Bases de Données NoSQL - TP2

Prénom: Abdelillah

Nom: SELHI

N°Étudiant: 12206456

Groupe: INFOA3

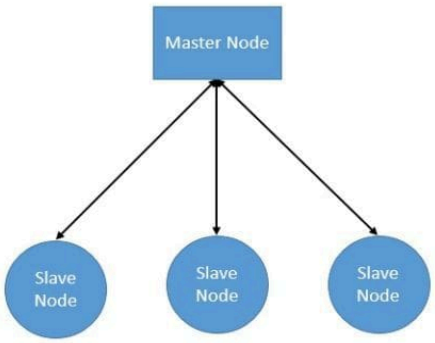
1. C’est quoi MongoDB?

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.

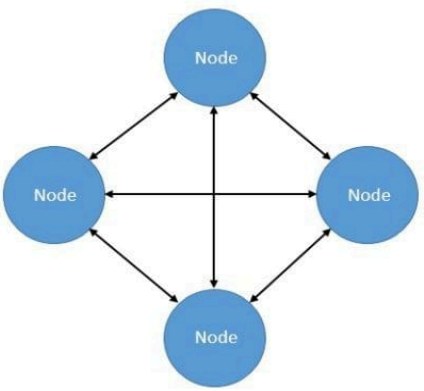
2. Architectures de Base de Données

Architecture Maître-Esclave vs Multi-nœuds

Caractéristique	**Maître-Esclave (Master-Slave)**	**Multi-Nœuds (Peer-to-Peer model)**
Rôles des Nœuds	Un seul nœud principal (Maître) gère les écritures. Les autres sont des Esclaves (Slaves) qui gèrent les lectures et répliquent les données.	Tous les nœuds peuvent potentiellement accepter des écritures.
Tolérance aux Pannes	Si le Maître tombe, le système d'écriture est interrompu jusqu'à l'élection d'un nouveau Maître.	Plus résilient. Si un nœud tombe, d'autres peuvent continuer à accepter des écritures.
Montée en Charge	Les lectures peuvent être distribuées, mais les écritures sont limitées par la capacité du Maître unique.	Meilleure distribution de la charge d'écriture sur tous les nœuds.



Master-Slave Model



Peer-to-Peer Model

Quel modèle utilise MongoDB?

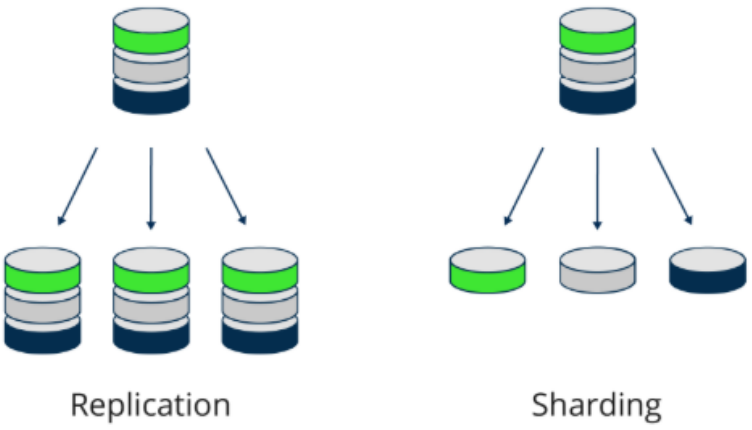
MongoDB utilise un modèle basé sur la **Réplication par Set de Répliques** (Replica Set). Un Replica Set est composé de plusieurs instances :

- **Primaire (équivalent du Maître) :** Un seul nœud accepte toutes les opérations d'écriture.
- **Secondaires (équivalents des Esclaves) :** Répliquent les données du Primaire et peuvent gérer les requêtes de lecture.
- **Élection :** Si le nœud Primaire tombe, les Secondaires élisent automatiquement un nouveau Primaire.

3. Stratégies de Montée en Charge

Réplication vs. Distribution (Sharding)

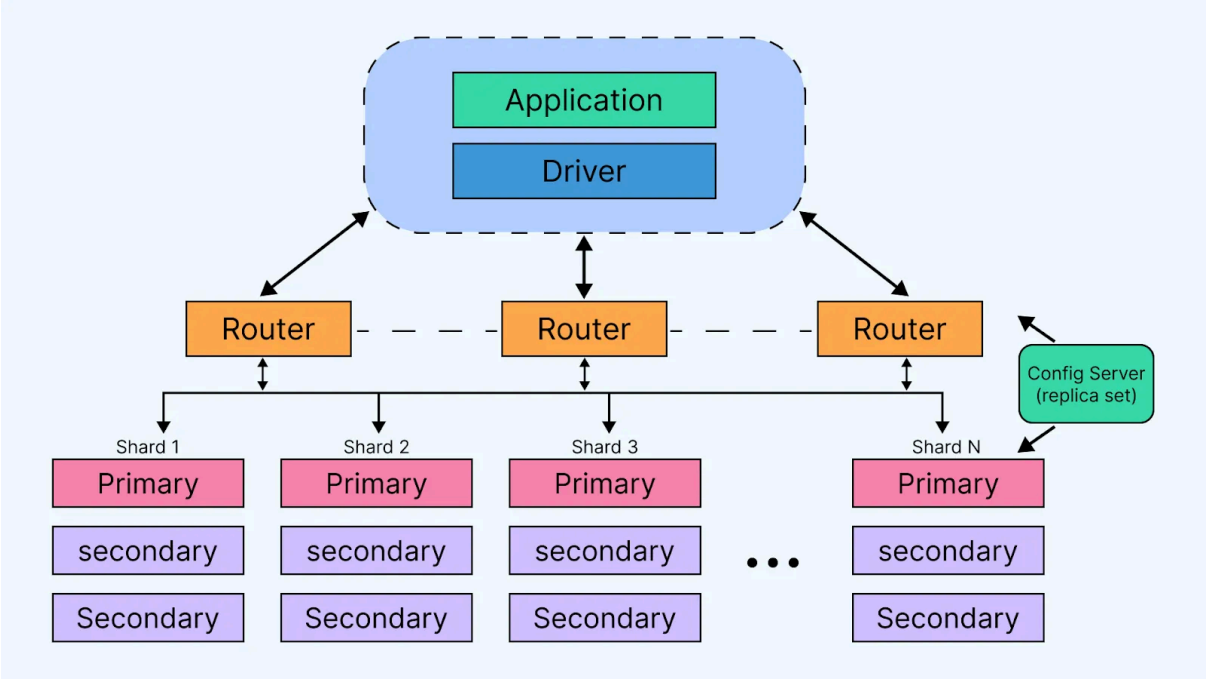
Stratégie	Définition	Objectif	Risques/Défis
Réplication	Copie des mêmes données sur plusieurs serveurs (nœuds).	**Haute Disponibilité** et **Tolérance aux Pannes** ; **Équilibrage de Charge** (distribution des lectures).	Coût en espace de stockage (duplication) ; Latence potentielle de la réplication (consistance éventuelle).
Distribution (Sharding)	Partitionnement des données en sous-ensembles (shards) stockés sur des serveurs distincts.	**Augmentation de la Capacité de Stockage** et **Amélioration des Performances d'Écriture/Lecture**.	Complexité de l'administration et de la distribution des données ; Jonctions de données plus complexes.



Quel modèle utilise MongoDB?

MongoDB utilise les deux stratégies pour la montée en charge :

1. **Réplication** via les **Replica Sets** (pour la haute disponibilité).
2. **Distribution** via le **Sharding** (pour l'évolutivité horizontale de la capacité et de la charge). Chaque "shard" est lui-même un Replica Set.

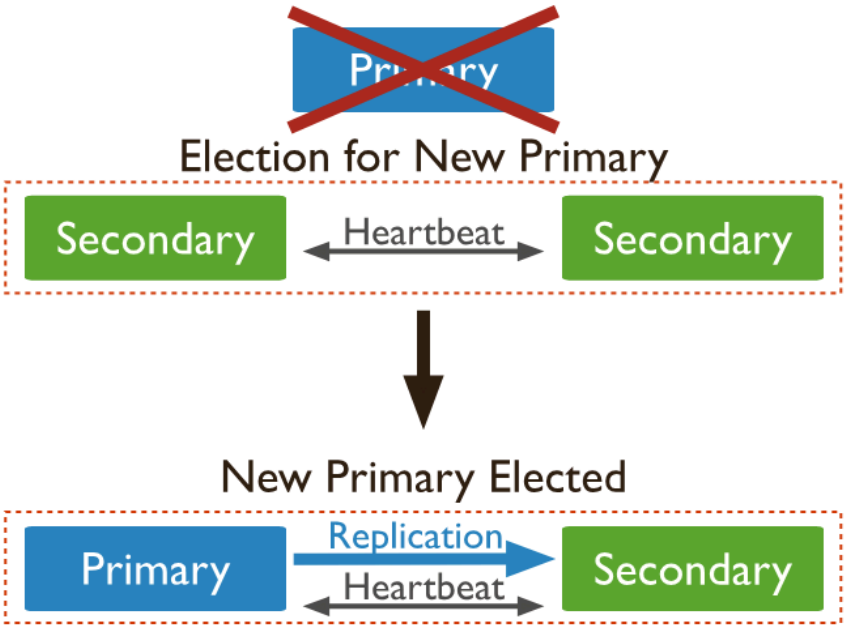


4. Tolérance aux Pannes

SI le maître tombe? (procédure d'élection)

Dans un Replica Set MongoDB, si le nœud **Primaire** tombe, les nœuds **Secondaires** initient immédiatement une **procédure d'élection** pour choisir un nouveau Primaire.

1. **Détection :** Les Secondaires détectent la panne du Primaire (absence de "heartbeats").
2. **Vote :** Les Secondaires éligibles se portent candidats et votent.
3. **Élection :** Le nœud qui obtient la majorité des voix devient le nouveau Primaire et commence à accepter les écritures.
4. **Basculement (Failover) :** L'application cliente est redirigée vers le nouveau Primaire.



5. Configuration de la Réplication

POUR configurer la réplication sur mongodb ?

La configuration implique de démarrer les instances de MongoDB avec l'option *Replica Set* activée, puis d'initialiser le set.

Un replica set MongoDB est un groupe de processus **mongod** qui maintiennent la même copie de données, avec un nœud primaire (master) qui accepte les écritures et un ou plusieurs nœuds secondaires (slave) qui répliquent ces données.

a. Démarrer les 3 nœuds MongoDB avec Docker

Paramètres nécessaires : Nom du Replica Set (**--replSet rs0**), Ports d'écoute (**27018, 27019, 27020**) et Répertoires de données (**disque1, disque2, disque3**).

Exemple de commande Docker:

```
abdelillah@abdelillah-HP-250-G9:~$ docker network create mongo-net
0ac1febb6aa3bc87bafffdf0b23c7dd3b47c8362d24ded971031c4c61eb0756c
abdelillah@abdelillah-HP-250-G9:~$ docker run -d --name mongo1 --network mon
go-net \
-p 27018:27017 \
-v $(pwd)/disque1:/data/db \
mongo --replSet rs0 --port 27017
47f1891323be6df303e277aaf3a321ef9d153ee573ef0348fe616c520676fcd5
abdelillah@abdelillah-HP-250-G9:~$ docker run -d --name mongo2 --network mon
go-net \
-p 27019:27017 \
-v $(pwd)/disque2:/data/db \
mongo --replSet rs0 --port 27017
90ee2d41d14ba01ad656abfd43e3d00505333403795ed67c53dafa3f6b73e73c
abdelillah@abdelillah-HP-250-G9:~$ docker run -d --name mongo3 --network mon
go-net \
-p 27020:27017 \
-v $(pwd)/disque3:/data/db \
mongo --replSet rs0 --port 27017
bfd2a65bbfd7281b32cccfa01318781df35dc7b17788996b9f13f33b05f03f64
```

b. Initialiser le Replica Set

- **Connexion au premier nœud :**
  docker exec -it mongo1 mongosh
- **Initialisation et ajout des autres nœuds :**

  rs.intiate()

  rs.add(“mongo2:27017”)

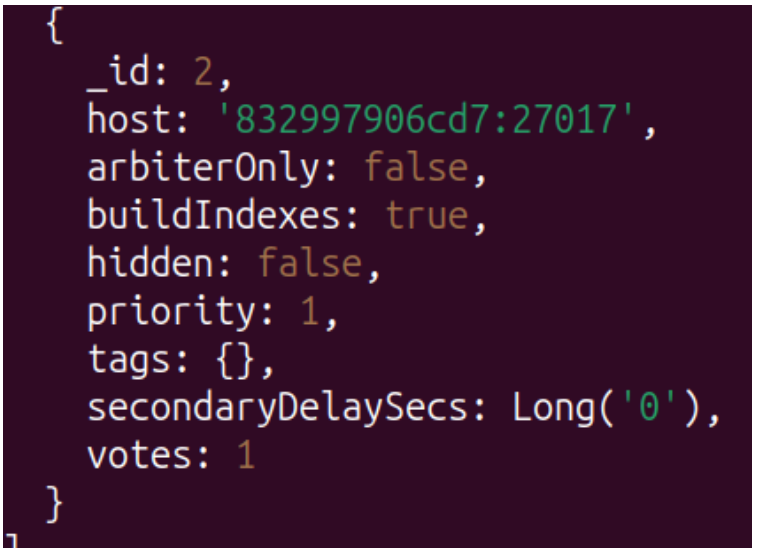
  rs.add(“mongo3:27017”)

6. Analyse de Configuration

Explique les champs… rs.config()

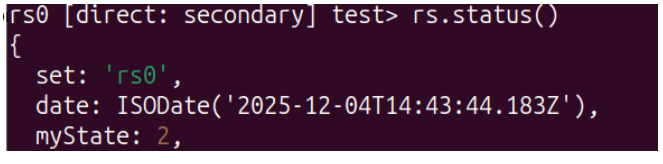
La commande rs.config() retourne le document de configuration du Replica Set. Les champs clés dans la section members sont :

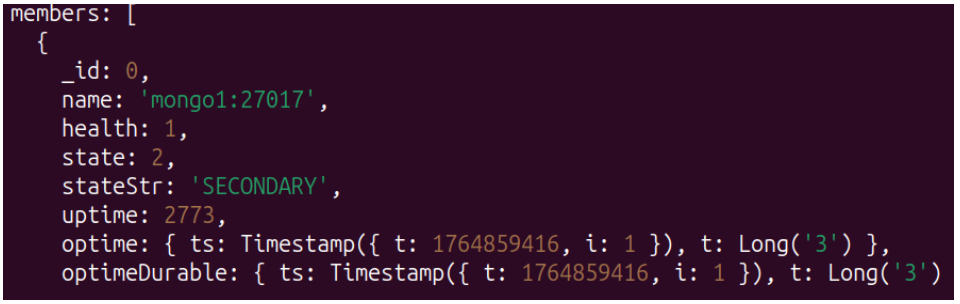
- **_id** : ID unique du membre au sein du set.
- **host** : Adresse réseau du membre (hostname:port).
- **priority** : Priorité du membre dans l'élection (plus élevée = plus de chance d'être Primaire).
- **votes** : Indique si le membre a une voix lors d'une élection (généralement 1).
- **arbiterOnly** : Si true, le membre participe au vote, mais ne stocke pas de données.
- **hidden** : Si true, le membre est invisible aux applications clientes.



Explique les champs… rs.config()

Cette commande fournit un aperçu de l'état actuel et opérationnel de tous les membres du Replica Set. Les champs clés comprennent :





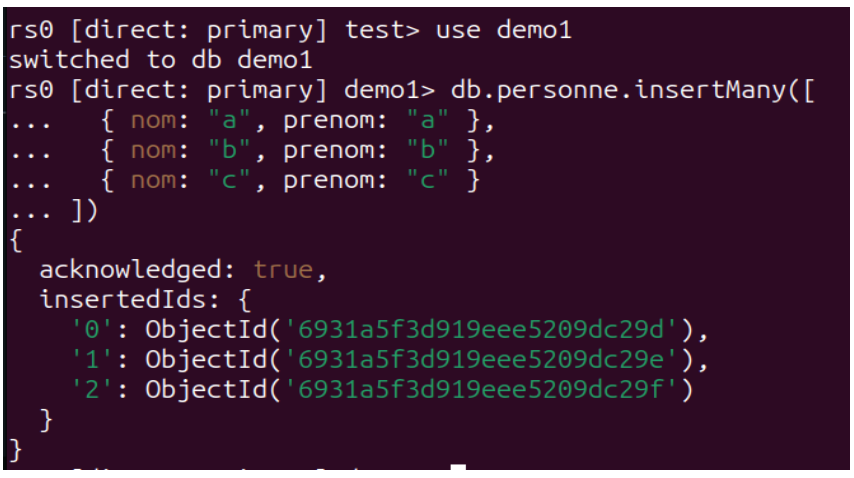
- **set**: Le nom du Replica Set.
- **myState:** L'état actuel du nœud à partir duquel la commande a été exécutée.
- **members:** Un tableau qui détaille l'état de chaque membre du Replica Set.
    - **name: L**'adresse réseau du membre.
    - **stateStr:** L'état du membre (PRIMARY / SECONDARY)
    - **health:** Un indicateur de santé.
    - **uptime:** Le temps écoulé (en secondes) depuis le démarrage du processus.
    - **optimeDate:** Le timestamp de la dernière opération de réplication appliquée par ce membre. Ce champ est crucial pour calculer le lag de réplication
    - **electionTime:**La date de la dernière élection remportée.

7. Démonstration de la réplication

Dans cette section, on illustre le fonctionnement du **Replica Set** en effectuant une écriture sur le nœud primaire **mongo1**, puis en vérifiant que les données sont bien répliquées sur les nœuds secondaires **mongo2** et **mongo3**.

1. **Connexion au nœud primaire**
    - Connexion au conteneur primaire :
      docker exec -it mongo1 mongosh
    - Sélection de la base de données :
      use demo1
2. **Création de la collection et insertion de documents**
    - Création de la collection personne et insertion de trois documents :

```
db.personne.insertMany([
  { nom: "a", prenom: "a" },
  { nom: "b", prenom: "b" },
  { nom: "c", prenom: "c" }
])
```



3. **Vérification sur le nœud secondaire mongo2**
    - Connexion : **docker exec -it mongo2 mongosh**
    - Passage en mode lecture sur un secondaire (si nécessaire): **rs.secondaryOk()**
    - Sélection de la base : **use demo1**
    - Vérification des données : **db.personne.find()**

```
rs0 [direct: secondary] demo1> use demo1
already on db demo1
rs0 [direct: secondary] demo1> db.personne.find()
[
  { _id: ObjectId('6931a5f3d919eee5209dc29d'), nom: 'a', prenom: 'a' },
  { _id: ObjectId('6931a5f3d919eee5209dc29e'), nom: 'b', prenom: 'b' },
  { _id: ObjectId('6931a5f3d919eee5209dc29f'), nom: 'c', prenom: 'c' }
]
rs0 [direct: secondary] demo1>
```

4. **Vérification sur le nœud secondaire mongo3**
    - Connexion : **docker exec -it mongo3 mongosh**
    - Passage en mode lecture (si nécessaire) : **rs.secondaryOk()**
    - Sélection de la base : **use demo1**
    - Vérification des données : **db.personne.find()**

```
rs0 [direct: secondary] test> use demo1
switched to db demo1
rs0 [direct: secondary] demo1> db.personne.find()
[
  { _id: ObjectId('6931a5f3d919eee5209dc29d'), nom: 'a', prenom: 'a' },
  { _id: ObjectId('6931a5f3d919eee5209dc29e'), nom: 'b', prenom: 'b' },
  { _id: ObjectId('6931a5f3d919eee5209dc29f'), nom: 'c', prenom: 'c' }
]
rs0 [direct: secondary] demo1>
```

Explication:

- l'insertion des documents dans la collection **personne** est effectuée uniquement depuis le nœud **primaire (mongo 1)** du replica set.
- Dès que le primaire confirme l'écriture, il enregistre l'opération dans son journal des opérations. Les nœuds secondaires (**mongo 2** et **mongo 3**) lisent en continu ce journal, appliquent les mêmes opérations localement, puis mettent à jour leurs propres données pour rester synchronisés avec le primaire.
- Ainsi, lorsque l'on exécute la commande find() sur les secondaires, on observe les mêmes documents que ceux insérés sur le primaire.