Big Data

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

Bourgeois Thomas

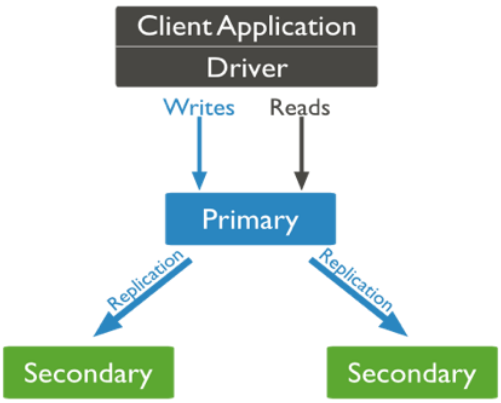
5A 2I

2020-2021

09/12/20

Sommaire

I - Architecture du ReplicaSet



II - Instanciation d'un ReplicaSet

On définit un répertoire de sauvegarde pour chacun des serveurs. Ici, nous partons sur trois serveurs dont les répertoires sont nommés rs1, rs2 et rs3, avec différents ports d'écoute (27018, 27019, 27020). Une fois les répertoires définis, on lance trois serveurs comme suit :

Serveur 1:

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

Serveur 2:

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

Serveur 3:

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

Cependant le ReplicaSet n'existe pas encore.

On lance ensuite un client : **mongo.exe –port 27018** pour se connecter au serveur communicant sur le port 27018, notre serveur principal :

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

Une fois connecté au serveur principal, nous allons initier notre ReplicaSet à l'aide de la commande **rs.initiate()** :

```
> rs.initiate()
{
        "info2" : "no configuration specified. Using a default configuration for the set",
        "me" : "localhost:27018",
        "ok" : 1,
        "operationTime" : Timestamp(1607532983, 1),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1607532983, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
```

On peut voir le nom de notre réseau avec le champ « me »

On va maintenant ajouter les différents serveurs à notre ReplicaSet avec cette commande : **rs.add()**

Ajout du serveur écoutant sur le port 27019 :

```
rstest:SECONDARY> rs.add("localhost:27019")
{
        "ok" : 1,
        "operationTime" : Timestamp(1607533604, 1),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1607533604, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
rstest:PRIMARY>
```

Ajout du serveur écoutant sur le port 27020 :

```
rstest:PRIMARY> rs.add("localhost:27020")
{
        "ok" : 1,
        "operationTime" : Timestamp(1607533681, 1),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1607533681, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
rstest:PRIMARY>
```

On peut alors constater que notre serveur devient primaire, en l'absence d'arbitre. Afin de vérifier que notre ReplicaSet est bien paramétré, on peut regarder sa configuration, via la commande **rs.conf()** :

```
rstest:PRIMARY> rs.conf()
{
        "_id" : "rstest",
        "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),
```

On peut donc observer ici la présence de 3 membres : 27018, 27019,27020.

La commande **rs.status()** nous permet de vérifier quel est le serveur primaire, ici il s'agit du 27018 :

```
"_id" : 0,
"name" : "localhost:27018",
"health" : 1,
"state" : 1,
"stateStr" : "PRIMARY",
```

```
"_id" : 1,
"name" : "localhost:27019",
"health" : 1,
"state" : 2,
"stateStr" : "SECONDARY",
```

```
"_id" : 2,
"name" : "localhost:27020",
"health" : 1,
"state" : 2,
"stateStr" : "SECONDARY",
```

En cas de coupure du serveur primaire, l'un des deux autre prendra automatiquement le relais. Si le serveur initialement primaire remonte, il prendra un rôle secondaire. Il n'y a réélection qu'en cas de coupure du serveur primaire. Ici nous n'avons que 3 serveurs, ainsi la réélection est relativement rapide. Cependant dans des architectures composées d'un grand nombre de serveurs, on peut observer de grosses latences avant d'obtenir un nouveau master. Pour accélérer ce processus, un arbitre peut être mis en place.

III - Mise en place d'un arbitre

Pour commencer, il faut créer un répertoire, que l'on va nommé arb, puis lancer notre serveur depuis ce répertoire sur le port 30000 :

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

L'arbitre n'est cependant pas encore ajouté. Pour ce faire, il faut se connecter avec le client sur le serveur primaire et utiliser la commande **rs.addArb()** :

```
rstest:PRIMARY> rs.addArb("localhost:30000")
{
        "ok" : 1,
        "operationTime" : Timestamp(1607535771, 1),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1607535771, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
```

Désormais, en cas de perte d'un serveur nécessitant une reconfiguration de notre réseau, l'arbitre sera en mesure de définir rapidement qui sera le nouveau Primary.

Toutes ces manipulations nous ont permis de créer de la redondance afin de rendre disponible en permanence nos données.

Chaque serveur possède un fichier de configuration qu'il est possible de modifier de manière pouvoir lire des informations sur un Secondary. Pour pouvoir modifier ce fichier, il faut utiliser cette commande : **mongod.exe --config mongod.conf**

Le C du théorème de CAP signifie Cohérence. Autrement dit, tous les nœuds du système voient exactement les mêmes données au même moment.