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

Mise en place d'une architecture distribuée avec MongoDB

Création et administration d'un Replica Set (Grappe de serveurs).

1. Introduction:

L'objectif de ce laboratoire est de mettre en pratique les concepts de haute disponibilité et de tolérance aux pannes dans MongoDB. Nous avons simulé un environnement distribué sur une seule machine Linux en créant un Replica Set composé de plusieurs nœuds : un Primaire (Primary), des Secondaires (Secondaries) et un Arbitre.

2. Préparation de l'environnement

Pour simuler plusieurs serveurs physiques sur une seule machine, nous avons créé des répertoires de données distincts pour chaque processus `mongod`.

Commandes exécutées :

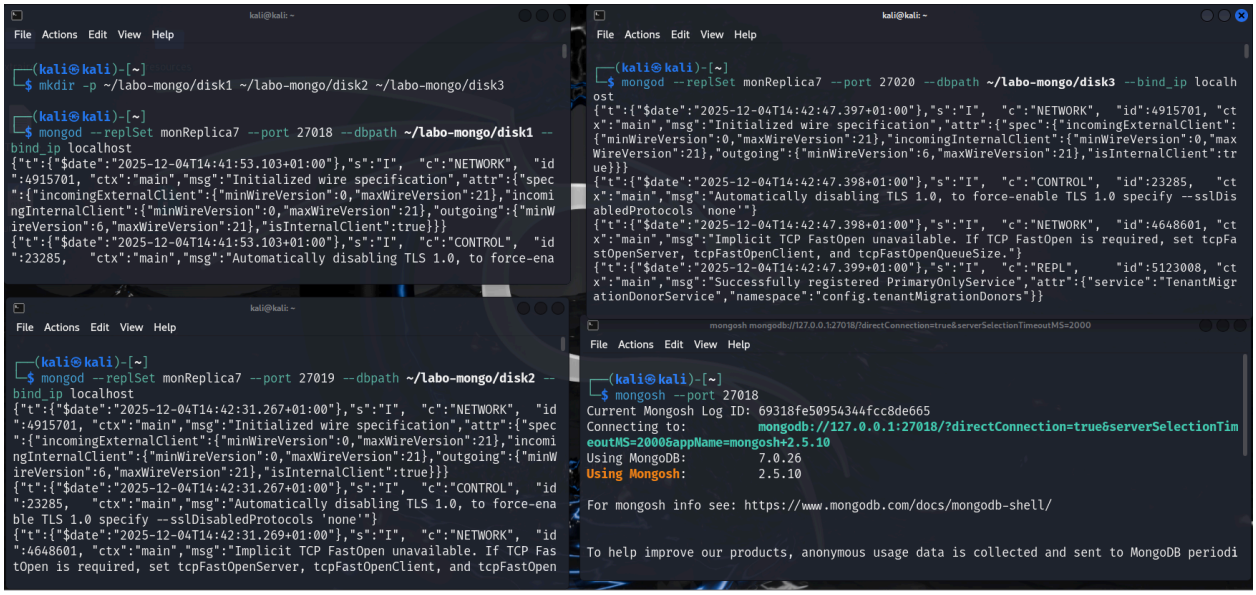
mkdir -p ~/labo-mongo/disk1 ~/labo-mongo/disk2 ~/labo-mongo/disk3 ~/labo-mongo/diskArb

3. Démarrage des nœuds de données

Nous avons lancé trois instances de serveurs MongoDB sur des ports différents (27018, 27019, 27020) appartenant au même ensemble de réplication nommé `monReplica7`.

Commande utilisée pour chaque serveur :

mongod --replSet monReplica7 --port 27018 --dbpath ~/labo-mango/disk1 --bind_ip localhost



4. Initialisation du Replica Set

Depuis le client `mongosh` connecté au port 27018, nous avons initialisé la grappe et ajouté les deux autres membres.

Commandes :

JavaScript
rs.initiate()
rs.add("localhost:27019")
rs.add("localhost:27020")

Après vérification avec `rs.status()`, nous constatons que l'élection a eu lieu : un nœud est devenu PRIMARY et les deux autres SECONDARY.

```
members: [
  {
    _id: 0,
    name: 'localhost:27018',
    health: 1,
    state: 1,
    stateStr: 'PRIMARY',
    uptime: 342,
    optime: { ts: Timestamp({ t: 1764856050, i: 1 }), t: Long('1') },
    optimeDate: ISODate('2025-12-04T13:47:30.000Z'),
    lastAppliedWallTime: ISODate('2025-12-04T13:47:30.228Z'),
    lastDurableWallTime: ISODate('2025-12-04T13:47:30.228Z'),
    syncSourceHost: '',
    syncSourceId: -1,
    infoMessage: '',
    electionTime: Timestamp({ t: 1764855860, i: 2 }),
    electionDate: ISODate('2025-12-04T13:44:20.000Z'),
    configVersion: 5,
    configTerm: 1,
    self: true,
    lastHeartbeatMessage: ''
  },
  {
    _id: 1,
    name: 'localhost:27019',
    health: 1,
    state: 2,
    stateStr: 'SECONDARY',
    uptime: 93,
    optime: { ts: Timestamp({ t: 1764856050, i: 1 }), t: Long('1') },
    optimeDurable: { ts: Timestamp({ t: 1764856050, i: 1 }), t: Long('1') },
```

5. Test de la réplication des données

Nous avons inséré une donnée sur le nœud Primaire pour vérifier sa propagation. Pour lire cette donnée depuis un nœud Secondaire (port 27019), nous avons dû modifier la préférence de lecture, car MongoDB interdit par défaut la lecture sur les esclaves pour garantir la cohérence forte.

Commandes :

JavaScript
// Sur le Primaire
db.test.insertOne({ message: "Hello Replica Set" })

// Sur le Secondaire
db.getMongo().setReadPref('secondary')
db.test.find()

6. Ajout d'un Nœud Arbitre

Pour prévenir les problèmes de partitionnement réseau ("Split-Brain") et assurer un quorum lors des élections, nous avons ajouté un nœud Arbitre. Ce nœud ne stocke pas de données.

Problème rencontré : Configuration du Write Concern

Lors de la tentative d'ajout de l'arbitre avec `rs.addArb()`, une erreur `NewReplicaSetConfigurationIncompatible` est survenue. Explication : Les versions récentes de MongoDB exigent une configuration explicite des règles d'écriture (*Default Write Concern*) avant d'accepter un arbitre.

Solution appliquée

Nous avons forcé la règle par défaut via une commande administrative :

JavaScript

```
db.adminCommand({
  setDefaultRWConcern: 1,
  defaultWriteConcern: { w: 1 }
})
rs.addArb("localhost:27021")
```

7. État final de l'architecture

En conclusion, nous avons vérifié l'état final de la grappe. L'arbitre est bien reconnu avec l'état `ARBITER` (code 7) et ne consomme pas de ressources de stockage pour les données.

8. Conclusion

Ce TP a permis de comprendre le mécanisme de réplication asynchrone de MongoDB. Nous avons appris à gérer manuellement les processus `mongod`, à interagir avec la configuration du Replica Set, et à résoudre des conflits de configuration liés aux versions récentes de la base de données (Write Concern). L'architecture est désormais résiliente à la panne d'un nœud.