MONGODB (suite)

Daniel BADIATA

Consultant Data

Daniel.badiata-kanza@u-pecfr

Objectifs

1/

Big Data, NoSQL et leur rôle dans les applications d'entreprise modernes

2/ Prise en main de MongoDB « administration & Développement »

3/ Réplication et Sharding & performances et indexation de MongoDB

4/ Spring Data et MongoDB

nom	**genre**	**ville**	**regime** (array)	**parlant** (bool)	**poids**
Leto	m	Lyon	carotte, courgette	oui	270
Kenny	f	Roscoff	raisin, carotte, tomate	non	430

Ecrire les requêtes permettant d'insérer ces documents dans une collection

Importer les données contenues dans le fichier (remplacer C:\Etudiants_examens.json par le chemin correspondant à l'emplacement du fichier sur votre ordinateur)

```
> mongoimport --db exemple2 --collection examens  --file
C:\Etudiants_examens.json
```

Utiliser les fonctions find(), update() et insert()

Quelques opérateurs

$in

Syntaxe générale

```
>db.collection.find({champ: {$in:[valeur1, valeur2]})
```

→ Recherche dans la collection, les documents qui satisfont aux conditions *expression1*, *expression2* et *expression3* à la fois.

Exemple :

Liste des étudiants habitant Paris et Bordeaux.

```
→db.etudiants.find({ville:{$in:['Paris','Bordeaux']})
```

Quelques opérateurs

$slice

Cet opérateur détermine le nombre d'éléments renvoyés dans un tableau

Syntaxe générale

>db.collection.find({ champ: value }, { tableau: {$slice: nombre} })

Exemple :

{_id: 6, nom: 'BUL', prénom:'Georges', notes:[13,15,20,9,10, 3,12]}

>db.etudiants.find({ nom:"BUL"}{ notes: {$slice: 3} }) → Affiche les 3 premières notes de l'étudiant BUL

$elemMatch

Appliqué sur un champ de type tableau, cet opérateur permet d'identifier les éléments du tableau qui satisfont simultanément plusieurs critères.

$slice

Permet de limiter le nombre d'éléments renvoyés par un tableau

Quelques opérateurs

Expressions régulières

mongoDB utilisent les expressions régulières compatibles PERL.

Il existe deux manières de procéder :

- **Soit en utilisant l'opérateur $regex**

{ <champ>: { $regex: /pattern/, $options: '<options>' } }

Ou { <champ>: { $regex: 'pattern', $options: '<options>' } }

Ou { <champ>: { $regex: /pattern/<options> } }

- **Soit en utilisant des patterns**

{ <field>: /pattern/<options> }

Quelques opérateurs

Exemples :

- db.etudiants.find({ nom: { $regex: /^b/ } }) → Etudiants dont le nom commence par un « b »
- db.etudiants.find({ nom: { $regex: /al$/ } }) → Etudiants dont le nom se termine par « al »
- db.etudiants.find({ nom: { $regex: /al$/ } }) → Etudiants dont le nom se termine par « al »

Quelques options

i: Pour ignorer la case

x: Pour ignorer les espaces

Projections

Il est possible de limiter les champs affichés lors d'une requête avec find().

Par défaut, tous les champs sont affichés.

Pour masquer certains champs

Syntaxe générale

>db.collection.find({}, {champ:1, champ2:0}), 0 pour masquer et 1 pour afficher.

Pour afficher des champs spécifiques d'un champ qui lui-même est un document de plusieurs champ, il faut utiliser « . ».

Exemple

{_id: 6, nom: 'BUL', prénom:'Georges', notes:[maths:13, physique: 12]}

Si l'on veut afficher uniquement les nom, prénom et la note de physique, on aura

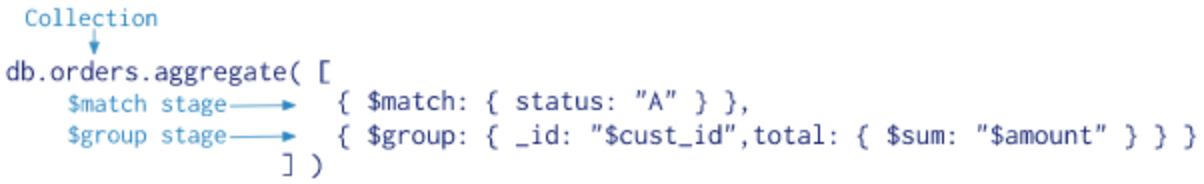
db.etudiants.find({},{_id:0, 'notes.maths':0})

Agrégats

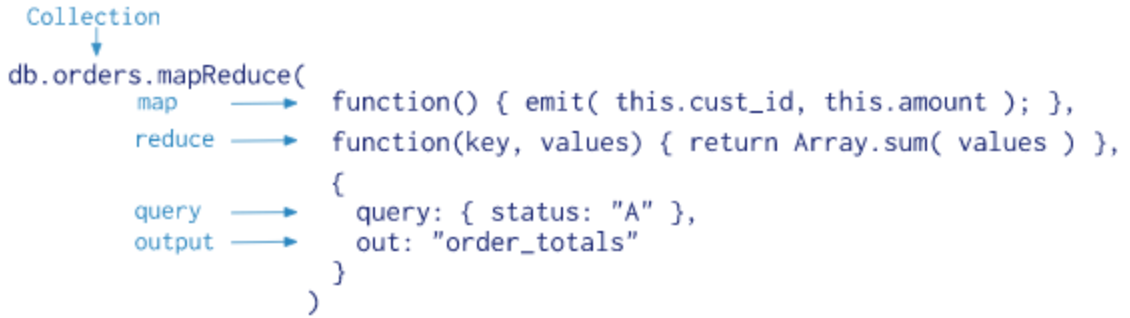
La fonction d'agrégatio n permet de faire des opérations similaires à celles réalisées avec l'opérateur group by de sql.

3 manières de faire les agrégations dans mongoDB: (source: site mongoDB)

- Pipeline



- Map-Reduce



MongoDb fournit également les fonctions distinct() et count() qui correspondent respectivement au select distinct et au count de sql.

Exemples:

- db.etudiants.distinct('nom')-→ Liste des étudiants
- db.etudiants.count()-→ Nombre d'étudiants

Import/Export

On distingue import/export d'une collection et import/export d'une base

COLLECTION

- **mongoexport**: pour faire un exeport des données d'une collection vers un fichier. Par défaut, c'est au format json.
  mongoimport: pour faire un import des données d'un fichier (csv, json, …) vers une collection.

Commandes relatives à une base

- **mongodump**: Export de toute une base vers un répertoire
- **mongorestore**: Pour faire un import du résultat de toute une base (résultat de la commande mongodump).

Import/Export

On distingue import/export d'une collection et import/export d'une base

COLLECTION

- **mongoexport**: pour faire un export des données d'une collection vers un fichier. Par défaut, c'est au format json.

Exemple1 : Fichier json

mongoexport -o c:\cours\etudiants.json -d esipe -c etudiants

→ Export de la collection etudiants de la base esipe dans un fichier csv nommé etudiants créé dans le répertoire c:\cours.

Exemple2 : Fichier csv

mongoexport --csv -o c:\cours\etudiants.csv -d esipe -c etudiants

→ Export de la collection etudiants de la base esipe dans un fichier csv nommé etudiants créé dans le répertoire c:\cours.

Import/Export

- **mongoimport**: pour faire un import d'un fichier vers une collection

Exemple1: Fichier json

mongoimport -db esipe -c etudiants c:\cours\etudiantING2.json

→ Import dans la collection etudiants de la base esipe des données contenues dans le fichier json etudiantING2.

Exemple2 : Fichier json

mongoimport -db esipe -c etudiants --type c:\cours\etudiantING2.csv --headerline

→ Import dans la collection etudiants de la base esipe des données contenues dans le fichier json etudiantING2.

Remarque : l'option headerline permet de préciser que les noms des champs sont sur la première ligne.

Import/Export

Quelques options pour import/export

--help : pour afficher l'aide

--headline : Pour utiliser la première ligne comme en-tête (.csv)

--type : Type du fichier pour l'import (json,csv,tsv). Json par défaut

-u : utilisateur

-p: mot de passe

-d: nom de la base de donnée

-c: nom de la collection

-h: le nom de l'hôte pour la connexion

--p: port de connexion

Import/Export

BASE

- **mongodump**: pour faire un export de toute une base

Exemple:

mongodump -d esipe -o c:\base_esipe

→ Export de la base esipe dans le répertoire c:\base_esipe.

- **mongorestore**: pour restaurer une base à parrtir des données contenues dans un répertoire donné et issues de la commande mongodump.

mongorestore -d esipe2 c:\base_esipe

→ Crée la base esipe2 à partir des fichiers contenus dans le répertoire c:\base_esipe

Import/Export

BASE

- **mongodump**: pour faire un export de toute une base

Exemple:

mongodump -d esipe -o c:\base_esipe

→ Export de la base esipe dans le répertoire c:\base_esipe.

- **mongorestore**: pour restaurer une base à parrtir des données contenues dans un répertoire donné et issues de la commande mongodump.

mongorestore -d esipe2 c:\base_esipe

→ Crée la base esipe2 à partir des fichiers contenus dans le répertoire c:\base_esipe

Indexation - Performances

Index

- MongoDB supporte également la définition d'index.
- Ces index peuvent être définis sur un seul champ, ou encore être constitués de la combinaison de plusieurs champs.
- Ils auront les avantages et inconvénients habituels:
  - Fortes améliorations de performance pour la lecture si on se sert de ces champs pour accéder aux enregistrements.
  - Baisse de performance en écriture puisque l'index doit être modifié lorsqu'on ajoute ou modifie un enregistrement.

Index

- **Création d'index sur un champ**

db.collection.createIndex({nomChamp:1})

→ Création d'un index sur le champ appelé nomChamp.

nomChamp:1→ Les valeurs du champs sont triées par ordre croissant

Si nomChamp:-1 alors tri par ordre décroissant

```
> db.ordinateurs.createIndex({marque:1})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
```

•Nombre de champs sur lesquels un index existe avant l'opération: 1 (_id est indéxé par défaut)
•Nombre d'index après :2
Ok:1 → opération réussie

- **Création d'index sur plusieurs champs**

db.collection.createIndex({nomChamp1:1, nomChamp2:1})

→ Création d'un index sur le champ appelé nomChamp.

Index

- **Liste des index**

db.collection.getIndexes()

→ Le résultat est un document qui affiche les champs sur lesquels il y a u index

•La collection « ordinateurs » contient 3 index :

Index

Suppression des index

>db.collection.dropIndex({nomChamp:1})

→ Suppression d'un index sur le champ nomChamp

>db.collection.dropIndexes()

→ Suppression de tous les index (Seul l'index _id est conservé)

Exemple: Créons une collection

```
db.Noms.insert({"Nom":« Daniel"})
 db.Noms.insert({"Nom":« David"})
db.Noms.insert({"Nom":« Gore"})
db.Noms.insert({"Nom":« Dubois"})
db.Noms.insert({"Nom":« Couznich"})
db.Noms.insert({"Nom":« Antoneti"})
db.Noms.insert({"Nom":"Smith"})
db.Noms.insert({"Nom":« Abdoul"})
db.Noms.insert({"Nom":« Zineb"})
```

Exemple:

Le stockage est fait de façon aléatoire.

Pour savoir comment MongoDB traite la requête, on va insérer un nouveau document :

db.Noms.find({"Nom":"Abdoul"}).explain()

On note entre autres les informations suivantes:

- **cursor : BasicCursor** : Ce qui signifie que MongoDB va parcourir toute la collection
- **nscannedObjects** : MongoDB atrouvé 9 objets qui correspondent à la requête

AUTHENTIFICATION

Authentification

Par défaut la connexion à mongoDB est sans authetification ("open bar").

Avant de mettre en place un système d'authentification, il faut:

- Créer un utilisateur avec le rôle userAdminAnyDatabase
- Modifier le fichier de config de mongoDB en spécifiant -auth
- Relancer le serveur mongoDB avec l'option -auth

```
mongod --auth   --storageEngine wiredTiger
C:\MongoDB_Donnees\data\db
```

- Se connecter avec :

```
mongo --port 27017 -u "dan" -p "dan" --authentificationDatabase
"admin"
```

Authentification

MongoDB comme la plupart des bases de données dispose de différents rôles classés en deux catégories:

1/ Rôles d'utilisateur

a/ Read

L'utilisateur dispose uniquement d'un rôle de lecture sur les collections de la base et les collections système

b/ Read/Write

L'utilisateur dispose d'un rôle de lecture/écriture sur les collections de la base et de lecture sur les collections système

2/ Rôles Administrateur de base de données

a/ dbAdmin

a/ dbOwner

C'est le rôle qui confère les droits les plus élargis sur la base.

C'est une combinaison des droits des rôles read/write, dbAdmin et userAdminAnyDatabase

a/ UserAdmin

L'utilisateur qui a ce rôle sur une base a le droit de créer, modifier et attribuer des droits à un utilisateur sur une base

Authentification

3/ Rôles sur les clusters

a/ clusterAdmin

b/clusterManager

c/clusterMonitor

d/hostManager

4/ Rôles Backup / Restore

Ces rôles sont rattachés à l'utilisateur admin

5/ All-Database Rôles

a/readAnyDatabase

Ce rôle permet d'avoir les droits de lecture sur toutes les bases sauf la base local et config

b/readWriteAnyDatabase

Ce rôle permet d'avoir les droits de lecture1/écriture sur toutes les bases sauf la base local et config

c/userAdminAnyDatabase

Donne les mêmes privilèges que UserAdmin pour l'ensemble des bases sauf local et config

d/dbAdminAnyDatabase

idem pour dbAdmin pour l'ensemble des bases

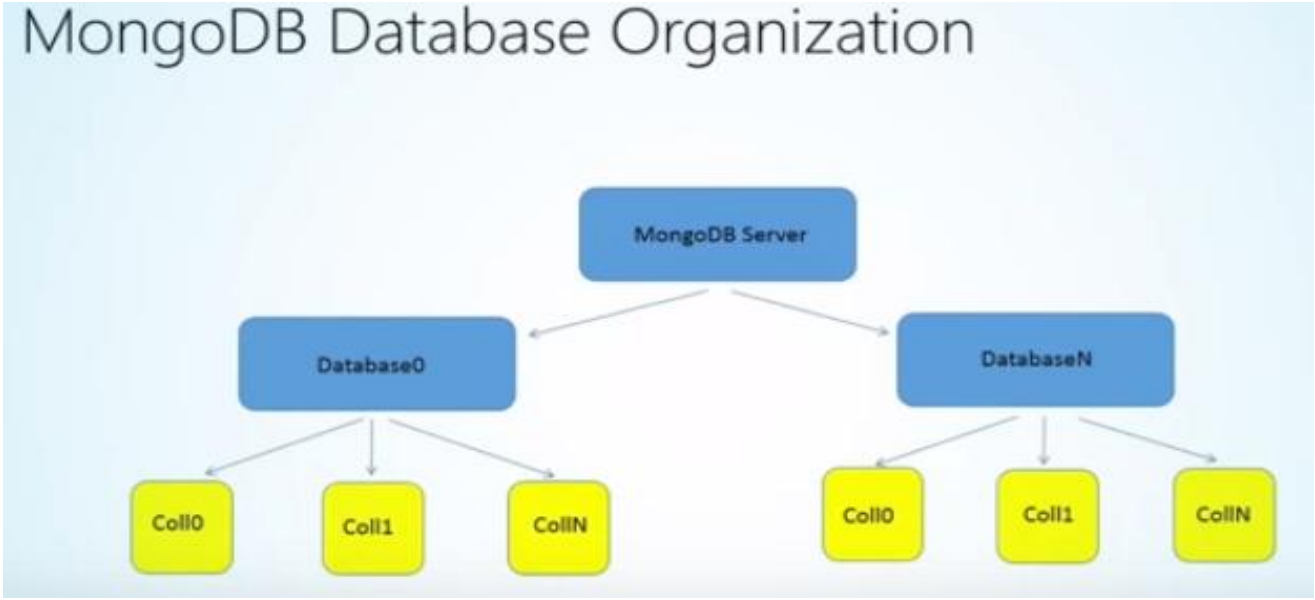
Réplication/Sharding

Réplication

Organisation des bases

Une instance mongoDB peut contenir:

- Une ou plusieurs bases de données
- Chacune des bases contient plusieurs collections
- Dans chaque collection, il y a plusieurs documents



Architecture: Réplication/Sharding

MongoDB a deux propriétés importantes en terme d'architecture interne:

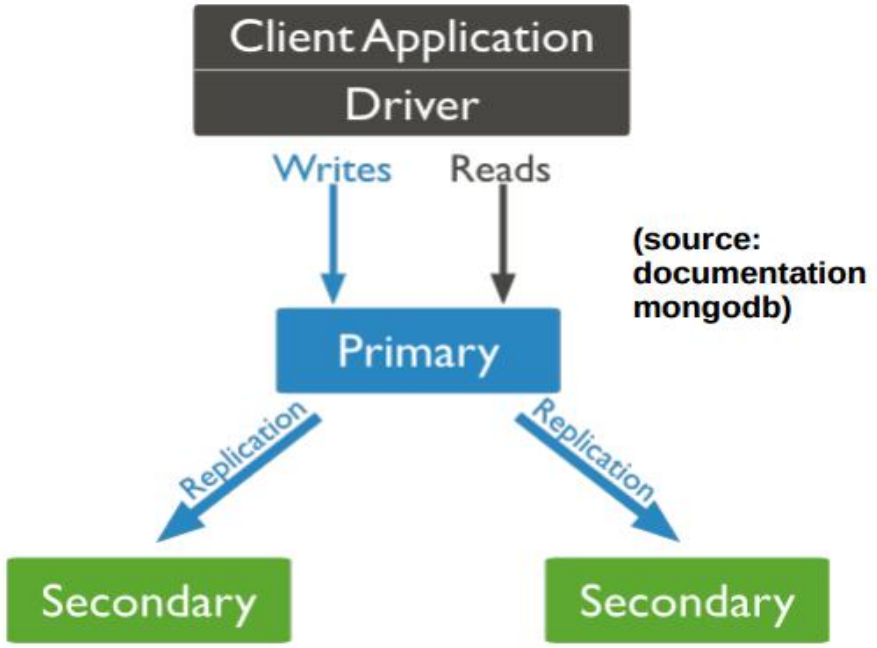
- Il peut être répliqué, pour éviter toute perte de données.
- Il est extrêmement scalable, par le biais d'une approche qu'on appelle le ***sharding***.
- L'une comme l'autre sont importantes dans le cadre d'une problématique « Big Data »;
- La première pour éviter toute perte de données critique, et la seconde parce qu'on est évidemment susceptible de stocker de très larges quantités de données au sein des bases de données.

NB:

Les mécanismes de réplication et de sharding sont détaillés plus loin dans le cours.

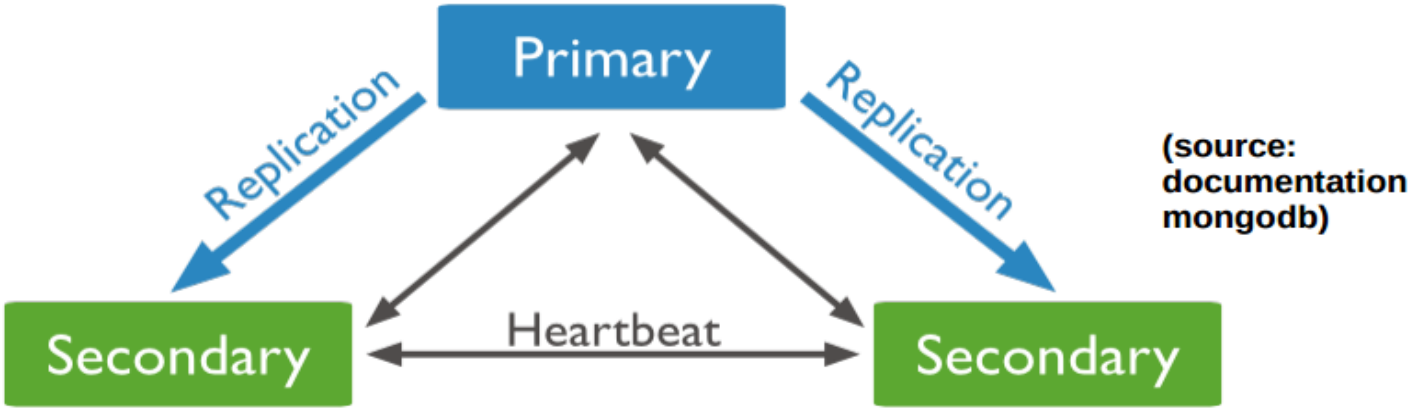
Architecture - **Réplication**

La redondance/réplication est possible par le biais de plusieurs instances du processus serveur de MongoDB (mongod).
Dans ce cas, on considère qu'une instance est primaire et que les autres sont secondaires.
Les nœuds secondaires répliquent toutes les opérations effectuées sur le nœud primaire via un mécanisme de logs (oplog)



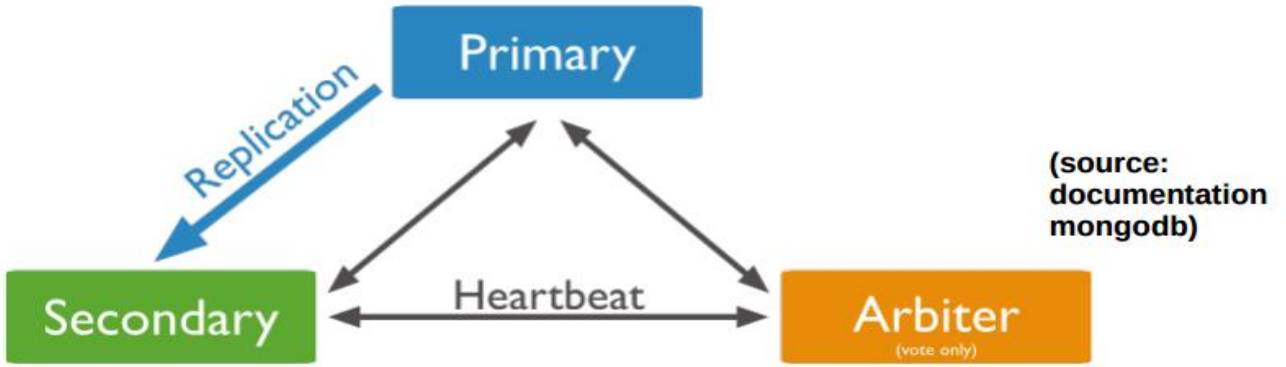
Architecture - **Réplication**

- Les nœuds communiquent en permanence par le biais de **heartbeats**.
- Si l'absence de heartbeats pour le nœud primaire est détecté, alors l'ensemble des nœuds restants élira un nouveau nœud parmi eux pour qu'il devienne le nouveau nœud primaire.



Architecture - **Réplication**

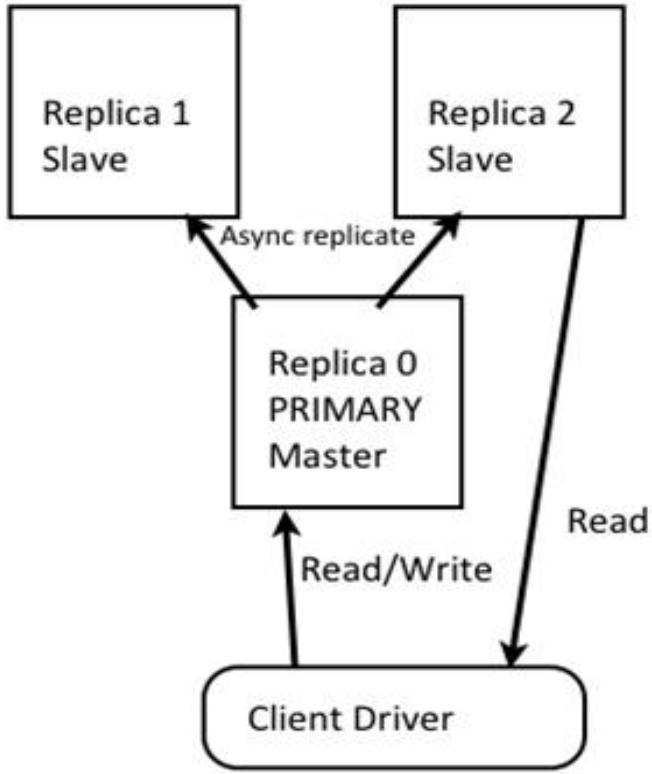
- On peut également définir des nœuds ***arbitres***.
- Il s'agit de nœuds légers ne répliquant pas les opérations du nœud primaire mais visant uniquement à assurer une majorité en cas de vote égal parmi les nœuds secondaires si le nœud primaire vient à tomber.



Architecture - **Réplication**

- Toutes les opérations d'écriture sont faites sur l'instance primaire. Puis répliquées sur les instances secondaires
- L'instance primaire est connue des drivers des langages de programmation
- L'instance dite primaire ou maître n'est pas fixe
- Si l'instance primaire tombe (est inaccessible), une autre instance primaire est élue.

- Classiquement, il y a 3 instances mongoDB sur un *replicat set*
- Si l'instance primaire tombe (est inaccessible), une autre instance primaire est élue.
  Et le driver du langage de programmation sait l'identifier



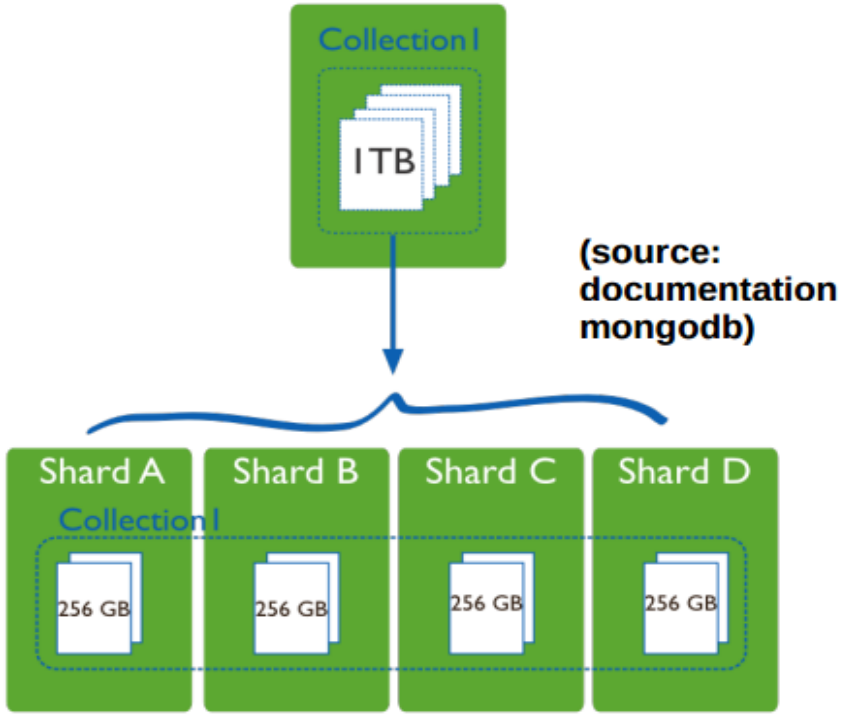
Sharding

Architecture - Sharding

- Le sharding est la technique utilisée pour assurer la **scalabilité** au sein de MongoDB.
- Elle consiste à avoir plusieurs instances de MongoDB (généralement plusieurs groupes de nœuds primaires+secondaires), concrètement indépendantes mais logiquement « unifiées ».
- Chaque instance aura la charge de stocker une partie des données.
- L'ensemble des instances contiendra ainsi la totalité des données et sera considéré conceptuellement comme la base de données unique « logique » MongoDB.

Architecture - Sharding

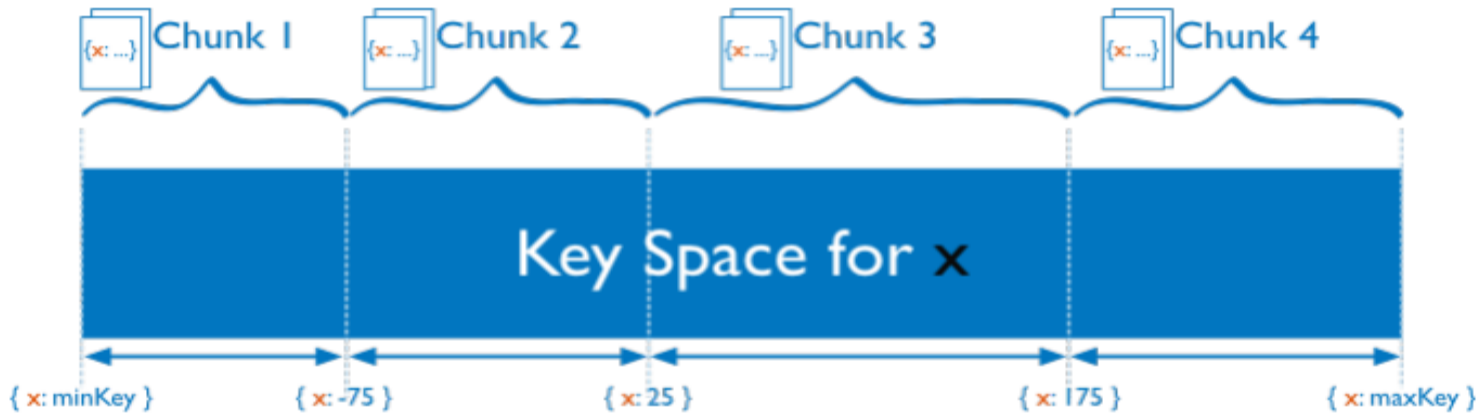
- Ici, on stocke 256 GB de données au sein de chacune des instances MongoDB.
- Conceptuellement, en revanche, on considère l'ensemble de ces instances comme un seul SGBD MongoDB; et on adressera le SGBD comme s'il s'agissait d'une instance unique.
- C'est MongoDB qui s'occupera de rediriger les écritures/lectures vers l'instance / le shard approprié.



Architecture - Sharding

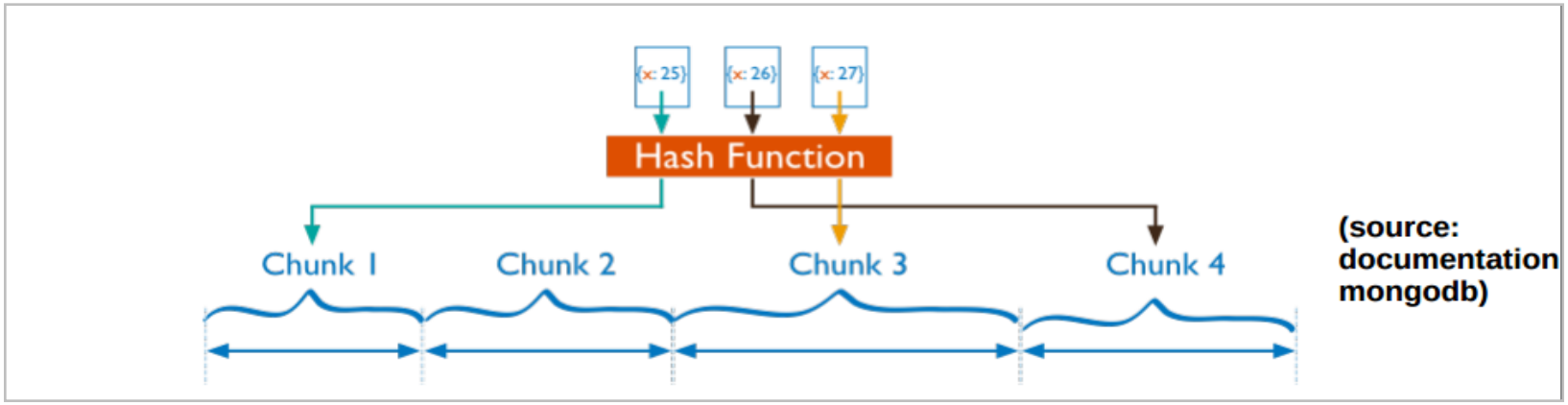
- Afin d'assurer le sharding, on désignera un champs particulier des documents concernés (par exemple, mais pas nécessairement et même pas optimalement l'ObjectID des documents) dont la valeur déterminera le shard qui devra contenir le document. On désigne ce champs sous le terme de shard key.
- Pour déterminer le shard à adresser, on pourra utiliser deux approches:
  - Une approche basée sur la valeur elle-même, par exemple un ID numérique. On pourrait ainsi par exemple configurer le cluster de telle sorte que les ID de 0 à 10000 soients stockés sur le shard 1, les IDs de 10001 à 20000 sur le shard 2, etc.
  - Une approche basée sur un hash de la valeur, afin d'assurer une meilleure répartition dans l'espace des possibilités.

- Si on choisi de se baser sur la valeur, alors deux valeurs proches seront probablement au sein du même shard.
- Selon les valeurs rencontrées, l'utilisation et le champs choisi, la répartition sera possiblement variable entre les shards



(source: documentation mongodb)

- Si en revanche on choisi de se baser sur un hash de la valeur de la shard key, alors deux valeurs proches seront probablement dans des shards différents, puisque la fonction de hashing répandra les valeurs dans l'espace de possibilités.
- La répartition entre les shards sera alors plus optimale.

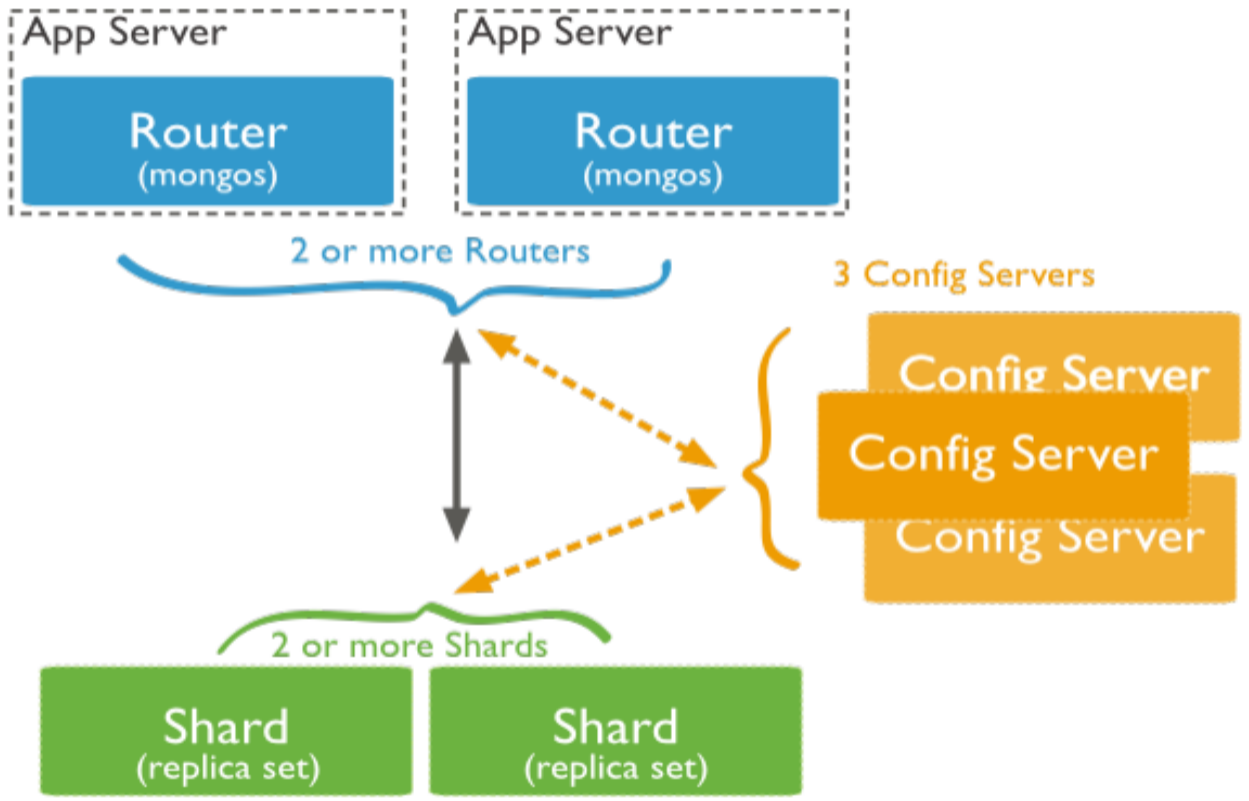


Architecture - Sharding

En terme d'architecture logicielle, MongoDB assure cette mécanique de sharding par le biais de trois composantes:

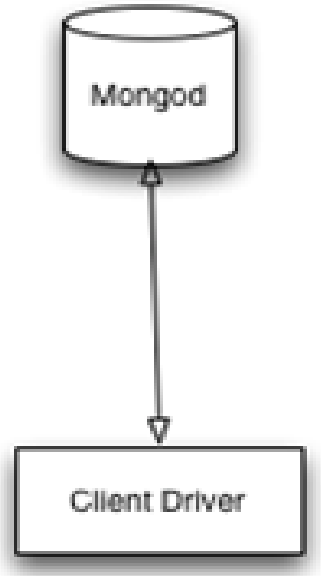
- Un ou plusieurs **routeurs de requêtes**. Il s'agit de serveurs indépendants (processus mongos); c'est sur ces serveurs que les clients se connecteront, et ce sont ces serveurs qui redirigeront les écritures et lectures sur le bon shard.
- Un ensemble de **trois serveurs de configuration**. Ils constituent ensemble la table de mappage assurant l'associations entre les valeurs de la clef de sharding et le shard à adresser, et sont en communication avec les routeurs et les shards.
- Enfin, les **shards eux-mêmes** (composé chacun à minima d'une instance de serveur MongoDB mais plus généralement d'un ensemble d'instances primaires et secondaires pour assurer la réplication).

Architecture - Sharding



(source: documentation mongodb)

Différentes configurations

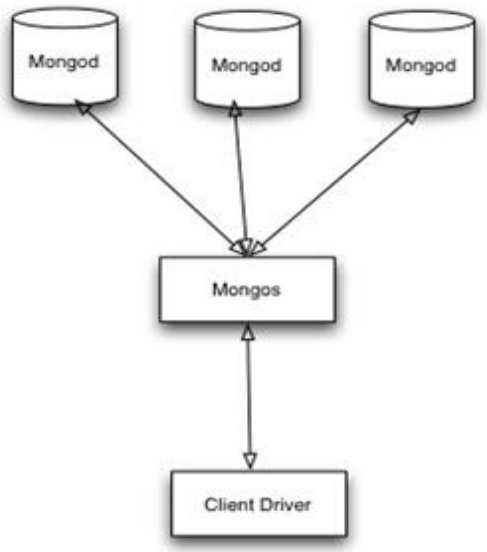


- 2 acteurs:
  - Le processus mongod
  - Le client
- Un client connecté directement au serveur
- C'est la configuration de la plupart des applications

Si l'application nécessite d'accélérer les lectures, alors ajouter un *replicat set*

Si l'application nécessite d'accélérer les écritures, il faut passer en auto-*sharding*

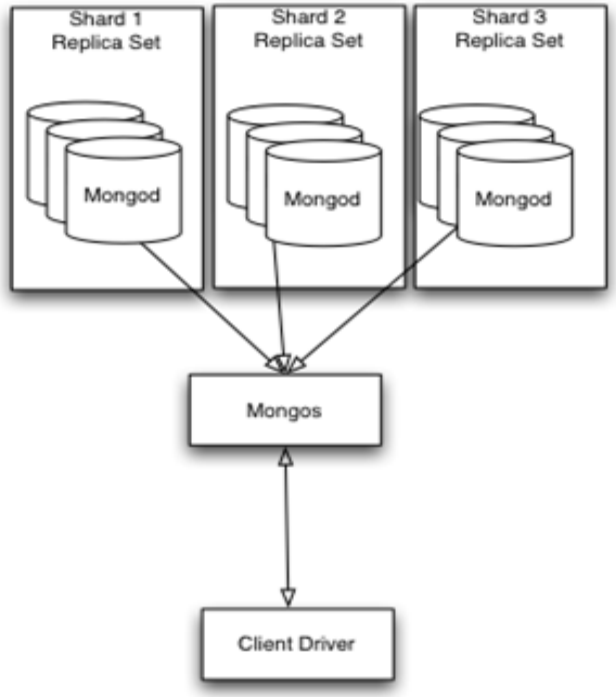
Différentes configurations



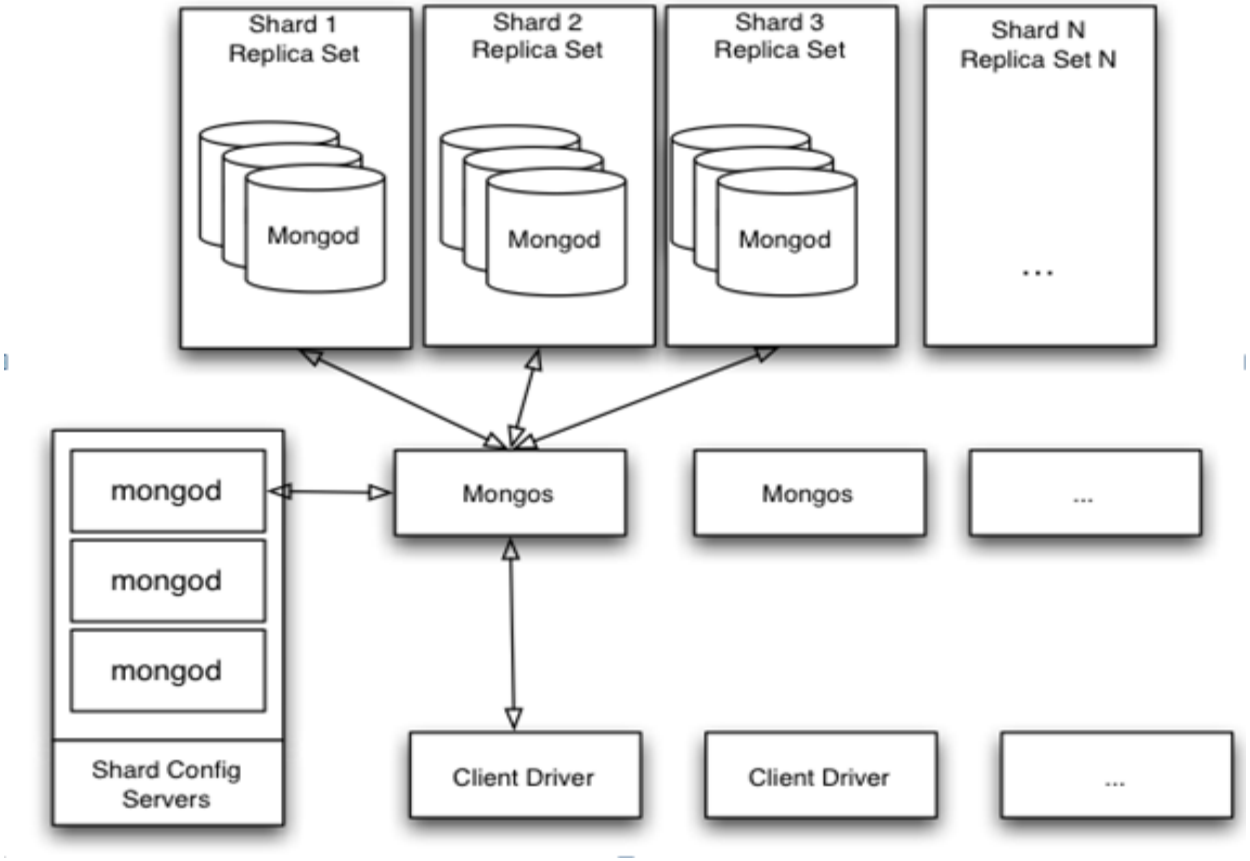
- 3 acteurs:
  - Le processus *mongod* (plusieurs instances)
  - Le routeur *mongos*, pour router les écritures vers la bonne instance mongod
  - Le client

- Si l'application nécessite d'accélérer les lectures, alors ajouter un *replicat set*
- Si l'application nécessite d'accélérer les écritures, il faut passer en auto-*sharding*

Architecture Réplication-Sharding



Architecture - déploiement complexe



TP Réplication / Sharding

(Cf. Document Réplication /Sharding)

TP Réplication / Sharding

Moteurs de stockage

1. Stockage WiredTiger

Disponible Depuis la version 3.0 sur les machines 64 bits.

Mode de stockage par défaut depuis la version 3.2 si aucune option n'est précisée *--storageEngine* .

Dans ce mode de stockage, le contrôle des écritures se fait au niveau document.

Par conséquent, plusieurs utilisateurs peuvent écrire, modifier des documents différents d'une même collection au même moment.

Les checkpoints sont des points de restauration

Moteurs de stockage

2. Stockage MMAPv1

Se sert uniquement du journal.

Utilisé par défaut dans les anciennes versions de mongoDB

3. Stockage In-Memory

Il est conseillé de mettre le répertoire du journal sur un autre disque que les données

Moteurs de stockage - WiredTiger

1. Journal

2. Snapshots and Checkpoints

Les checkpoints sont des points de restauration

3. Compression

4. Mémoire

- 50% de la RAM minus 1 GB ou 256 MB.

Restauration

- **Mongodump : export des données**

Syntaxe:

mongodump --out <cheminDestination>

Mongorestore : import des données

mongorestore < *cheminDestination* >

Modification moteurs de stockage

1. Standalone

mongodump --out <cheminDestination>

Mongorestore : import des données

mongorestore < *cheminDestination* >

MongoDB et Java

- Comme indiqué précédemment, MongoDB propose entre autres une API Java (mise à disposition sous la forme d'un .jar) pour accéder à une instance MongoDB depuis un programme Java.
- Cette API est téléchargeable depuis le site officiel.
- D'une manière générale, toutes les fonctions vues précédemment via le shell sont également utilisables en Java.

MongoDB et JAVA

- Avant toute opération, on devra d'abord créer un objet de type MongoClient.
- La classe en question: *com.mongodb.MongoClient*
- Par exemple:

```
MongoClient client = new MongoClient();
```

- On pourra ici également spécifier l'IP/nom d'hôte de l'instance à laquelle on souhaite se connecter, et le port TCP qui lui correspond.

Par exemple:

```
MongoClient client = new MongoClient("localhost", 27017);
```

MongoDB et JAVA

- Ensuite, on sélectionnera la base de données voulue par le biais de la méthode getDatabase() de cet objet, qui renvoie un objet de type MongoDatabase.
- Par exemple:
- *MongoClient client = new MongoClient("localhost", 27017);*
  MongoDatabase db = client.getDatabase(« ESIPE");
- … Ici pour sélectionner la base de données « ESIPE ».
- A noter que la fonction est strictement identique au « use » du shell MongoDB; et qu'une base de données non existante sera donc automatiquement « créée ».

MongoDB et JAVA

- D'une manière générale, le type utilisé pour formater les documents *MongoDB est la classe org.bson.Document*.
- C'est celle qui est utilisée pour construire un document BSON; qu'il s'agisse d'un document au sens propre ou d'un argument aux différentes fonctions de MongoDB (par exemple find(), ou encore sort()).
- La méthode la plus utilisée de cette classe est la méthode append().
- Exemple:

```
Document doc=new Document().append("nom", "Smith");
doc.append("prenom", "John");
```

Ce qui créerait un document: {nom: "Smith", prenom: "John"}

Principe du paradigme MapReduce : Syntaxe générale

```
>db.collection.mapReduce(
function() {emit(key,value);}, //la fonction map
function(key,values) {return reduceFunction}, { // La fonction reduce
out: collection,
query: document,
sort: document,
 limit: number
 }
)
```

Quelques outils graphiques

1) Edda
2) UMONGO aka JMONGO Browser
3) MongoExplorer
4) Mongovision
5) MongoVUE (payant)
6) Opricot
7) mViewer
8) PHPMoAdmin
9) RockMongo
10) Meclipse
11) GEnghis
12) Humongous
13) Monad management for MongoDB
14) Robomongo (gratuit)

Modélisation des données dans MongoDB

Les relations représentent les liens qu'ont les documents dans mongoDB.

Comme dans les SGBD traditionnels, elles peuvent être de différents types.

On en distingue 4:

- **1:1 (one to one)** : Un objet A peut être en relation à 1 et un seul objet B
- **1:N (one to many)**: Un objet A peut être en relation à 1 ou plusieurs objets B
- **N:1 (many to one)** : Plusieurs objets A peuvent être en relation avec 1 et 1 seul objet B
- **N:N (many to many)** : Plusieurs objets A peuvent être en relation avec plusieurs objets B

Modélisation des données dans MongoDB

- Le modèle de données doit être spécifié selon les besoins de l'uitlisateur
- Il faut identifier les objets qui peuvent être mis ensemble et ainsi les mettre dans un même document.
- Si des objets ne peuvent être mis ensemble, alors les mettre dans des documents séparés. Toutefois, il faut s'assurer que l'on a besoin de jointures pour pour les avoir ensemble dans la suite
- Ne pas oublier que dans MongoDB, ce qui compte, c'est le temps d'exécution des requêtes et non pas l'espace disque. Les données peuvent donc être dupliquées.

Modélisation des données dans MongoDB

Contrairement aux SGBD relationnels, mongoDB ne supporte pas les jointures.

Pour modéliser les données dans MongoDB, deux approches sont possibles:

Documents imbriqués ou documents liés

Voici quelques éléments à prendre en compte lors de la modélisation

Documents imbriqués

- Structure des données simples
- Taille des documents limités à 16 Mo -- > (Au délà voir GridFS)
- Taille de document plus important
- Quelle est la fréquence de mise à jour?

Documents liés

- Structure des données complexe
- Pas de limite de taille
- Plus de documents de petite taille
- Quels sont les besoins en maintenance?

Quelques infos complémentaires

Document BSON

- Taille max d'un document : 16 Mo
- Nombre de documents imbriqués : Jusqu'à 100

Collection

- Nb de documents dans une collection : illimité (Sauf pour une collection plafonnée-> max: 2^{32} documents
- Nb d'index par collection : 64
- Nb de champs dans un index composé : 31

Replica set / Sharding

- Nb maximum de nœuds dans un cluster : 50
- Nb de nœud ayant droit de vote : 7
- Nb de champs dans un index composé : 31

FIN DU COURS