Mise en place d'une architecture tolérante aux pannes avec MongoDB

Pour mettre en place une architecture avec MongoDB tolérante aux pannes, on procède de la façon suivante :

Définir un répertoire de sauvegarde pour chacun des serveurs. Ici nous choisissons de créer 3 serveurs dont les répertoires sont nommés rs1, rs2 et rs3 avec différents ports d'écoute (27018, 27019,27020).

On utilise la commande « mkdir » pour créer un répertoire.

On associe les répertoires de stockage à chacun de nos réplicaset que l'on met en écoute sur les ports. On effectue cela grâce à la commande :

```
C:\Program Files\MongoDB\Server\4.0\bin>mongod --replSet rs0 --port 27018 --dbpath rs1
```

```
C:\Program Files\MongoDB\Server\4.0\bin>mongod --replSet rs0 --port 27019 --dbpath rs2
```

```
C:\Program Files\MongoDB\Server\4.0\bin>mongod --replSet rs0 --port 27020 --dbpath rs3
```

Le replicaSet ne fonctionne cependant pas on va donc le mettre en place. Pour cela je me connecte à mon port 27018 qui correspond à notre serveur principal :

```
C:\Program Files\MongoDB\Server\4.0\bin>mongo --port 27018
```

Pour initialiser notre replicaSet on se connecte sur le serveur principal et on applique la commande suivante :

```
rs.initiate()
```

Cela nous permet de voir la machine présente dans le replicaset. Ici on s'aperçoit que c'est bien la machine rs1 sur le port 27018.

```
> rs.initiate()
{
        "info2" : "no configuration specified. Using a default configuration for the set",
        "me" : "localhost:27018",
        "ok" : 1,
        "operationTime" : Timestamp(1607528625, 1),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1607528625, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
```

Si nous souhaitons vérifier la configuration de notre réplicaSet nous utilisons la commande « rs.conf() » cela nous permet de renvoyer les membres qui appartiennent au réplicaSet.

```
rs0:SECONDARY> rs.conf()
```

On peut maintenant ajouter nos différents réplicaSet en les nommant par le numéro de port précédemment établi en procédant comme ci-dessous :

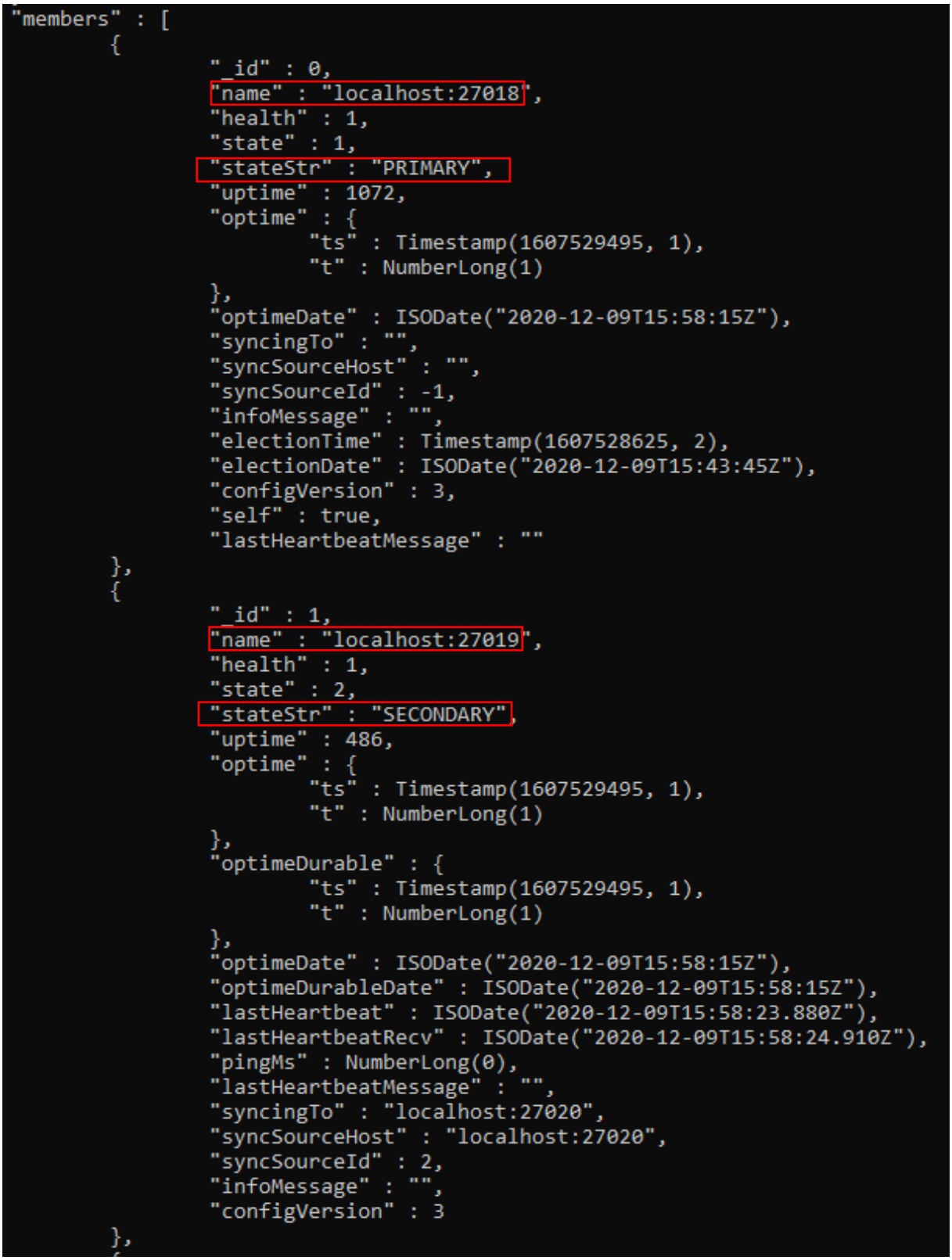
```
rs0:PRIMARY> rs.add("localhost:27019")
{
        "ok" : 1,
        "operationTime" : Timestamp(1607529018, 1),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1607529018, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
```

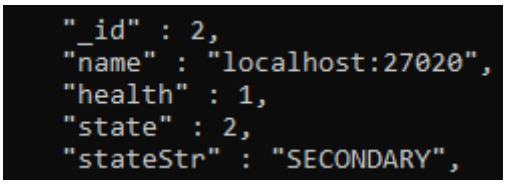
```
rs0:PRIMARY> rs.add("localhost:27020")
{
        "ok" : 1,
        "operationTime" : Timestamp(1607529023, 1),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1607529023, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
```

On peut maintenant vérifier la configuration de notre réplicaSet avec « rs.conf() ». On observe bien que notre réplicaSet contient bien les 3 membres définis précédemment.

```
rs0:PRIMARY> rs.conf()
{
        "_id" : "rs0",
        "version" : 3,
        "protocolVersion" : NumberLong(1),
        "writeConcernMajorityJournalDefault" : true,
        "members" : [
                {
                        "_id" : 0,
                        "host" : "localhost:27018",
                        "arbiterOnly" : false,
                        "buildIndexes" : true,
                        "hidden" : false,
                        "priority" : 1,
                        "tags" : {

                        },
                        "slaveDelay" : NumberLong(0),
                        "votes" : 1
                },
                {
                        "_id" : 1,
                        "host" : "localhost:27019",
                        "arbiterOnly" : false,
                        "buildIndexes" : true,
                        "hidden" : false,
                        "priority" : 1,
                        "tags" : {

                        },
                        "slaveDelay" : NumberLong(0),
                        "votes" : 1
                },
                {
                        "_id" : 2,
                        "host" : "localhost:27020",
                        "arbiterOnly" : false,
                        "buildIndexes" : true,
                        "hidden" : false,
                        "priority" : 1,
                        "tags" : {

                        },
                        "slaveDelay" : NumberLong(0),
                        "votes" : 1
                }
        ],
        "settings" : {
                "chainingAllowed" : true,
                "heartbeatIntervalMillis" : 2000,
                "heartbeatTimeoutSecs" : 10,
                "electionTimeoutMillis" : 10000,
                "catchUpTimeoutMillis" : -1,
                "catchUpTakeoverDelayMillis" : 30000,
                "getLastErrorModes" : {

                },
                "getLastErrorDefaults" : {
                        "w" : 1,
                        "wtimeout" : 0
                },
                "replicaSetId" : ObjectId("5fd0f0b1eb5f1bdcb321b14e")
```

La commande « rs.status() » nous permet de voir les membres de notre réplicaset qui sont défini comme serveur primaire et secondaire :





On remarque que notre serveur rs1 est le serveur primaire et rs2 et rs3 sont les serveurs secondaires.

Pour confirmer que notre réplicaset est bien configuré on peut quitter le serveur rs1 en enlevant l'écoute sur le port 27018. On se connecte à présent sur le port 27019 donc sur rs2 et on applique la commande « rs.status() » pour savoir quel serveur est passé en primaire.

On remarque que le serveur rs2 est passé en serveur primaire et rs3 en secondaire lorsqu'on déconnecte le serveur rs1.

Maintenant si on relance notre serveur rs1 et que l'on affiche le status sur le serveur rs2 on s'aperçoit que rs2 est resté en primary et rs1 et rs3 en secondary. En effet une fois qu'un serveur a été élu primaire il reste primaire jusqu'à ce qu'il soit déconnecté.

Il est possible qu'il y ait un gros temps de latence lors de l'élection du serveur primaire nous allons donc pour cela créer un arbitre pour que les serveurs arrivent à se mettre d'accord.

Pour cela on va créer un répertoire « arb » qui va jouer le rôle de l'arbitre

```
C:\Program Files\MongoDB\Server\4.0\bin>mkdir arb
```

On peut maintenant lancer notre serveur et notre réplicaSet à ce niveau.

```
C:\Program Files\MongoDB\Server\4.0\bin>mongod --port 30000 --dbpath arb --replSet rs0
```

On se connecte à présent sur notre serveur primaire qui est actuellement rs2 sur le port 27019 pour y ajouter notre arbitre.

```
rs0:PRIMARY> rs.addArb("localhost:30000")
{
        "ok" : 1,
        "operationTime" : Timestamp(1607531379, 1),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1607531379, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
```

On vérifie la présence de l'arbitre sur notre serveur :

```
 "_id" : 3,
 "name" : "localhost:30000",
 "health" : 1,
 "state" : 7,
 "stateStr" : "ARBITER",
 "uptime" : 157,
 "lastHeartbeat" : ISODate("2020-12-09T16:32:15.149Z"),
 "lastHeartbeatRecv" : ISODate("2020-12-09T16:32:15.281Z"),
 "pingMs" : NumberLong(0),
 "lastHeartbeatMessage" : "",
 "syncingTo" : "",
 "syncSourceHost" : "",
 "syncSourceId" : -1,
 "infoMessage" : "",
 "configVersion" : 4
```

On s'aperçoit que celui-ci a bien été configuré.

Lorsque qu'un problème survient sur notre réseau et qu'un serveur tombe nous sommes maintenant capables de déterminer rapidement quel serveur va reprendre la main et passer en primary.

Cette configuration va nous permettre d'assurer une qualité de service et de garantir que nos données soit disponible en permanence.

Questions subsidiaires

1. Comment accepter de lire des informations sur un esclave ?

La cohérence à terme est obtenue en autorisant les clients (autrement dit, très concrètement, le driver MongoDB intégré à une application) à effectuer des lectures sur les esclaves.

2. Que signifie le C du théorème de CAP ?

Cela signifie que tous les nœuds du système voient exactement les mêmes données au même moment. Un client récupère donc systématiquement la dernière version du document.

3. Décrire les étapes à suivre pour le sharding (partitionnement)

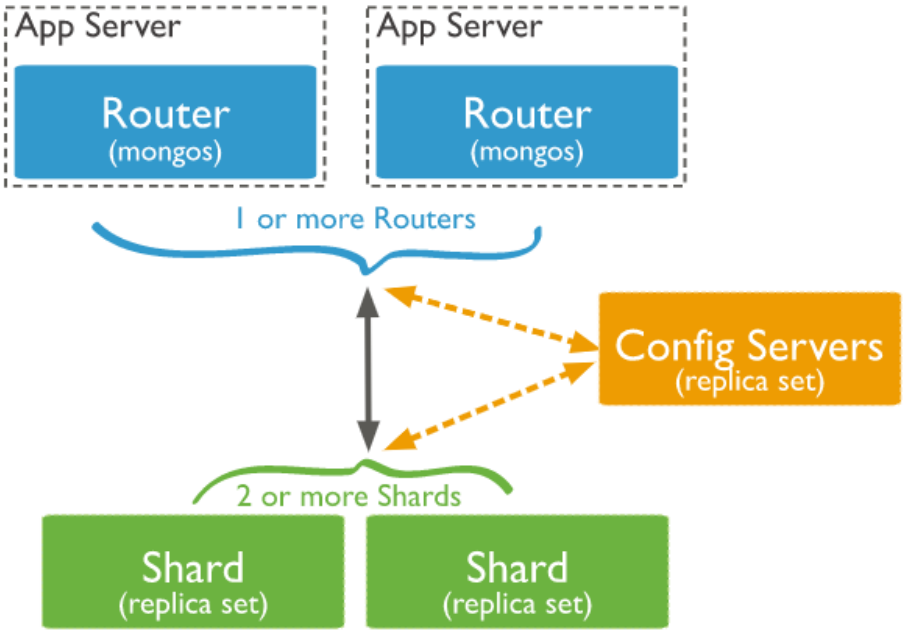


Figure 1 Architecture partitionné

Etape 1 : Configurer les serveurs de configuration et activez la réplication entre eux

Etape 2 : Initialiser le jeu de réplicas sur un des serveurs de configuration

Etape 3 : Configurer les serveurs de partitionnement et activer la réplication entre eux.

Etape 4 : Initialiser le jeu de réplicas sur un des serveurs partitionnés

Etape 5 : Démarrer les mangues pour le cluster fragmenté

Etape 6 : Brancher le serveur de route mongo

Etape 7 : Activer le partitionnement sur la BD et les collections

Etape 8 : Partitionner avec la clé de partition de collection

Etape 9 : Insérer des données pour faire des tests

Etape 10 : Vérifier l'état du partitionnement