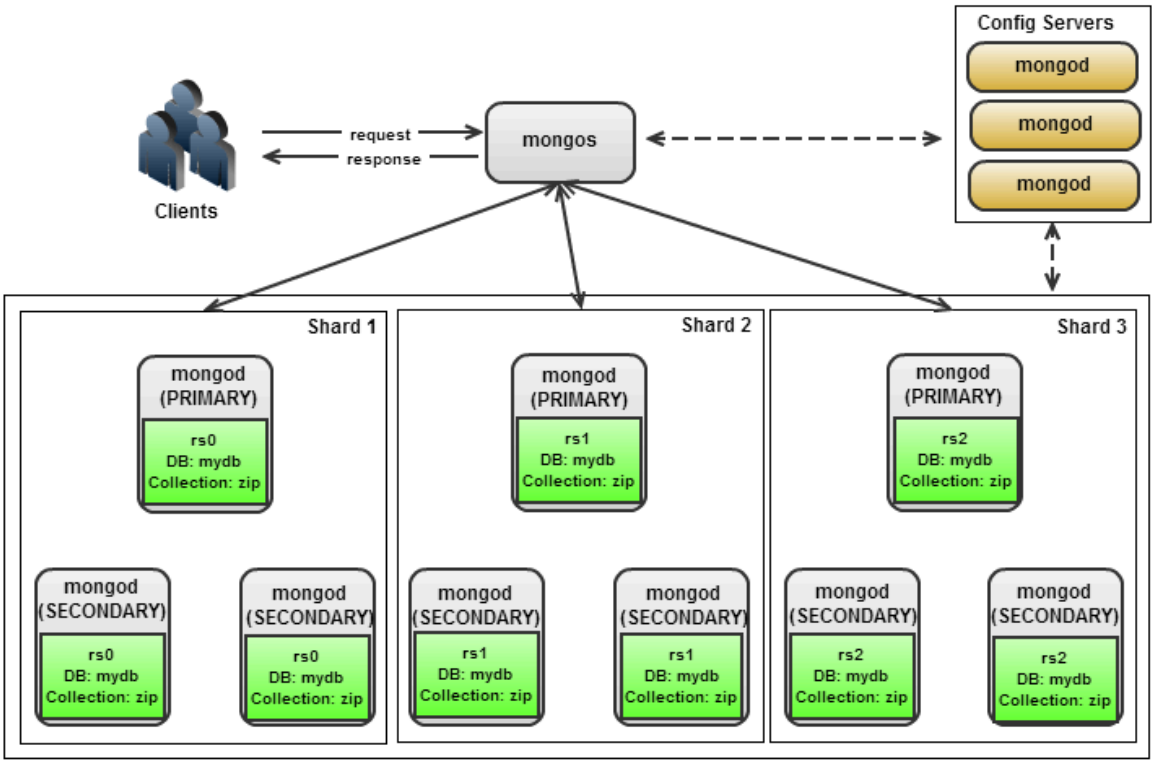
MongoDB est un système de gestion de base de données **NoSQL orienté document**. Il stocke les données sous forme de documents BSON (Binary JSON), qui sont des structures de données flexibles (similaires à des objets JSON) regroupées dans des "collections". **Contrairement aux bases de données relationnelles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

Dans les systèmes **NoSQL**, l’objectif principal est de soutenir des charges importantes (volume de données, débit de lectures/écritures, pics de trafic) tout en conservant une **disponibilité** élevée.

Deux mécanismes clés structurent cette approche :

- **La réplication**, qui **duplique** les mêmes données sur plusieurs nœuds afin d’améliorer la tolérance aux pannes et la continuité de service.
- **Le sharding (partitionnement horizontal)**, qui découpe le jeu de données en fragments distribués sur plusieurs serveurs afin d’augmenter la capacité totale de stockage et de traitement.

La réplication vise avant tout la haute disponibilité : un nœud primaire accepte les écritures et réplique les opérations vers des nœuds secondaires, permettant un basculement automatique en cas de panne du primaire. À l’inverse, **le sharding vise la scalabilité** : les documents d’une collection sont répartis entre plusieurs shards selon une shard key, ce qui permet de paralléliser stockage et charge au lieu de concentrer toutes les opérations sur une seule machine. En pratique, ces mécanismes sont complémentaires : la réplication protège contre la perte de service/données, tandis que le sharding répond à la croissance du dataset et au besoin d’augmenter le débit global du système.



2- Sharding avec MongoDB

a- création du docker compose

```yaml
services:
  ▷ Run Service
  configsvr:
    image: mongo
    container_name: configsvr
    command: ["mongod","--configsvr","--replSet","replicaconfig","--dbpath","/data/db","--port","27019","--bind_ip_all"]
    ports:
      - "27019:27019"
    volumes:
      - configsvrdb:/data/db

  ▷ Run Service
  shard1:
    image: mongo
    container_name: shard1
    command: ["mongod","--shardsvr","--replSet","replicashard1","--dbpath","/data/db","--port","20004","--bind_ip_all"]
    ports:
      - "20004:20004"
    volumes:
      - shard1db:/data/db

  ▷ Run Service
  shard2:
    image: mongo
    container_name: shard2
    command: ["mongod","--shardsvr","--replSet","replicashard2","--dbpath","/data/db","--port","20005","--bind_ip_all"]
    ports:
      - "20005:20005"
    volumes:
      - shard2db:/data/db

  ▷ Run Service
  mongos:
    image: mongo
    container_name: mongos
    depends_on:
      - configsvr
      - shard1
      - shard2
    command: ["mongos","--configdb","replicaconfig/configsvr:27019","--bind_ip_all","--port","27020"]
    ports:
      - "27020:27020"

volumes:
  configsvrdb: {}
  shard1db: {}
  shard2db: {}
```

Ce fichier décrit une mini-architecture de sharding MongoDB: un conteneur **configsvr** lance mongod en mode serveur de configuration avec un replica set nommé replicaconfig et écoute sur le port 27019 pour stocker les métadonnées de sharding (localisation des chunks, configuration du cluster). **Deux conteneurs shard1 et shard2** lancent chacun mongod en mode shard (–shardsvr) avec leurs replica sets respectifs replicashard1 (port 20004) et replicashard2 (port 20005). Enfin, le conteneur **mongos** démarre le routeur (mongos) en se connectant au config server via --configdb replicaconfig/configsvr:27019, expose le port 27020, et sert de point d'entrée unique.

b- créer les conteneur en executant le docker compose

avec la commande **docker compose up -d**

```
abdelillah@abdelillah-HP-250-G9:~/Documents/INFOA3/NoSQL/NoSQL/tp3$ docker c
ompose up -d
[+] Running 8/8
 ✔ Network tp3_default      Created                                    0.1s
 ✔ Volume tp3_shard2db      Created                                    0.0s
 ✔ Volume tp3_configsvrdb   Created                                    0.0s
 ✔ Volume tp3_shard1db      Created                                    0.1s
 ✔ Container shard1         Started                                    2.2s
 ✔ Container configsvr      Started                                    2.2s
 ✔ Container shard2         Started                                    2.7s
 ✔ Container mongos         Started                                    3.0s
abdelillah@abdelillah-HP-250-G9:~/Documents/INFOA3/NoSQL/NoSQL/tp3$
```

c- Vérifier les conteneur en marche avec docker ps:

```
abdelillah@abdelillah-HP-250-G9:~/Documents/INFOA3/NoSQL/NoSQL/tp3$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS                                                 NAMES
4a90deac4567   mongo     "docker-entrypoint.s…"   2 minutes ago   Up 2 minutes   0.0.0.0:27020->27020/tcp, [::]:27020->27020/tcp       mongos
aa8f8d7be54c   mongo     "docker-entrypoint.s…"   2 minutes ago   Up 2 minutes   0.0.0.0:20004->20004/tcp, [::]:20004->20004/tcp       shard1
eef0ab95248b   mongo     "docker-entrypoint.s…"   2 minutes ago   Up 2 minutes   0.0.0.0:27019->27019/tcp, [::]:27019->27019/tcp       configsvr
85db94a5faff   mongo     "docker-entrypoint.s…"   2 minutes ago   Up 2 minutes   0.0.0.0:20005->20005/tcp, [::]:20005->20005/tcp       shard2
```

On remarque qu'on bien le **mongos**, **config server**, **shard1** et **shard2**

d- Initialiser le replica set du config server

Il faut d'abord se connecter au config server avec la commande:
docker exec -it configsvr mongosh --port 27019

puis exécuter:

```
test> rs.initiate({ _id: "replicaconfig", configsvr: true, members: [{ _id: 0, host: "configsvr:27019" }] })
{
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1765901942, i: 1 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1765901942, i: 1 })
}
replicaconfig [direct: secondary] test>
```

Puis vérifier le status avec **rs.status()**

e- Initialiser les replica sets des shards

Shard1:

docker exec -it shard1 mongosh --port 20004
rs.initiate({ _id: "replicashard1", members: [{ _id: 0, host: "shard1:20004" }] })
rs.status()

Shard2:

docker exec -it shard2 mongosh --port 20005
rs.initiate({ _id: "replicashard2", members: [{ _id: 0, host: "shard2:20005" }] })
rs.status()

f- Se connecter à mongos et ajouter les shards

se connecter avec: **docker exec -it mongos mongosh --port 27020**
puis ajouter les shards:

```
[direct: mongos] test> sh.addShard("replicashard1/shard1:20004")
...
{
  shardAdded: 'replicashard1',
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1765902506, i: 57 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1765902506, i: 57 })
}
[direct: mongos] test> sh.addShard("replicashard2/shard2:20005")
...
{
  shardAdded: 'replicashard2',
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1765902514, i: 26 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1765902514, i: 26 })
}
```

puis vérifier l'ajout avec **sh.status()**

```
[direct: mongos] test> sh.status()
...
shardingVersion
{
  _id: ObjectId('69418676a8a09d869fe7ce6b'),
  clusterId: ObjectId('69418676a8a09d869fe7ce6a')
}
---
shards
[
  {
    _id: 'replicashard1',
    host: 'replicashard1/shard1:20004',
    state: 1,
    topologyTime: Timestamp({ t: 1765902506, i: 32 }),
    replSetConfigVersion: Long('-1')
  },
  {
    _id: 'replicashard2',
    host: 'replicashard2/shard2:20005',
    state: 1,
    topologyTime: Timestamp({ t: 1765902514, i: 3 }),
    replSetConfigVersion: Long('-1')
  }
]
```

g- Activer le sharding sur la base

```
[direct: mongos] test> sh.enableSharding("mabasefilms")
...
{
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1765902877, i: 21 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1765902877, i: 21 })
}
```

h- Sharder la collection films (clé: titre)

```
[direct: mongos] test> sh.shardCollection("mabasefilms.films", { "titre": 1 })
{
  collectionsharded: 'mabasefilms.films',
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1765902955, i: 37 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1765902955, i: 36 })
}
```

i- Lancer l'insertion (script Python) via mongos

Dans monappunparun.py, pour viser mongos il faut remplacer mongodb://localhost:27017/ par mongodb://localhost:27020/

selon les versions de python, il faudra peut être installer le module pymongo:

```
abdelillah@abdelillah-HP-250-G9:~/Téléchargements$ python3 -m pip install pymongo --break-system-packages
Defaulting to user installation because normal site-packages is not writeable
Collecting pymongo
  Downloading pymongo-4.15.5-cp312-cp312-manylinux2014_x86_64.manylinux_2_17_x86_64.manylinux_2_28_x86_64.whl.metadata (22 kB)
Collecting dnspython<3.0.0,>=1.16.0 (from pymongo)
  Downloading dnspython-2.8.0-py3-none-any.whl.metadata (5.7 kB)
Downloading pymongo-4.15.5-cp312-cp312-manylinux2014_x86_64.manylinux_2_17_x86_64.manylinux_2_28_x86_64.whl (1.7 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.7/1.7 MB 6.8 MB/s eta 0:00:00
Downloading dnspython-2.8.0-py3-none-any.whl (331 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 331.1/331.1 kB 11.1 MB/s eta 0:00:00
Installing collected packages: dnspython, pymongo
Successfully installed dnspython-2.8.0 pymongo-4.15.5
```

Puis on peut utiliser la fonction **getShardDestribution()** sur la collection pour voir que le sharding a été bien effectué entre Shard1 et Shard2

par exemple elle te montre le nombre de doc par shard:

Shard 2:

mabasefilms> db.films.getShardDistribution()
Shard replicashard2 at replicashard2/shard2:20005
{
data: '106.85MiB',
docs: 861911,
chunks: 1,
'estimated data per chunk': '106.85MiB',
'estimated docs per chunk': 861911
}

Questions / Réponses

1. Qu'est-ce que le sharding dans MongoDB et pourquoi est-il utilisé ?
Le sharding est un mécanisme qui découpe les données en fragments distribués sur plusieurs serveurs. Il est utile pour augmenter la capacité totale de stockage et de traitement et vise la scalabilité, un objectif principal des systèmes de bases de données NoSQL.

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

- **Réplication :** Vise la haute disponibilité en dupliquant les mêmes données sur plusieurs nœuds pour améliorer la tolérance aux pannes et la continuité de service.
- **Sharding :** Vise la scalabilité en répartissant les documents d'une collection entre plusieurs shards selon une clé de sharding ce qui permet de paralléliser le stockage et la charge.
  Les deux mécanismes sont complémentaires : la réplication protège contre la perte de service/données, tandis que le sharding répond à la croissance du dataset.

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

- **Config Servers (configsvr) :** Lance mongod en mode serveur de configuration pour stocker les métadonnées de sharding.
- **Shards (shard1 et shard2) :** Lancent chacun mongod en mode shard (–shardsvr) avec leurs replica sets respectifs. stockent les données.
- **Routeur Mongos (mongos) :** Démarre le routeur et sert de point d'entrée unique au cluster, en se connectant au config server.

4. Quelles sont les responsabilités des config servers (CSRS) dans un cluster shardé ?
Le serveur de configuration (**configsvr**) est responsable du stockage des métadonnées de sharding, qui incluent la localisation des **chunks** et la configuration du cluster.

5. Quel est le rôle du mongos router ?
Le routeur **mongos** sert de point d'entrée unique au cluster shardé pour les applications clientes et voir le ou les shards qui ont les données nécessaires en consultant les serveurs de configuration.

6. Comment MongoDB décide-t-il sur quel shard stocker un document ?
Les documents d'une collection sont répartis entre plusieurs shards selon une shard key, La valeur de cette clé est utilisée pour déterminer le bloc où va se trouver la donées.

7. Qu'est-ce qu'une clé de sharding et pourquoi est-elle essentielle ?
La clé de sharding (ou shard key) est l'élément qui permet de répartir les documents d'une collection entre les différents shards. Elle est essentielle car elle est la base de la distribution des données et du partitionnement.

8. Quels sont les critères de choix d'une bonne clé de sharding ?
Une clé efficace doit garantir une répartition équilibrée de la charge et éviter la surcharge d'un serveur unique (**hot shard**). donc parmi les critères: le nombre de valeurs distinctes, l'homogénéité de la distribution et autres.

9. Qu'est-ce qu'un chunk dans MongoDB ?
Il représente un sous-ensemble des documents d'une collection et est hébergé sur un shard.

10. Comment fonctionne le splitting des chunks ?
est un mécanisme automatique supervisé par mongos et les Config Servers. Lorsqu'un chunk atteint sa limite de taille MongoDB le divise en deux nouveaux chunks plus petits pour maintenir la taille de données.

11. Que fait le balancer dans un cluster shardé ?
un processus qui s'exécute sur le routeur. Son rôle est de surveiller le nombre de chunks sur chaque shard et d'assurer une distribution uniforme des données.

12. Quand et comment le balancer déplace-t-il des chunks ?
Le balancer se met en action lorsqu'un déséquilibre est détecté entre le nombre de chunks possédés par les shards. Il transfère les chunks des shards les plus chargés vers les moins chargés.

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

Un hot shard est un shard surchargé par rapport aux autres.
Prévention :

- Choisir une clé de sharding bien répartie.
- Utiliser une clé hachée si la clé est séquentielle.

14. Quels problèmes une clé de sharding monotone peut-elle engendrer ?
Une clé qui augmente toujours envoie toutes les écritures vers un seul shard, créant un hot shard.

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

- Activer le sharding sur la base (étape **g**). **sh.enableSharding("db")**
- Sharder la collection (étape **h**) en spécifiant une clé de sharding **sh.shardCollection("db.col", { clé: 1 })**

16. Comment ajouter un nouveau shard à un cluster MongoDB ?
l'étape consistant à se connecter à **mongos** et à ajouter les shards (étape **f**) est: **sh.addShard()**

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

- La commande **sh.status()**.

18. Dans quels cas faut-il envisager d'utiliser un hashed sharding key ?
Quand la clé naturelle est monotone et que l'on veut répartir uniformément les écritures.

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

Quand les requêtes utilisent souvent des intervalles de valeurs.

20. Qu'est-ce que le zone sharding et quel est son intérêt ?
Le zone sharding permet d'associer certaines données à des shards précis selon la clé de sharding. Il est utilisé pour contrôler la localisation des données ou séparer des charges de travail.

21. Comment MongoDB gère-t-il les requêtes multi-shards ?
Si une requête n'utilise pas la clé de sharding, mongos l'envoie à tous les shards, puis regroupe les résultats.

22. Comment optimiser les performances de requêtes dans un environnement shardé ?
Utiliser des requêtes avec la clé de sharding
Créer les bons index
Choisir une clé de sharding adaptée
Surveiller l'équilibrage et la charge

23. Que se passe-t-il lorsqu'un shard devient indisponible ?
(cas **répliqué**) Si le primaire tombe, un secondaire devient primaire par élection.
Si tout le shard est indisponible, seules les données de ce shard sont inaccessibles

24. Comment migrer une collection existante vers un schéma shardé ?
Activer le sharding sur la base puis définir la **clé de sharding** et ensuite MongoDB répartit ensuite les données automatiquement en appliquant des **migrations**.

25. Quels outils ou métriques utiliser pour diagnostiquer les problèmes de sharding ?
sh.status(): Donne une vue globale du cluster shardé : liste des shards, répartition des chunks. Permet de repérer rapidement un déséquilibre ou un hot shard.

config.chunks: La collection config.chunks indique pour chaque chunk sa plage de clés et le shard sur lequel il se trouve. Elle sert à analyser précisément la distribution des données.

Outils de monitoring et logs: Les outils de monitoring (MongoDB Atlas, Monitoring interne) permettent de suivre la charge, le trafic et les opérations par shard.

Les logs de mongos et des shards aident à identifier les erreurs, les migrations de chunks et les problèmes de performance.