TP n°2 : Réplication et tolérance aux pannes avec MongoDB et Cassandra

Partie 1 : la réplication sous MongoDB :

Introduction :

On peut aborder l'organisation d'un système NoSQL en observant comment les données circulent et se répartissent entre plusieurs serveurs. Dans un environnement distribué, la réplication joue un rôle essentiel : elle consiste à créer plusieurs copies d'une même information pour que, même si un nœud devient inaccessible, les autres puissent continuer à répondre. Grâce à cette duplication contrôlée, on peut maintenir un service disponible et réactif, tout en limitant les risques de perte de données.

Lorsqu'on regroupe plusieurs nœuds, on forme ce qu'on appelle une grappe. Cette grappe fonctionne comme un ensemble coordonné, où chaque machine contribue au stockage et au traitement. À mesure que les besoins augmentent, on peut agrandir cette grappe en ajoutant de nouveaux nœuds, sans remettre en question l'architecture globale. Cette capacité à s'étendre naturellement est une des forces des bases NoSQL.

À l'intérieur de cette organisation, certains systèmes utilisent une relation maître-esclave. Le maître reçoit les écritures et sert de point de référence, tandis que les esclaves répliquent ses données et assument souvent une partie des lectures. Grâce à cette séparation des rôles, on peut répartir la charge, éviter les conflits et garantir que l'information circule correctement dans tout le cluster. Au sein d'une grappe, si un nœud maître disparaît, on peut utiliser un nœud esclave comme nouveau nœud maître afin de continuer à assurer l'ensemble du service grâce à une élection.

En combinant ces mécanismes, un système NoSQL peut donc assurer stabilité, évolutivité et rapidité. Chaque élément — réplication, grappe, répartition des rôles — contribue à construire un ensemble cohérent, où les données restent accessibles malgré les aléas du réseau et les montées en charge. Cette organisation distribuée forme la base de nombreux outils modernes, pensés pour manipuler des volumes de données toujours plus importants.

Mise en place des noeuds de réplication :

Dans un premier temps, nous allons ouvrir 4 terminaux : 3 qui vont servir de serveur, que l'on va nommer Serv1, Serv2 et Serv3 ici tandis que le dernier servira de client.
Afin de lancer des noeuds différents sur chaque serveur, on va créer à dans notre container les dossier disque1, disque2 et disque3 qui permettent respectivement de stocker des données de Serv1, Serv2 et Serv3 :

```
Serv1#mkdir disque1
Serv2#mkdir disque2
Serv3#mkdir disque3
```

On va ensuite ouvrir chaque serveur sur des ports différents en précisant le set de réplication, le port utilisé et la chemin de stockage :

```
Serv1#mongod --replSet monreplicaset --port 27018 --dbpath disque1
Serv2#mongod --replSet monreplicaset --port 27019 --dbpath disque2
Serv3#mongod --replSet monreplicaset --port 27020 --dbpath disque3
```

On va ensuite se connecter au premier noeud :

```
Client#mongosh –port 27018
```

A présent, nous sommes bien connectés avec le client au nœud de Serv1. Nous allons initialiser la grappe de la manière suivante :

```
Client#rs.initiate()
```

De cette manière, le nœud Serv1 est le nœud maître de la grappe. Nous allons à présent ajouter les deux autres nœuds à la grappe. Il faut bien s'assurer d'avoir le nom de votre machine ou son adresse IP afin de pouvoir ajouter correctement les autres nœuds. Il est possible de voir ce dernier dans le champs "me" de la commande précédente :

```
{
  info2: 'no configuration specified. Using a default configuration for the set',
  me: 'localhost:27018',
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1764855957, i: 1 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1764855957, i: 1 })
}
```

Ajout des deux autres noeuds à la grappe :

```
Client#rs.add("localhost:27019")
Client#rs.add("localhost:27020")
```

Lecture et écriture sur noeuds esclaves :

MongoDB est un système en écriture asynchrone. Cela signifie que lors d'une écriture, le noeud maître va prendre l'écriture en compte, l'indiquer dans les logs puis va la communiquer aux noeuds esclaves. De cette manière la cohérence à un instant T n'est pas assurée mais elle sera assurée dans un futur proche à l'aide des logs et des communications entre les nœuds. C'est pourquoi la lecture et l'écriture sur le nœud maître sont privilégiées.

Paramètres de base : écriture sur le port 27018, qui est donc le nœud primary. Il s'agit ici du du maître. Les ports 27019 et 27020 sont quant-à eux les nœuds esclaves. On ne peut ni lire ni écrire sur ces derniers. On peut cependant forcer la lecture sur les esclaves mais pas l'écriture.

Afin de tester ces propriétés, nous allons créer une base, puis une collection que l'on va remplir depuis le primary. On va ensuite insérer des données que l'on va essayer de lire depuis le secondary, ce qui devrait échouer. On va forcer la lecture pour pouvoir lire sur ce dernier. Enfin, on va essayer d'écrire depuis le secondary, ce qui devrait échouer.

Création de la base :
```
Client#use demo1
```

Création de la collection :
```
Client# demo1> db.createCollection("personnes")
```

Insertion d'un élément :
```
Client# demo1>db.personnes.insertOne({"nom":"youcef"})
```

Lecture des éléments :
```
Client# demo1> db.personnes.find()

[
  { _id: ObjectId('6936a5324fefeb923e9dc29d'), nom: 'youcef' },
  { _id: ObjectId('6936a53a4fefeb923e9dc29e'), nom: 'youcef' },
  { _id: ObjectId('6936a5434fefeb923e9dc29f'), nom: 'Arthur' },
  { _id: ObjectId('6936a54a4fefeb923e9dc2a0'), nom: 'Cath' }
]
```

Déconnexion :
```
Client# demo1> exit
```

Reconnection au secondary :
```
Client#mongosh –port 27019
Client#use demo1
```

On va tenter de lire :
```
Client# demo1> show collections
```

Normalement, à ce niveau, on devrait avoir une erreur qui nous indique que l'on ne peut pas lire car nous sommes sur un nœud esclave. Dans ma propre manipulation, j'ai pu lire tout de même, sûrement à cause de paramètres par défauts qui autorisent les lectures ou parce que le compte que j'utilise a des droits particuliers, bien que je ne les ai pas donnés.

Afin de forcer la lecture sur un noeud esclave, on utilise la commande suivante :
```
Client# demo1> rs.secondaryOk()
```

On peut ensuite lire convenablement les données :

```
Client# demo1> show collections
personnes

Client# demo1> db.personnes.find()
[
  { _id: ObjectId('6936a5324fefeb923e9dc29d'), nom: 'youcef' },
  { _id: ObjectId('6936a53a4fefeb923e9dc29e'), nom: 'youcef' },
  { _id: ObjectId('6936a5434fefeb923e9dc29f'), nom: 'Arthur' },
  { _id: ObjectId('6936a54a4fefeb923e9dc2a0'), nom: 'Cath' }
]
```

Par contre l'écriture reste interdite :

```
Client#demo1> db.personnes.insertOne({"nom":"FZ"})
MongoServerError[NotWritablePrimary]: not primary
```

Reprise sur panne :

Nous allons à présent nous intéresser à la reprise sur panne dans ce genre de système.

La théorie veut que lorsque le noeud maître est en panne, un élection d'un nouveau maître parmi les esclaves se fait. Cela est possible car des messages heartbeat sont envoyés fréquemment afin de s'assurer que tous les nœuds sont actifs.

Nous allons tester cette reprise en désactivant le serveur maître :

```
Serv1#exit
```

On vérifie que le serveur est bien déconnecté en essayant de s'y connecter :

```
Client#mongosh –port 27018
Current Mongosh Log ID: 6936d6472efc5dc28e9dc29c
Connecting to:
mongodb://127.0.0.1:27018/?directConnection=true&serverSelectionTimeoutMS=2000&app
Name=mongosh+2.5.9
MongoNetworkError: connect ECONNREFUSED 127.0.0.1:27018
```

A présent, en se connectant au serveur 3, on voit que ce dernier est le nouveau primary :

```
Client#mongosh –port 27020
monreplicaset [direct: primary] test>
```

On arrive bien à lire les données :

```
Client# use demo1
switched to db demo1
Client# demo1> show collections
personnes
Client# demo1> db.personnes.find()
[
  { _id: ObjectId('6936a5324fefeb923e9dc29d'), nom: 'youcef' },
  { _id: ObjectId('6936a53a4fefeb923e9dc29e'), nom: 'youcef' },
```

```
  { _id: ObjectId('6936a5434fefeb923e9dc29f'), nom: 'Arthur' },
  { _id: ObjectId('6936a54a4fefeb923e9dc2a0'), nom: 'Cath' }
]
```

Mais on peut également insérer des données :

```
Client# demo1> db.personnes.insertOne({"nom":"FZ"})
{
  acknowledged: true,
  insertedId: ObjectId('6936d813dafebe56259dc29d')
}
```

Enfin, on peut définir un processus qui servira d'arbitre. Ce dernier ne stocke pas des données de la base mais il participe aux élections en cas de panne du primary.

On va dans un premier temps créer le dossier qui lui sera associé :

```
Arbitre#mkdir arbitre1
```

Par la suite, on lance le processus arbitre :

```
Arbitre# mongod --replSet simulation44 --port 27028 --dbpath arbitre1
```

Partie 2 : Question sur la réplication et la tolérance aux pannes :

I - Compréhension de base :

1. Qu'est-ce qu'un Replica Set dans MongoDB ?

Le réplica set est une grappe de serveur permettant de mettre en place une architecture distribuée dans MongoDB

2. Quel est le rôle du Primary dans un Replica Set ?

Le primary d'un replica set est le maître de la grappe. Ce dernier permet de lire et d'écrire sur la base. Il s'agit également du nœud qui envoie des informations de mise à jour aux esclaves.

3. Quel est le rôle essentiel des Secondaries ?

Les secondaries sont les noeuds esclaves, ils sont donc les noeuds qui vont répliquer le primary et qui prendront la suite en cas de panne sur ce dernier grâce à une élection.

4. Pourquoi MongoDB n'autorise-t-il pas les écritures sur un Secondary ?

Cela permet d'éviter les incohérence à terme car MongoDB assure la cohérence à terme. De plus, le système étant asynchrone, si nous pouvons écrire sur un secondary, cela signifie que l'on pourra aux mêmes endroits en même temps ce qui pose un problème de concurrence.

5. Qu'est-ce que la cohérence forte dans le contexte MongoDB ?

La cohérence forte concerne la possibilité d'écrire et de lire uniquement sur le primary

6. Quelle est la différence entre readPreference : "primary" et "secondary" ?
En mode Primary, on ne lit que sur le Primary alors qu'en mode Secondary, on va lire uniquement sur les secondaries.

7. Dans quel cas pourrait-on souhaiter lire sur un Secondary malgré les risques ?
Quand on veut pouvoir accéder de manière très rapide aux données (par exemple lorsque l'on fait de l'analytique sur les données) sans s'inquiéter d'avoir ou non les données les plus récentes.

II - Commandes & configuration :

8. Quelle commande permet d'initialiser un Replica Set ?
rs.initiate()

9. Comment ajouter un nœud à un Replica Set après son initialisation ?
rs.add("noeud") avec noeud = adresse du noeud qu'on ajoute. On fait cette commande depuis le noeud maitre qui a été initialisé.

10. Quelle commande permet d'afficher l'état actuel du Replica Set ?
rs.status()

11. Comment identifier le rôle actuel (Primary / Secondary / Arbitre) d'un nœud ?
On peut le voir avec la commande rs.status(), un champ parmi ceux des noeuds permet de voir son rôle.

12. Quelle commande permet de forcer le basculement du Primary ?
rs.stepDown() permet de relancer une nouvelle élection. Ainsi, un secondary deviendra le primary.

13. Comment peut-on désigner un nœud comme Arbitre ? Pourquoi le faire ?
rs.addArb("noeud")
Cela permet de désigner un nœud qui ne stocke pas de données mais qui vote en cas d'élection. Cette manœuvre permet, en cas de grappe avec un nombre impair de nœud, de désigner un groupe majoritaire même si les deux ne peuvent communiquer. Ainsi, on ne crée pas de nœud de données inutile, il le crée simplement afin de voter et décider de quel groupe est majoritaire.

14. Donnez la commande pour configurer un nœud secondaire avec un délai de réplication

(slaveDelay)

On va modifier la valeur cfg.member[id].slaveDelay avec la commande rs.config().

III - Résilience et tolérance aux pannes :

15. Que se passe-t-il si le Primary tombe en panne et qu'il n'y a pas de majorité ?
Les écritures deviennent impossible sur la grappe concernée.

16. Comment MongoDB choisit-il un nouveau Primary ? Quels critères utilise-t-il ?
Il le choisit via une élection. Cette élection se base sur la priorité affectée à chaque noeud et à quel point les logs sont récents.

17. Qu'est-ce qu'une élection dans MongoDB ?
Il s'agit du processus permettant de définir le nouveau primary

18. Que signifie auto-dégradation du Replica Set ? Dans quel cas cela survient-il ?
C'est un désistement du Primary. Il se rétrograde en secondary automatiquement. Cela arrive quand il ne voit plus les membres du replica set (perte de la majorité), les oplog en retard, latence réseau importante. Cela permet d'éviter des écritures non sécurisées.

19. Pourquoi est-il conseillé d'avoir un nombre impair de nœuds dans un Replica Set ?
Afin de toujours avoir un vote majoritaire.

20. Quelles conséquences a une partition réseau sur le fonctionnement du cluster ?
Une latence accrue, une déconnexion au sein de la grappe qui va diviser la grappe en deux grappes.

IV - Scénarios pratiques :

21. Vous avez 3 nœuds : 27017 (Primary) , 27018 (Secondary) , et 27019 (Arbitre) . Que se passe-t-il si le Primary devient injoignable ?
Si le primary devient injoignable, il y a une élection parmi le reste de la grappe. Étant donné qu'il reste un arbitre et un secondary, ce sera le nœud secondary qui deviendra le nouveau primary.

22. Vous avez configuré un Secondary avec un slaveDelay de 120 secondes. Quelle est son utilité ? Quels usages peut-on en faire dans la vraie vie ?
Cela signifie que la réplication va s'effectuer avec un délai de 120 secondes sur ce nœud. Cela peut permettre de revenir à un version antérieure de la base en cas d'erreur importante (suppression par erreurs, mauvaise insertion)

23. Un client exige une lecture toujours à jour, même en cas de bascule. Quelles options de readConcern et writeConcern recommanderiez-vous ?
Il faut utiliser l'option majority sur w en writeConcern et level pour readConcer. Cela permet de s'assurer que ce qu'on lit se trouve sur la majorité des noeuds et donc de voir des données toujours fiable, même en cas de bascule.

24. Dans une application critique, vous voulez garantir que l'écriture est confirmée par au moins deux nœuds. Quelle option de writeConcern devez-vous utiliser ?
On va mettre w:2 afin d'avoir deux nœuds qui confirment que l'écriture a été effectuée. On a ici le primary et un secondary.

25. Un étudiant a lu depuis un Secondary et récupéré une donnée obsolète. Expliquez pourquoi et comment éviter cela.
Comme MongoDb est en écriture asynchrone, on peut avoir une écriture qui n'est pas encore propagée lors de la lecture. Pour éviter cela, on va vouloir appliquer readPrefer en mode primary.

26. Montrez la commande pour vérifier quel nœud est actuellement Primary dans votre ReplicaSet.
Il faut regarder le champ "stateStr" des noeuds avec la commande "rs.status()"

27. Expliquez comment forcer une bascule manuelle du Primary sans interruption majeure.
On va exécuter la commande rs.stepDown() sur le noeud primary.

28. Décrivez la procédure pour ajouter un nouveau nœud secondaire dans un Replica Set en fonctionnement.
On va effectuer la commande rs.add("noeud")

29. Quelle commande permet de retirer un nœud défectueux d'un Replica Set ?
On va effectuer la commande rs.remove("noeud")

30. Comment configurer un nœud secondaire pour qu'il soit caché (non visible aux clients) ? Pourquoi ferait-on cela ?
Pour configurer un noeud en caché :

```
cfg = rs.conf()
cfg.members[id].hidden = true
rs.reconfig(cfg)
```

On peut faire cela pour avoir un nœud qui servira uniquement de backup par exemple. Il n'est donc pas voué à être lu ou utilisé.

31. Montrez comment modifier la priorité d'un nœud afin qu'il devienne le Primary préféré.
cfg = rs.conf()
cfg.members[n].priority = <high value>
rs.reconfig(cfg)

32. Expliquez comment vérifier le délai de réplication d'un Secondary par rapport au Primary.
rs.printSlaveReplicationInfo()
On voit directement le lag.

33. Que fait la commande rs.freeze() et dans quel scénario est-elle utile ?
Empêche un nœud de participer aux élections. C'est utile pour une maintenance par exemple, on va rendre le primary indisponible de manière volontaire mais on ne veut pas qu'un autre nœud soit élu. On peut également contrôler les élections de cette manière.

34. Comment redémarrer un Replica Set sans perdre la configuration ?
Il suffit de relancer le réplica set avec le même nom de set.

35. Expliquez comment surveiller en temps réel la réplication via les logs MongoDB ou commandes shell.

Dans les logs du Secondary :

tail -f /var/log/mongodb/mongod.log

On y voit :

- réceptions d'oplog
- connexions au Primary
- retard éventuel

Exemples de messages :
oplog fetcher
sync source
replication lag

En commandes on peut peut regarder rs.status et s'intéresser à : state, optime, heartbeat, health, sinc source.