TP's

nosql Module

Diginamic, Loic Guillois

Version 1.0, 2018-01-24

Table des matières

Chapitre 1. MongoDB

1.1. Découverte de la solution cloud Atlas

1.1.1. Mise en pratique

MongoDB offre la possibilité de créer une instance gratuite. Cette instance peut être très pratique pour effectuer des essais ou tout simplement pour développer un projet avec un budget limité. Cette offre ne demande pas de carte bancaire, vous pouvez donc créer un instance en suivant les instructions à l'adresse suivante : https://www.mongodb.com/cloud/atlas

Quelles sont les limitations de cette instance ?

A partir de quel volume de donnée devrez vous passer sur une instance payante ?

1.1.2. Estimation des coûts d'hébergement

MongoDB met à votre disposition toutes les informations permettant de calculer le coût d'une infrastructure Atlas. Nous allons donc l'utiliser pour répondre à la question suivante.

Quel serait le TCO d'un cluster MongoDB avec l'architecture suivante ?

- Vos utilisateurs sont majoritairement à Paris : 90%
- Une partie de votre trafic provient du reste du monde : 10%
- Bande passante de votre application: 10Go par mois
- Base de donnée de 30Go
- Mise en place du sharding avec 5 shards
- Mise en place de la réplication avec 3 serveurs par replica set
- Sauvegarde

Attention: il y a plusieurs réponses possibles, essayez d'utiliser différentes configuration et projetez vous sur les avantages et les inconvénients.

Vous souhaitez améliorer la capacité à monter en charge (scalabilité) de votre système. Quelles actions pouvez-vous réaliser ? Quelle est le coût de ces actions ?

Vous souhaitez améliorer la disponibilité de votre architecture. Quelles actions pouvez-vous réaliser ? Quelles est le coût de ces actions ?

Pour l'ensemble de ces questions construisez un document synthétique avec tout les éléments que vous jugerez nécessaire (tableaux, schéma, explications…).

Pour aller plus loin, vous pouvez suivre le cours en ligne gratuit : https://university.mongodb.com/courses/M123

1.2. Mise en place de la réplication

L'objectif de ce travaux pratique est de mettre en oeuvre un replica set comprenant 3 serveurs. Rappelez vous, un replica set MongoDB doit contenir un nombre impair de serveurs et au minimum 3.

Dans un premier temps, nous allons créer 3 dossiers qui vont permettre de stocker les données sur le disque dur. Assurez vous que l'espace disque est suffisant.

Quel est l'espace disque disponible sur la partition que vous utilisez?

```
# mkdir /data/r0
# mkdir /data/r1
# mkdir /data/r2
```

Nous allons maintenant installer MongoDB. Vous pouvez le télécharger librement dans sa version Community Server. La version actuelle est la 3.6.2.

Une fois installé, nous pouvons lancer les 3 instances MongoDB. On va fournir à chacune d'entre elles 3 paramètres : le port TCP d'écoute, le chemin vers le dossier de données et le nom du replica set. Pour cet exemple, nous avons décidé de l'appeler arbitrairement foo.

Attention, vous devrez ouvrir plusieurs terminaux. Chaque instance va monopoliser un terminal. Il y aura un avantage à cela: vous aurez accès directement aux logs de chacun d'entre eux directement.

```
# mongod --port 27017 --dbpath /data/r0 --replSet foo
# mongod --port 27018 --dbpath /data/r1 --replSet foo
# mongod --port 27019 --dbpath /data/r2 --replSet foo
```

Nous allons ensuite lancer la console MongoDB. Celle ci va se connecter automatiquement sur le port par défaut du serveur, à savoir 27017. La console sera donc connecté sur notre première instance.

```
# mongo
```

Nous allons configurer le replica set. Pour se faire, nous créons un objet JavaScript comportant notamment les identifiant et les hôte (couple adresse IP/port TCP).

```
> cfg = {
_id : "foo",
members : [
{ _id : 0, host : "localhost:27017"},
{ _id : 1, host : "localhost:27018"},
{ _id : 2, host : "localhost:27019"},
] }
```

Une fois l'objet créé on peut l'utiliser en appelant la fonction Javascript rs.initiate.

```
rs.initiate(cfg)
```

Le cluster est créé. Vous pouvez observez ce qui se passe dans les consoles de chacun des serveurs. Vous devriez pour lire des messages dans les logs vous informant de la création du replica set.

Vous pouvez désormais récupérer les informations sur le replica set avec la commande suivante:

```
rs.status()
```

Quel est le serveur primaire ?

Qui sont les serveurs secondaires ?

Nous allons maintenant essayer d'écrire des données en base.

```
use test
db.user.insert({name:'Loic'})
```

Observez ce qui se passe dans les logs des différents serveurs.

Toujours dans la console MongoDB, essayez de lire les données:

```
db.user.findOne()
```

Nous allons maintenant nous connecter sur un serveur secondaire. Ouvez un nouveau terminal et tapez la commande suivante:

```
mongo --port 27018
```

Nous allons vérifier que les données ont bien été répliquées. Essayez de nouveau de lire les données:

```
use test
db.user.findOne()
```

Cela dit, il nous ai impossible d'écrire des données sur un serveur secondaire. Si nous voulons l'autoriser, nous devons appeler la fonction JavaScript suivante :

```
rs.slaveOk()
```

Nous pouvons désormais écrire des données sur les serveurs secondaires.

```
db.user.insert({name:'Elvis'})
```

Vous savez maintenant comment lire les données. Vérifiez que les données sont écrites sur les autres serveurs.

Pour aller plus loin, visitez la page suivante : https://docs.mongodb.com/manual/replication/

1.3. Mise en place du sharding

L'objectif de ce travaux pratique est de mettre en oeuvre le sharding en réutilisant le replica set que nous venons de créer. Rappelez vous, le sharding consiste à répartir les données sur l'ensemble des serveurs.

Dans un premier temps, nous allons devoir créer le replica set spécifique à la configuration de notre shard (un shard est un ensemble de serveur sur lequel le sharding a été mis en place).

Nous allons créer 3 dossiers qui vont permettre de stocker les données sur le disque dur. Assurez vous que l'espace disque est suffisant. Contrairement aux serveurs classiques, les serveurs de configurations vont stockées très peu de données.

```
# mkdir /data/configdb0
# mkdir /data/configdb1
# mkdir /data/configdb2
```

Nous allons lancer les 3 instances MongoDB de configuration. On va fournir à chacune d'entre elles 3 paramètres : le port TCP d'écoute, le paramètre configsvr et le chemin vers le dossier de données.

Attention, vous devrez également ouvrir plusieurs terminaux. Chaque instance va monopoliser un terminal de la même manière en vous permettant un accès aux logs.

```
# mongod --port 27020 --configsvr --dbpath /data/configdb0
# mongod --port 27021 --configsvr --dbpath /data/configdb1
# mongod --port 27022 --configsvr --dbpath /data/configdb2
```

Nous allons pouvoir lancer le serveur mongos. Celui ci prend pour paramètre un port TCP et la liste des serveurs de configuration.

```
# mongos --port 27023 --configdb foo/localhost:27020,foo/localhost:27021,foo/localhost:27022
```

On se connecte sur ce serveur.

```
$ mongo --port 27023
```

On active le sharding en effectuant un appel de fonction JavaScript.

```
> use admin
> db.runCommand({addshard : "foo/localhost:27017,localhost:27018,localhost:27019"})
```

Avant de tester le sharding, nous devons également l'activé sur chaque base de données et chaque collection. Concernant la collection, nous devons également préciser quelle clef utiliser. La bonne pratique est d'utiliser l'attribut _id.

```
> db.runCommand({enablesharding: "test"})
> db.runCommand({shardcollection: "test.hits", key:{"_id":1}})
```

On peut maintenant injecter un grand volume de données:

```
> use test
> for(i=0;i<50000;i++) {
db.hits.insert({tag: "search", ctx : ["blah"] });
db.hits.insert({tag: "video", ctx : [parseInt(Math.random() * 100), "play"] });
db.hits.insert({tag: "video", ctx : [parseInt(Math.random() * 1000), "play"] });
}
```

Cette requêtes va durer plusieurs secondes. Vous avez le temps d'aller observer ce qui se passe dans les différentes console au niveau des logs.

Vous pouvez à tout moment suivre l'état du replica set avec la commande que nous avons déjà utiliser:

```
rs.status()
```

Vous pouvez également suivre l'état du shard:

```
sh.status()
```

Quelles sont les informations importantes qui sont affichées ?

Pour aller plus loin, visitez la page suivante : https://docs.mongodb.com/manual/sharding/

1.4. Usage de base

Nous allons prendre en main les requêtes de base de MongoDB. Le langage de requête de MongoDB est très riche et nécessite de longues heures de formation et de pratique pour être maitrisées. Nous allons ici manipulez les fonctions principales qui anime la vie quotidienne d'un développeur MongoDB.

La commande suivante permet de lister les bases de données:

```
show dbs
```

Quelles sont les bases que vous voyez ?

La commande suivante permet d'utiliser une base de donnée. Avec MongoDB vous n'avez pas besoin de créer une base. Vous la sélectionnez et vous l'utiliser simplement : elle sera créer lorsque des données seront écrites.

```
use formation
```

Vous pouvez listez les collections avec la commande suivante:

```
show collections
```

De la même manière, on ne crée pas de collection. Il suffit d'écrire des données dans la collection voulue pour que celle ci soit créé. Ici nous créons un document avec différents types de données: texte, date, valeur numérique.

```
db.users.insert({name: 'Loic', date : new Date(), dept: 44})
```

Vous avez compris le principe. Maintenant insérez une dizaine de documents dans cette collection, en ajoutant par exemple un champs firstname.

Voici quelques exemples de requêtes de lecture. A vous d'interprêtez ce qu'elles font !

```
db.users.find()
db.users.findOne()
db.users.find({dept : 44})
db.users.find({dept : { $gte: 44}})
db.users.find().sort({date : -1})
db.users.find().limit(5)
db.users.find().limit(2).pretty()
```

Les requêtes permettant de comptez ont la même syntaxe que celles du find.

```
db.users.count()
db.users.count({dept : 44})
```

En vous référant à la documentation, écrivez les requêtes qui permettent de répondre aux questions suivantes:

- Combien y a-t-il de documents qui ont un département supérieur au numéro 33 ?
- Quelle est la liste des documents n'ayant pas de champs firstname ?
- Quelle est la liste des documents trié par nom (ordre alphabétique) en la limitant à 4 éléments ?

La commande suivante permet de mettre à jour un document.

```
db.users.update({dept : 44}, { $set: { label : 'Loire Atlantique' }})
```

Attention à l'importance du mot clef $set. Sans lui, c'est l'intégralité du document qui est remplacé.

A noté également que par sécurité, cette commande ne met à jour qu'un seul document par défaut. Pour mettre à jour tout les documents, on ajoutera un paramètre :

```
db.users.update({dept : 44}, { $set: { label : 'Loire Atlantique' }}, { multi: true })
```

Pour supprimer des documents, on utilise la commande remove:

```
db.users.remove({ dept: 44 });
```

La commande suivante supprime les documents d'une collection. Cela permet de conserver les index qui ont été créé.

```
db.users.remove({})
```

Pour détruire une base de donnée on utilisera la commande suivante. Attention, elle va supprimer à la fois les données et les index. Si vous travaillez sur un replica set, la suppression sera effective sur l'ensemble des serveurs.

```
db.dropDatabase();
```

Avec MongoDB, vous pouvez stocker des documents JSON. Vous pouvez notamment ajouter du contenu arborescent ou des tableaux:

```
db.article.insert({title: 'Hello', author : { name : 'Loic', city: 'Nantes'}, tags : [ 'actualite', 'politique']})
```

Pour aller plus loin, rendez-vous sur la page suivante: https://docs.mongodb.com/manual/crud/

1.5. Aggregation framework

Dans ce TP, nous allons mettre en oeuvre le framework d'agrégation. Bien que MongoDB propose une implémentation Map/Reduce pour le traitement de données en masse, le framework d'agrégation est plus pratique à utiliser et offre plus de souplesse. Il y a beaucoup de chose à comprendre pour en maîtriser toute la puissance. Nous allons ici effectuer quelques traitements de base pour vous permettre d'en comprendre les base de fonctionnement.

Pour commencer, nous allons télécharger le jeu de donnée comportant des informations sur une lsite exhaustive de restaurants aux Etats-Unis. On y retrouve le type de cuisine ainsi que des notes et une localisation.

Téléchargez le fichier https://raw.githubusercontent.com/mongodb/docs-assets/primer-dataset/primer-dataset.json

Nous pouvons maintenant l'importer en base. Ici, nous importons les données dans la collection restaurants de la base de donnée test. Nous ajoutons le paramètre --drop pour partir d'une base vierge avant l'import.

```
mongoimport --db test --collection restaurants --drop --file ~/downloads/primer-dataset.json
```

On se positionne sur la base de donnée et on regarde les données importées:

```
use test
db.restaurants.findOne()
db.restaurants.count()
```

Voici quelques commandes d'agrégation. Executez les unes à unes en essayant de comprendre ce qu'elles réalisent.

```
db.restaurants.aggregate({$group : { _id : "$cuisine"}})
db.restaurants.aggregate({$group : { _id : "$cuisine", count : { $sum : 1}}})
db.restaurants.aggregate({$group : { _id : "$cuisine", count : { $sum : 1}}}, { $sort : { count : -1 }})

db.restaurants.aggregate({$unwind: "$grades"})
db.restaurants.aggregate({$unwind: "$grades"}, {$group : { _id : "$cuisine", grade: {$avg :
"$grades.score"}}}, { $sort : { grade : -1 }})
```

Chapitre 2. Redis

2.1. Prise en main de Redis

Redis est une base de données clef/valeur. L'usage de base est très simple et ne permet pas de stocker des données aussi structurée que dans MongoDB. Aujourd'hui, c'est la base clef/valeur la plus utilisée. Le jeu de commande disponible s'est enrichi au fur et à mesure des années. Vous pouvez les découvrir ici : https://redis.io/commands

Redis est disponible sur les machines Unix : Linux, Mac OS. A ce jour, Windows n'est pas supporté officiellement. Vous pouvez le télécharger à l'adresse suivante : https://redis.io/download. La version actuelle est la 4.0.

Une fois, téléchargez vous pouvez extraire l'archive.

```
# tar -xvzf redis-4.0.7.tar.gz
```

Ensuite vous pourrez lancer le serveur. Il n'a pas besoin de configuration. Par défaut, il stockera les données en mémoire vive et écoutera le port TCP 6379

Attention, le serveur va monopoliser la console avec l'affichage des logs.

```
# redis-server
```

Dans un autre terminal, lancez le client. Il va se connecter automatiquement sur le port par défaut :

```
# redis-cli
```

Afin de vérifier que notre client est bien connecté avec le serveur, on peut utiliser la commande PING:

```
> PING
PONG
```

Essayez de couper le serveur et exécutez de nouveau cette commande.

On peut ensuite utiliser les fonctions les plus simples de Redis qui illustre le principe de clef/valeur:

```
> SET USER  "Loic"
OK

> GET USER
"Loic"
```

On peut renommer une clef:

```
> SET USER USER1
```

On peut supprimer une clef:

```
> DEL USER1
(integer) 1
```

Redis nous indique que l'on a supprimé un élément.

On peut également vérifier l'existance d'une clef en base:

```
> EXISTS USER1
(integer) 0
```

Redis permet de manipuler des compteurs facilement:

```
> SET COUNT 0
OK
> INCR COUNT
(integer) 1
> INCR COUNT
(integer) 2
> DECR COUNT
(integer) 1
```

Redis permet également de manipuler des listes, des ensembles… Nous n'aurons pas le temps de tout aborder !

L'un des mécanismes important apporté par Redis est l'implémentation du design pattern publish/subscribe. Il permet une communication asynchrone entre les clients connectés. Son utilisation peut permettre l'implémentation simplifiée d'un mécanisme de discussion ou de notification par exemple.

```
> SUBSCRIBE redisChat
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "redisChat"
3) (integer) 1
```

Dans un autre terminal, on lance également un client pour dialoguer avec l'autre client.

```
# redis-cli
> PUBLISH redisChat "Super formation"
(integer) 1
```

Le retour nous indique à combien de client le message a été envoyé. Dans notre exemple, seul un seul client a reçu le message.

Sur la console de l'autre client, vous allez voir apparaître le message :

```
2) "redisChat"
3) "Super formation"
```

Ce mécanisme tient très bien la montée en charge. Vous pouvez essayer de connecter de nombreux clients.

Redis supporte les transactions. Pour ce faire on utiliser deux mots clef: MULTI et EXEC. Toutes les commandes sont mises en attente après une commande MULTI, jusqu'à ce que l'on appel la fonction EXEC.

```
> MULTI
OK
> SET tutorial redis
QUEUED
> GET tutorial
QUEUED
> INCR visitors
QUEUED
> EXEC
1) OK
2) "redis"
3) (integer) 1
```

Redis nous précise bien que les requêtes ont été mise en attente avec le retour QUEUED. Suite à l'appel de la commande EXEC, celles ci sont exécuté dans l'ordre.

Il existe également de nombreuses commandes de maintenance. La plus importante à retenir est la suivante:

```
> INFO
```

Quelles sont les informations affichées ?

On peut également se limiter aux informations d'une seule section, comme par exemple ici les informations sur la mémoire:

```
> INFO memory
```

2.2. Cluster

Redis permet de créer facilement un cluster. Contrairement à MongoDB, on ne choisi pas entre la mise en place du sharding et un simple replica set. Redis est auto-shardé, c'est à dire qu'à partir du moment où l'on crée un cluster, le sharding est en place, sans possibilité de le configurer particulièrement. La réplication peut par contre se configurer au travers d'un ratio (une valeur numérique entière).

Dans un premier temps, nous allons devoir créer plusieurs fichiers de configuration. Comme nous allons créer un cluster avec 6 noeuds, vous allez pouvoir créer autant de fichier numéroté de redis0.conf à redis5.conf

```
port 7000
cluster-enabled yes
cluster-config-file nodes.conf
cluster-node-timeout 5000
appendonly yes
```

Attention, vous devrez attribuer un numéro de port différent pour chaque serveur. Dans notre cas, nous ferons simple (port 7000 pour redis0.conf, port 7001 pour redis1.conf etc.)

Le paramètre appendonly permet de forcer l'écriture sur disque dur.

Ensuite, nous allons créer les dossiers de données :

```
mkdir cluster-test
cd cluster-test
mkdir 7000 7001 7002 7003 7004 7005
```

Nous pouvons maintenant lancer nos différents noeuds. Vous allez devoir ouvrir 6 terminaux et lancer manuellement chacun des serveurs. Par exemple, pour le premier:

```
cd 7000
redis-server ../redis0.conf
```

Une fois lancés, les serveurs sont disponibles et fonctionnels. Cependant nous n'avons pas configurer pour le moment le cluster. Nous avons besoin pour cela d'un outil complémentaire. Pour continuer, téléchargez le fichier http://download.redis.io/redis-stable/src/redis-trib.rb

Nous allons procéder à l'installation des dépendances :

```
sudo apt-get update
sudo apt-get install ruby
sudo gem install redis
```

Et donner les droits en exécution à notre outil:

```
chmod a+x redis-trib.rb
```

Nous pouvons désormais créer notre cluster en une seule commande:

```
./redis-trib.rb create --replicas 1 127.0.0.1:7000 127.0.0.1:7001 127.0.0.1:7002 127.0.0.1:7003
127.0.0.1:7004 127.0.0.1:7005
```

Nous pouvons désormais tester notre architecture:

```
$ redis-cli -c -p 7000
redis 127.0.0.1:7000> set foo bar
-> Redirected to slot [12182] located at 127.0.0.1:7002
OK
redis 127.0.0.1:7002> set hello world
-> Redirected to slot [866] located at 127.0.0.1:7000
OK
redis 127.0.0.1:7000> get foo
-> Redirected to slot [12182] located at 127.0.0.1:7002
"bar"
redis 127.0.0.1:7000> get hello
-> Redirected to slot [866] located at 127.0.0.1:7000
"world"
```

Vous pouvez essayer de couper des serveurs et d'observer les logs des serveurs. Essayez de faire des écritures et relancer les serveurs coupés, tout en observant les logs.

Chapitre 3. Cassandra

3.1. Prise en main Cassandra

Cassandra est une base de donnée orientée colonnes. La syntaxe du langage est commme nous allons le voir très proche du langage SQL. C'est pour cette raison que cette solution est souvent utilisé pour résoudre des problématiques de performances en lieu et place d'une base de donnée SQL. Elle permet dans sont usage d'améliorer les performances pour un coût de développement réduit lorsque le périmètre fonctionnel le permet.

Pour lancer la console cassandra, on exécute la commande suivante dans le terminal:

```
> cqlsh
```

Nous allons ensuite devoir créer un keyspace. Un keyspace est l'équivalent d'une base de donnée dans le jargon SQL. On remarque que le paramètre replication est obligatoire:

```
create keyspace mykeyspace with replication = {'class':'SimpleStrategy','replication_factor' : 2};
```

Ensuite, on se positionne sur notre keyspace:

```
use mykeyspace;
```

Tout comme pour une base SQL, nous devons créer une table en précisant le nom et le type de chaque colonne:

```
create table usertable (userid int primary key, usergivenname varchar, userfamilyname varchar,
userprofession varchar);
```

On peut ensuite commencer à insérer des données:

```
insert into usertable (userid, usergivenname, userfamilyname, userprofession) values (1, 'Guillois', 'Loïc',
'Formateur');
```

Créez plusieurs utilisateurs en prenant soin de varier les nom, prénom et profession.

On peut lister les données d'une table:

```
select * from usertable;
```

Pour mettre à jour les données, la syntaxe est là encore équivalente à SQL:

```
update usertable set userprofession = 'IT Consultant' where userid = 1;
```

La lecture des données est là encore très proche de la syntaxe SQL:

```
select * from usertable where userid = 1;
```

On peut aussi filtrer les données selon le contenu des colonnes:

```
select * from usertable where userprofession = 'IT Consultant';
```

Attention: les index sont obligatoires contrairement aux bases SQL. C'est une contrainte technique qui amène aussi à une bonne pratique. Vous renconterez donc l'erreur suivante tant que vous n'aurez pas créer d'index:

```
InvalidRequest: Error from server: code=2200 [Invalid query] message="Cannot execute this query as it might involve data filtering and thus may have unpredictable performance. If you want to execute this query despite the performance unpredictability, use ALLOW FILTERING"
```

Pour créer un index, on utilise la commande suivante. Il vous faudra créer des index qui couvrent toutes vos requêtes sans quoi vous aurez de nouveau l'erreur.

```
create index idx_dept on usertable(userprofession);
```

On peut à nouveau effectuer notre requête:

```
select * from usertable where userprofession = 'IT Consultant';
```