

# Bases de données NoSQL

## Partitionnement



**Nom: HAMRANI**

**Prénom: Amziane**

## Introduction

Lors des TP précédents, nous avons étudié les **SGBD NoSQL** et leur rôle fondamental dans la **montée en charge (scalabilité horizontale)** et la **tolérance aux pannes**. Contrairement aux bases de données relationnelles classiques, dont la scalabilité repose principalement sur l'augmentation des ressources d'une seule machine (scalabilité verticale), les SGBD NoSQL sont conçus pour fonctionner sur **un ensemble de machines distribuées**.

Dans ce contexte, deux problématiques majeures ont été mises en évidence :

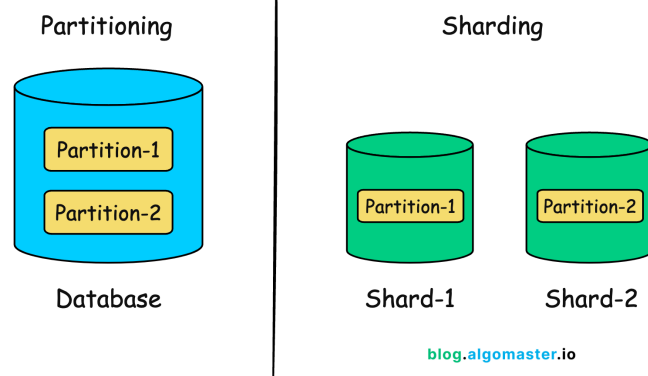
- la **disponibilité des données** en cas de panne matérielle ou logicielle ;
- la **capacité à absorber une augmentation du volume de données et du nombre de requêtes**.

Pour répondre à la première problématique, le TP précédent était consacré à la **réplication**. La réplication permet de maintenir plusieurs copies synchronisées des mêmes données sur différents nœuds, assurant ainsi la **reprise sur panne**, la **haute disponibilité** et, dans certains cas, une amélioration des performances en lecture. Dans MongoDB, cette fonctionnalité repose sur les **replica sets**, qui constituent l'unité de base de la tolérance aux pannes.

Cependant, la réplication seule ne permet pas de résoudre le problème de la croissance massive des volumes de données. En effet, même si les données sont répliquées, elles restent stockées de manière complète sur chaque nœud. C'est dans ce contexte qu'intervient le **partitionnement des données**, également appelé **sharding**.

De manière générale, le sharding consiste à **diviser un jeu de données en plusieurs fragments**, appelés *shards*, répartis sur différents serveurs. Chaque shard ne stocke qu'une partie des données, ce qui permet :

- de répartir la charge de stockage ;
- de paralléliser les traitements ;
- d'améliorer les performances globales du système ;
- de permettre une montée en charge quasi linéaire en ajoutant de nouveaux nœuds.



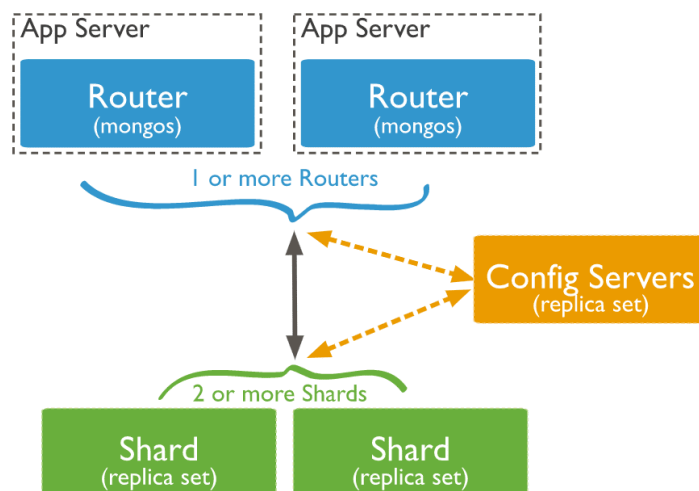
## Mongodb et le sharding:

Dans le contexte de **MongoDB**, le sharding est une fonctionnalité native permettant de distribuer automatiquement les documents d'une collection sur plusieurs shards, en fonction d'une **clé de sharding**. MongoDB fournit une architecture dédiée pour gérer ce partitionnement, reposant sur plusieurs composants spécialisés.

Un cluster MongoDB shardé s'appuie principalement sur :

- les **shards**, qui stockent les données (généralement sous forme de replica sets afin d'assurer la tolérance aux pannes) ;
- les **config servers**, qui conservent les métadonnées de partitionnement (emplacement des chunks, configuration du cluster, etc.) ;
- le **routeur mongos**, qui agit comme point d'entrée pour les clients et redirige les requêtes vers les shards appropriés.

L'objectif de ce TP est de **créer et configurer un cluster MongoDB shardé**, puis d'analyser le comportement du système lors de l'insertion et de la distribution des données. Contrairement au TP précédent, l'accent est mis ici sur le **partitionnement horizontal des données**, et non uniquement sur leur réplication. Dans ce TP, l'ensemble de l'architecture est déployé à l'aide de **Docker**, ce qui permet de simuler facilement un environnement distribué composé de plusieurs nœuds MongoDB sur une seule machine.



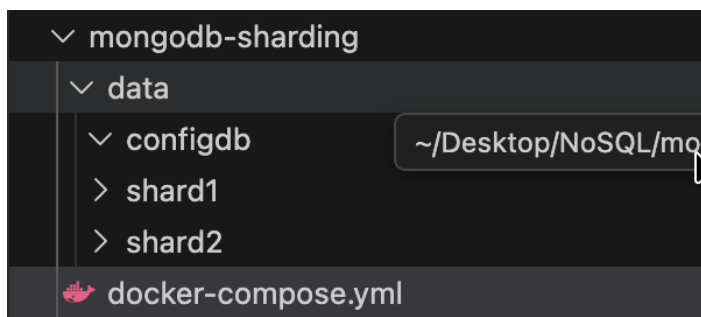
## Mise en place du cluster shardé MongoDB

Cette partie décrit les différentes étapes nécessaires à la création du cluster shardé, ainsi que le rôle de chaque commande exécutée. Ce document est une adaptation complète de ton TP de partitionnement MongoDB, afin que tout soit réalisable avec Docker (Docker + Docker Compose), sans lancer manuellement mongod dans plusieurs terminaux.

### Architecture cible (Docker)

- configsvr : Config Server (replica set replicaconfig)
- shard1 : Shard 1 (replica set replicashard1)
- shard2 : Shard 2 (replica set replicashard2)
- mongos : Routeur MongoDB

### Arborescence du projet



### Fichier docker-compose.yml

```
version: "3.8"

services:
  configsvr:
    image: mongo:7.0
    container_name: configsvr
    command: >
      mongod --configsvr --replSet replicaconfig --port 27019
    volumes:
      - ./data/configdb:/data/db
    ports:
      - "27019:27019"

  shard1:
    image: mongo:7.0
    container_name: shard1
    command: >
      mongod --shardsvr --replSet replicashard1 --port 27018
    volumes:
      - ./data/shard1:/data/db
    ports:
```

```

- "27018:27018"

shard2:
  image: mongo:7.0
  container_name: shard2
  command: >
    mongod --shardsvr --replSet replicashard2 --port 27017
  volumes:
    - ./data/shard2:/data/db
  ports:
    - "27017:27017"

mongos:
  image: mongo:7.0
  container_name: mongos
  depends_on:
    - configsvr
    - shard1
    - shard2
  command: >
    mongos --configdb replicaconfig/configsvr:27019 --port 27020
  ports:
    - "27020:27020"

```

## Démarrage du cluster

```

amzianehamrani@MacBook-Air-de-Amziane-2: ~/mongodb-sharding % docker compose up -d
WARN[0000] /Users/amzianehamrani/Desktop/NoSQL/mongodb-sharding/docker-compose.yml: the attribute 'version' is obsolete, it will be ignored, please remove it to avoid potential confusion
[+] Running 5/5
✔ Network mongodb-sharding_default Created                                0.0s
✔ Container shard2 Started                                                0.3s
✔ Container configsvr Started                                             0.4s
✔ Container shard1 Started                                                0.4s
✔ Container mongos Started                                                0.5s
amzianehamrani@MacBook-Air-de-Amziane-2: ~/mongodb-sharding % docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS                               NAMES
2f4ab554c7b4   mongo:7.0  "docker-entrypoint.s..." 53 seconds ago Up 52 seconds 27017/tcp, 0.0.0.0:27020->27020/tcp  mongos
82a35b64ac30   mongo:7.0  "docker-entrypoint.s..." 53 seconds ago Up 53 seconds 27017/tcp, 0.0.0.0:27018->27018/tcp  shard1
23c336010fa2   mongo:7.0  "docker-entrypoint.s..." 53 seconds ago Up 53 seconds 0.0.0.0:27017->27017/tcp             shard2
b4a08b6c443a   mongo:7.0  "docker-entrypoint.s..." 53 seconds ago Up 53 seconds 27017/tcp, 0.0.0.0:27019->27019/tcp  configsvr

```

## Initialisation des Replica Sets

- Config Server:

Connexion:

```
docker exec -it configsvr mongosh --port 27019
```

Création du RS:

```

test> rs.initiate({
...   _id: "replicaconfig",
...   members: [{ _id: 0, host: "configsvr:27019" }]
... })
{ ok: 1 }

```

- Shard 1

Connexion:

```
docker exec -it shard1 mongosh --port 27018
```

Création du RS:

```
test> rs.initiate({
... _id: "replicashard1",
... members: [{ _id: 0, host: "shard1:27018" }]
... })
{ ok: 1 }
```

- Shard 2

Connexion:

```
docker exec -it shard2 mongosh --port 27017
```

Création du RS:

```
test> rs.initiate({
... _id: "replicashard2",
... members: [{ _id: 0, host: "shard2:27017" }]
... })
{ ok: 1 }
```

## Ajout des shards au cluster

Connexion

```
docker exec -it mongos mongosh --port 27020
```

Ajout des shards

```
[direct: mongos] test> sh.addShard("replicashard1/shard1:27018")
{
  shardAdded: 'replicashard1',
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1765805668, i: 5 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1765805668, i: 5 })
}
[direct: mongos] test> sh.addShard("replicashard2/shard2:27017")
{
  shardAdded: 'replicashard2',
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1765805684, i: 14 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1765805684, i: 4 })
}
```

Vérification:

```
sh.status()
```

## Activation du sharding

- Activer le sharding sur la base:

```
[direct: mongos] test> sh.enableSharding("Films")
{
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1765805913, i: 9 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1765805913, i: 3 })
}
```

- Sharder la collection:

```
[direct: mongos] test> sh.shardCollection(
... "Films.films",
... { titre: 1 }
... )
...
{
  collectionssharded: 'Films.films',
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1765806004, i: 38 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1765806004, i: 38 })
}
```

## Insertion de documents:

- Exécution du script d'insertion:

```
python3 insert.py
```

- Vérification du sharding:

```
[direct: mongos] Films> db.films.getShardDistribution()
Shard replicashard1 at replicashard1/shard1:27018
{
  data: '34.26MiB',
  docs: 273929,
  chunks: 1,
  'estimated data per chunk': '34.26MiB',
  'estimated docs per chunk': 273929
}
---
Shard replicashard2 at replicashard2/shard2:27017
{
  data: '68.31MiB',
  docs: 822892,
  chunks: 1,
  'estimated data per chunk': '68.31MiB',
  'estimated docs per chunk': 822892
}
```

## Questions / Réponses:

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

Le sharding, ou partitionnement des données, consiste à **diviser un jeu de données en plusieurs fragments** appelés *shards*, qui sont répartis sur différents serveurs. Il est utilisé pour la **scalabilité horizontale** afin de gérer la croissance massive des volumes de données et l'augmentation du nombre de requêtes, ce que la réplication seule ne permet pas.

Le sharding permet de :

- Répartir la charge de stockage.
- Paralléliser les traitements.
- Améliorer les performances globales du système.
- Permettre une montée en charge quasi linéaire en ajoutant de nouveaux nœuds.

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

- **Réplication** : Maintient plusieurs **copies synchronisées des mêmes données** sur différents nœuds (via des *replica sets*), assurant la **tolérance aux pannes** et la **haute disponibilité**.
- **Sharding (Partitionnement)** : **Divise le jeu de données** en fragments distincts (*shards*), chaque *shard* ne stockant qu'une **partie des données**. Il est conçu pour la gestion de volumes de données massifs et la distribution de la charge.

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

Un cluster MongoDB shardé s'appuie principalement sur trois composants:

- **Shards** : Ce sont les nœuds qui **stockent les données** divisées (souvent des *replica sets* pour la tolérance aux pannes).
- **Config Servers (CSRS)** : Ils **conservent les métadonnées de partitionnement** du cluster, y compris l'emplacement des *chunks* et la configuration globale.
- **Routeur Mongos** : Il sert de **point d'entrée pour les clients**. Il dirige les requêtes des clients vers le ou les *shards* appropriés, en utilisant les métadonnées des *config servers*.

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

Les config servers (CSRS) sont responsables de la conservation des **métadonnées du partitionnement** (sharding). Ces métadonnées incluent :

- Le mappage des données (l'emplacement des *chunks* sur les différents *shards*).
- La configuration du cluster shardé.

## 5. Quel est le rôle du mongos router ?

Le routeur mongos agit comme un **proxy de requête**. Son rôle est de :

- Recevoir les requêtes des applications clientes.
- Déterminer le ou les *shards* qui détiennent les données nécessaires, en consultant les *config servers*.
- Router les requêtes vers les *shards* appropriés.
- Réassembler les résultats de plusieurs *shards* avant de les renvoyer au client (pour les requêtes multi-shards).

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

MongoDB utilise la **clé de sharding** (sharding key) définie pour la collection. La valeur de cette clé pour chaque document est utilisée pour déterminer le *chunk* (bloc de données) auquel appartient le document, et par conséquent, le *shard* sur lequel il sera stocké.

## 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)** dans les documents d'une collection. Elle est essentielle car :

- Elle détermine comment les données sont **divisées en chunks**.
- Elle dicte la **répartition des données** dans le cluster.
- Elle influence directement l'efficacité des opérations (lecture et écriture) et l'équilibre de la charge entre les *shards*.

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

Une bonne clé de sharding doit offrir une **cardinalité élevée** et une **fréquence de modification uniforme** pour assurer une distribution équilibrée de la charge et éviter les *hot shards* (shards surchargés). Les critères principaux sont :

- **Cardinalité** : Avoir un grand nombre de valeurs uniques.
- **Fréquence de modification** : Les valeurs de la clé ne doivent pas augmenter ou diminuer de manière monotone pour éviter les déséquilibres.
- **Granularité** : Permettre une division fine et homogène des données.
- **Pertinence de la requête** : Être incluse dans la plupart des requêtes pour les diriger efficacement vers un seul *shard* (requêtes ciblées).

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

Un *chunk* est une **plage contiguë de valeurs de clé de sharding** qui représente un sous-ensemble des données de la collection. Chaque *chunk* est stocké sur un *shard* spécifique et a une taille maximale par défaut (généralement 64 Mo).

## 10. Comment fonctionne le splitting des chunks ?

Le *splitting* (fractionnement) est un processus automatique géré par le mongos et les *config servers*. Lorsqu'un *chunk* atteint sa taille maximale prédéfinie (par défaut 64 Mo), MongoDB le **divise en deux chunks plus petits** pour garantir que la taille des données par *chunk* reste gérable. Ce processus s'exécute en arrière-plan sans interruption des opérations.

### 11. Que fait le balancer dans un cluster shardé ?

Le *balancer* est un processus qui s'exécute sur le routeur mongos et dont le rôle est d'assurer une **répartition uniforme des *chunks*** entre tous les *shards* du cluster. Il prévient les déséquilibres et les *hot shards* en effectuant des migrations de *chunks* entre les *shards*.

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

Le *balancer* se déclenche lorsqu'il détecte un déséquilibre significatif dans le nombre de *chunks* entre les *shards* (la différence de *chunks* dépasse un certain seuil). Il déplace les *chunks* des *shards* qui en ont trop vers ceux qui en ont moins. Le déplacement se fait en arrière-plan pour minimiser l'impact sur les performances.

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

Un *hot shard* (ou *shard* à chaud) est un **shard qui reçoit un volume disproportionné de requêtes en écriture et/ou en lecture** par rapport aux autres *shards*. Cela entraîne des problèmes de performance pour tout le cluster. Il peut être évité par :

- Le choix d'une clé de sharding avec une **cardinalité élevée** et une distribution uniforme des opérations.
- L'utilisation d'une **clé de sharding hachée (*hashed sharding key*)** si la clé naturelle est monotone (pour distribuer uniformément les écritures).

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

Une clé de sharding monotone (dont la valeur augmente ou diminue toujours, comme un timestamp ou un ID auto-incrémenté) dirige **toutes les nouvelles écritures vers le même *chunk***, et donc vers le **même *shard***. Cela crée un **hot shard** et annule l'avantage de l'écriture parallèle du sharding.

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

L'activation du sharding se fait via le routeur mongos :

- **Sur la base de données** : Utilisez la commande `sh.enableSharding("nom_de_la_base")`.
- **Sur la collection** : Utilisez la commande `sh.shardCollection("nom_de_la_base.nom_de_la_collection", { clé: 1 })`, où `{ clé: 1 }` définit la clé de sharding et son type.

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

L'ajout d'un nouveau *shard* se fait via le routeur mongos en utilisant la commande `sh.addShard()`. Le *balancer* se chargera ensuite automatiquement de déplacer les *chunks* existants vers ce nouveau *shard* pour rééquilibrer le cluster.

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

La commande la plus courante est `sh . status ( )`, qui fournit un résumé de l'état du cluster, incluant :

- Les *shards* configurés.
- L'état des *config servers*.
- L'état des bases de données et des collections *shardées*.
- Les informations sur le *balancer*.

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

Le *hashed sharding key* (clé de sharding hachée) est utilisé lorsque la clé de sharding naturelle est **monotone** (par exemple, une date ou un `_id` par défaut) et qu'il est nécessaire de distribuer les **écritures** de manière uniforme. Le hachage garantit une distribution aléatoire des données sur les *shards*, évitant les *hot shards* lors des insertions.

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

Le *ranged sharding key* (clé de sharding par plage) est privilégié lorsque les requêtes impliquent des **plages de valeurs** (*range queries*). Il permet à mongos de diriger la requête vers un seul *shard* ou un sous-ensemble limité de *shards* (requête ciblée), car les données avec des valeurs proches sont stockées ensemble.

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

Le *zone sharding* (ou *tag aware sharding*) permet d'associer des plages de *chunks* (des zones) à des *shards* spécifiques. Son intérêt est de :

- **Contrôler la localisation des données** (ex : stocker les données des clients européens sur des serveurs en Europe pour respecter les réglementations).
- **Isoler les charges de travail** (ex : dédier certains *shards* aux données des utilisateurs premium).

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

Pour une requête qui ne peut pas être ciblée sur un seul *shard* (par exemple, une requête sans la clé de sharding), le routeur mongos utilise le processus de **scatter-gather** :

1. **Scatter** : mongos diffuse la requête à **tous** les *shards* du cluster.
2. **Gather** : Chaque *shard* exécute la requête sur sa partie des données.
3. **Merge** : mongos collecte les résultats de tous les *shards*, les agrège, les trie si nécessaire, et renvoie le résultat final au client.

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

L'optimisation repose principalement sur :

- **Requêtes ciblées** : Assurer que la majorité des requêtes incluent la clé de sharding pour qu'elles puissent être dirigées vers un seul *shard* et éviter le *scatter-gather*.
- **Indexation** : Créer des index appropriés, en particulier sur la clé de sharding.
- **Clé de sharding efficace** : Choisir une clé qui répartit uniformément les données et correspond aux requêtes les plus fréquentes.

- **Surveillance** : Utiliser des outils de diagnostic (`sh.status()`, logs, etc.) pour identifier les déséquilibres et les *hot shards*.

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

Étant donné que chaque *shard* est généralement un *replica set*, l'indisponibilité d'un nœud ne cause pas la panne du *shard*.

- Si le **primaire** tombe en panne, un des secondaires prend le relais via le processus d'élection du *replica set*.
- Si le *shard* entier (le *replica set*) devient indisponible, les données stockées sur ce *shard* ne sont plus accessibles. Les requêtes ciblant ce *shard* échoueront ou retourneront une erreur, mais le reste du cluster continue de fonctionner (disponibilité partielle).

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

La migration d'une collection non *shardée* vers un schéma *shardé* se fait en deux étapes :

1. **Activer le sharding sur la base de données** (`sh.enableSharding()`).
2. **Partitionner la collection** en spécifiant la clé de sharding (`sh.shardCollection()`).

Une fois le sharding activé, MongoDB crée un *chunk* initial unique contenant toutes les données de la collection. Le *balancer* (ou une opération manuelle) commence ensuite à diviser ce *chunk* (processus de *splitting*) et à migrer les nouveaux *chunks* vers d'autres *shards* pour équilibrer la distribution.

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

Les principaux outils et commandes sont :

- **`sh.status()`** : Vue d'ensemble du cluster, des *shards*, des *chunks* et du *balancer*.
- **`db.printShardingStatus()`** : Alias de `sh.status()`.
- **`db.getSiblingDB("config").chunks.find()`** : Pour voir l'emplacement et la plage de clés de chaque *chunk*.
- **MongoDB Atlas / Monitoring de MongoDB** : Interfaces graphiques pour surveiller la charge, le trafic et la répartition des données entre les *shards*.
- **Logs du mongos et des *shards*** : Pour identifier les problèmes de performance, les échecs de migration ou les alertes de déséquilibre.