Nom : Loubna MANGOUCHI
N°Étudiant : 124099931
Classe : INFO3

Réplication et Haute Disponibilité avec MongoDB

Haute Disponibilité avec MongoDB

1. Objectifs du TP

L'objectif de ce travail pratique est de mettre en place un **Replica Set** (ensemble de répliques) avec MongoDB afin de :

- Comprendre le mécanisme de réplication des données (Primaire vs Secondaire).
- Tester la cohérence des données entre les nœuds.
- Simuler une panne serveur pour observer l'élection automatique d'un nouveau maître (Failover).

2. Infrastructure

Nous avons démarré trois instances de serveurs `mongod` sur la même machine (localhost) écoutant sur les ports suivants :

- **Port 27018** (Initialement le Primaire)
- **Port 27019** (Secondaire)
- **Port 27020** (Secondaire)
- **Port 27020** (Arbiter)

3. Initialisation du Replica Set

Nous nous sommes connectés au serveur sur le port **27018** pour initier la configuration du cluster. **Commande exécutée :** `rs.initiate()` avec la configuration des membres (`_id: 0` à `2`).

> **Observation :** Le serveur 27018 prend le rôle de **PRIMARY**.

4. Manipulation des données et Réplication

4.1. Écriture sur le nœud Primaire

Nous avons créé une base de données `demo1` et une collection `person`, puis inséré trois documents JSON.

Commandes :

JavaScript
use demo1
db.createCollection("person")
db.person.insert({nom: "Dupont", ...}) // Insertion de plusieurs éléments
db.person.find()

4.2. Lecture sur un nœud Secondaire

Nous nous sommes connectés au serveur **27019** (Secondaire). Par défaut, MongoDB interdit la lecture sur les nœuds secondaires pour éviter de lire des données potentiellement non à jour (cohérence forte).

Pour autoriser la lecture, nous avons dû changer la configuration de la session. **Commande :** `rs.secondaryOk()` (ou `rs.slaveOk()` sur les anciennes versions).

Une fois la lecture activée, nous avons pu vérifier que les données insérées sur le 27018 ont bien été répliquées sur le 27019.

4.3. Tentative d'écriture sur un Secondaire

Nous avons tenté d'insérer une donnée depuis le port **27019**. **Observation :** L'opération est rejetée. Les nœuds secondaires sont en **lecture seule** stricte.

5. Tolérance aux pannes (Failover)

5.1. Simulation de la panne

Nous avons simulé un crash du serveur primaire en arrêtant brutalement le processus du port **27018** (Ctrl+C).

Observation : Les autres nœuds du cluster détectent la panne et initient une élection.

5.2. Résultat de l'élection

En nous connectant au serveur **27019**, nous constatons qu'il a changé de statut. Il est passé de `SECONDARY` à `PRIMARY`. Les données sont toujours accessibles.

6. Ajout d'un Arbitre (Bonus)

Nous avons préparé l'ajout d'un nœud "Arbitre". Ce type de nœud participe aux élections mais ne stocke aucune donnée. **Action :** Création du répertoire `arbitre1` et démarrage d'une instance `mongod` sur un nouveau port (ex: 27028) dédiée à ce rôle.

Conclusion

Ce TP a permis de valider le fonctionnement de la réplication asynchrone de MongoDB. Nous avons confirmé que l'écriture se fait uniquement sur le maître, que la lecture sur les esclaves nécessite une configuration explicite, et que le cluster assure une haute disponibilité en élisant automatiquement un nouveau maître en cas de panne.