MongoDB Précis et concis

Table des matières

Installation / Cloud

Installation locale sur Windows

2 possibilités : avec installeur **MSI** ou **archive**.

Avec archive

- **Télécharger l'archive** de la dernière version Windows https://www.mongodb.com/download-center/community/releases ou https://www.mongodb.com/try/download/community en sélectionnant « ZIP ». Il est recommandé d'utiliser une version 64 bits en production.
- **Dézipper l'archive** à la racine, on peut également conserver le nom de l'archive. Exemple « C:\mongodb-win32-x86_64-windows-5.0.0 »
- Création d'un dossier « data » à la racine du lecteur C:/ par exemple. Dans ce dossier on crée :
    - Un dossier « db » qui contiendra les databases générées par MongoDB

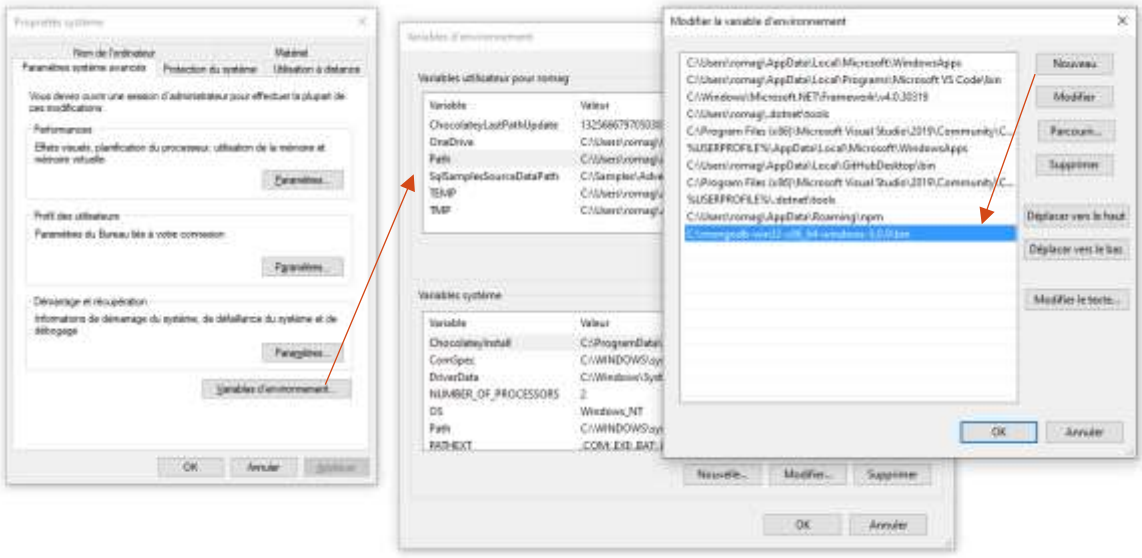
S'assurer d'avoir les permissions d'écriture sur le dossier, sinon

```
icacls C:\data /grant romag:(OI)(CI)F
```

    - Un dossier « config » qui contiendra le fichier de configuration que l'on va définir soi-même « mongod.cfg »
    - Un dossier « log » qui contiendra le fichier de logs « mongod.log » généré par MongoDB
- Créer le **fichier de configuration** « **mongod.cfg** » dans le dossier « data/config » (format Yaml, que des espaces/pas de tabs)

```
# mongod.conf

# for documentation of all options, see:
#   http://docs.mongodb.org/manual/reference/configuration-options/


# where to write logging data.
systemLog:
   destination: file
   logAppend: true
   path:  C:\data\log\mongod.log

# Where and how to store data.
storage:
   dbPath: C:\data\db

# network interfaces
net:
   bindIp: 127.0.0.1
   port: 27017
```

- Ajout à la **variable d'environnement PATH** de Windows vers le répertoire « bin » contenant de MongoDB pour pouvoir saisir « mongod » directement dans la console



- **Installation du service**

```
mongod --install --config C:\data\config\mongod.cfg
```

Ou pour un utilisateur

```
mongod --install --config C:\data\config\mongod.cfg --serviceUser romag --servicePassword Secret12345
```

Pour désinstaller le service

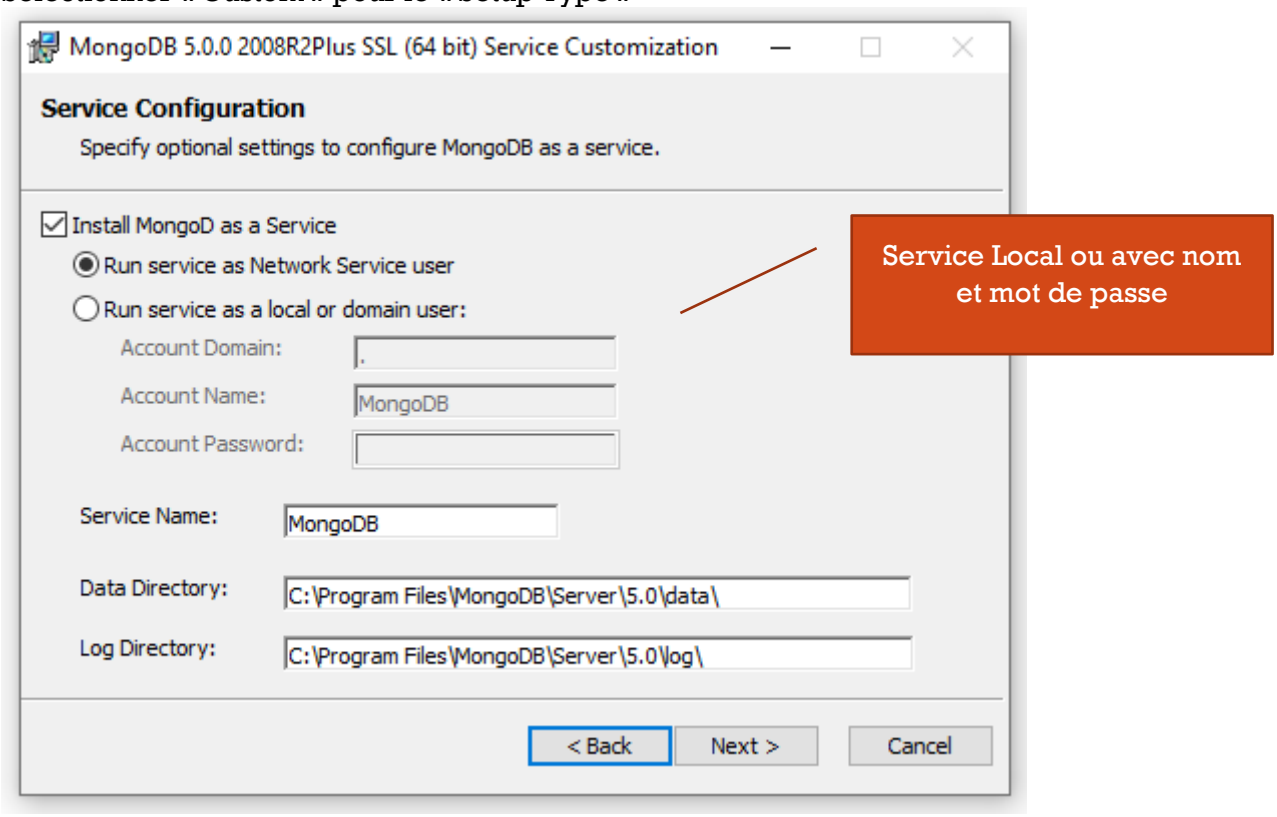
```
mongod --remove
```

Avec installeur MSI

Permet d'installer rapidement MongoDB en tant que service.

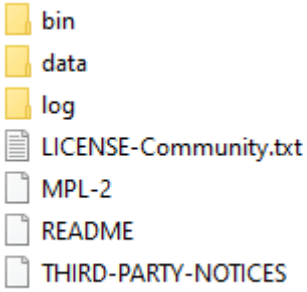
Version community (gratuit) https://www.mongodb.com/try/download/community en sélectionnant « msi ». Il y a d'autres version (entreprise, etc.) mais payantes.

Sélectionner « Custom » pour le « Setup Type »



À savoir que le service a un démarrage « Automatique » et est lancé.

Les dossiers « bin », « data » et « log » se trouvent dans le répertoire d'installation



Le **fichier de configuration** « **mongod.cfg** » se trouve dans le dossier « bin »

```
# mongod.conf

# for documentation of all options, see:
#   http://docs.mongodb.org/manual/reference/configuration-options/
```

```
# Where and how to store data.
storage:
  dbPath: C:\Program Files\MongoDB\Server\5.0\data
  journal:
    enabled: true
#  engine:
#  wiredTiger:

# where to write logging data.
systemLog:
  destination: file
  logAppend: true
  path:  C:\Program Files\MongoDB\Server\5.0\log\mongod.log

# network interfaces
net:
  port: 27017
  bindIp: 127.0.0.1


#processManagement:

#security:

#operationProfiling:

#replication:

#sharding:

## Enterprise-Only Options:

#auditLog:

#snmp:
```

Démarrer le service

Si le service n'est pas démarré

```
net start mongodb
```

Ou directement depuis l'interface de « Services » de Windows.

Pour arrêter le service, il suffira de faire

```
net stop mongodb
```

Pour se connecter avec l'interface Shell

Depuis une invite de commande en mode « administrateur »

```
mongo
```

On peut désormais accéder aux différentes databases en ligne de commande.

Pour quitter l'interface de commandes de Mongodb

```
exit
```

ReplicaSet

La réplication c'est avoir un database et des des databases avec des copies des données permettant de gérer les situations critiques par exemple. On a une database « primary » ou principale.

Au plus simple

Création d'une clé « rs1.key » avec « openssl »

Créer un dossier « key » qui contiendra la key dans le dossier « data ».

Télécharger l'archive puis depuis une invite de commande en mode administrateur, naviguer jusqu'au dossier « bin » contenant l'utilitaire (« cd … »)

```
openssl rand -base64 756> C:/data/key/rs1.key
```

Modifier le fichier configuration.

```
# mongod.conf

# for documentation of all options, see:
#   http://docs.mongodb.org/manual/reference/configuration-options/


# where to write logging data.
systemLog:
   destination: file
   logAppend: true
   path:  C:\data\log\mongod.log

# Where and how to store data.
storage:
   dbPath: C:\data\db

# network interfaces
net:
   bindIp: 127.0.0.1
   port: 27017

#security
security:
   keyFile: C:\data\key\rs1.key
   authorization: enabled

#replication
replication:
   oplogSizeMB: 10240
   replSetName: "rs1"
   enableMajorityReadConcern: true
```

Ajouter le chemin vers la key

Ajouter la section « replication » avec le nom du replicaSet « rs1 » par exemple

Désinstaller et réinstaller le service

```
mongod --remove
mongod --install --config C:\data\config\mongod.cfg
```

Démarrer le service MongoDB

Depuis le Shell (en mode administrateur), se connecter, créer un utilisateur avec tous les droits

```
mongo
rs.initiate()
```

Presser entrée une seconde fois pour être sur « PRIMARY »

```
> rs.initiate()
{
        "info2" : "no configuration specified. Using a default configuration for the set",
        "me" : "127.0.0.1:27017",
        "ok" : 1
}
rs1:SECONDARY>
rs1:PRIMARY>
```

Création d'un user/ admin qui a vraiment tous les droits (sinon on peut avoir des errors « unauthorized »)

```
use admin
db.createUser({user:"jerome",pwd:"secret",roles:[{role:"root",db:"admin"}]})
```

Et connexion au Shell avec ce user

```
mongo -u jerome -p secret
```

Méthodes de gestion utiles

Pour afficher la config

```
rs.config()
```

Afficher le status

```
rs.status()
```

Pour reconfigure le replicaSet (documentation)

```
rs.reconfig(cfg)
```

Si besoin d'indiquer le replicaSet dans la chaine de connexion à MongoDB (avec un projet Node.js par ex)

```
const url = "mongodb://jerome:secret@127.0.0.1:27017/sample?authSource=admin&retryWrites=false&replicaSet=rs1";
```

Mais ce n'est pas forcément nécessaire

```
const url = "mongodb://jerome:secret@127.0.0.1:27017/sample?authSource=admin";
```

Note: les transactions (avec mongoose) ne sont utilisables qu'avec un replicaSet

Avec 3 clusters

Exemple : On crée 3 dossiers « data\temp\m1 », « data\temp\m2 », « data\temp\m3 » qui contiendront databases répliquées

On lance les servers sur un port différent (pas la peine de lancer le service MongoDB)

```
start mongod --port 30001 --dbpath C:\data\temp\m1 --replSet rs1
start mongod --port 30002 --dbpath C:\data\temp\m2 --replSet rs1
```

```
start mongod --port 30003 --dbpath C:\data\temp\m3 --replSet rs1
```
Puis on se connecte au Shell en indiquant le port

```
mongo --port 30001
rs.initiate()
```
Presser jusqu'à être sur « PRIMARY » puis ajout

```
rs.add("127.0.0.1:30002")
rs.add("127.0.0.1:30003")
```

Note pour ajouter en tant qu'arbiter « rs.add("127.0.0.1:30003",true) »

On peut voir les membres avec « rs.status() »

On peut **insérer des données** dans la database test par exemple. Puis **vérifier que les données ont bien été répliquées** sur les autres clusters.

```
use test
db.messages.insertOne({message:"message 1"})
```
Connexion au SECONDARY depuis le Shell MongoDB

```
db = connect("127.0.0.1:30002")
db.setSecondaryOk() // ou db.setSlaveOk()
```
Puis on interroge simple pour voir si les données ont été ajoutées

```
db.messages.find()
```

Pour stopper les servers

Se connecter sur chaque port « mongo --port 30001 » « mongo --port 30002 » « mongo --port 30003 » et

```
use admin
db.shutdownServer()
```

MongoDB Atlas

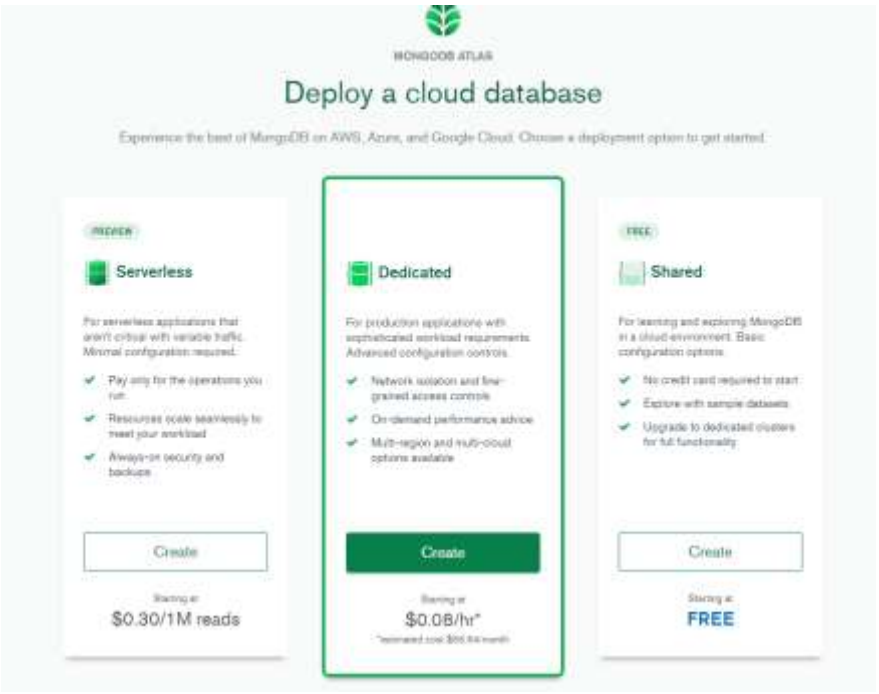
Managed MongoDB Hosting | Database-as-a-Service | MongoDB

Se connecter ou créer un compte / ou cliquer sur start « free »

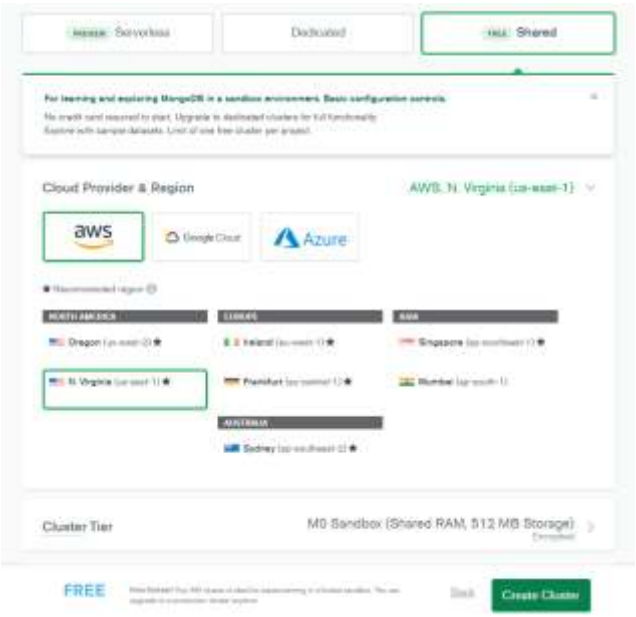
Créer un nouveau projet

Objectif : créer une base, donner l'accès à cette base selon l'IP, créer un utilisateur / admin pour sécuriser la base de données

1. Cliquer sur le bouton « New project » … donner un nom au projet (exemple « mongo-server ») puis « Create project ».
2. Onglet « Database » … « Create a database » … « Shared » pour free

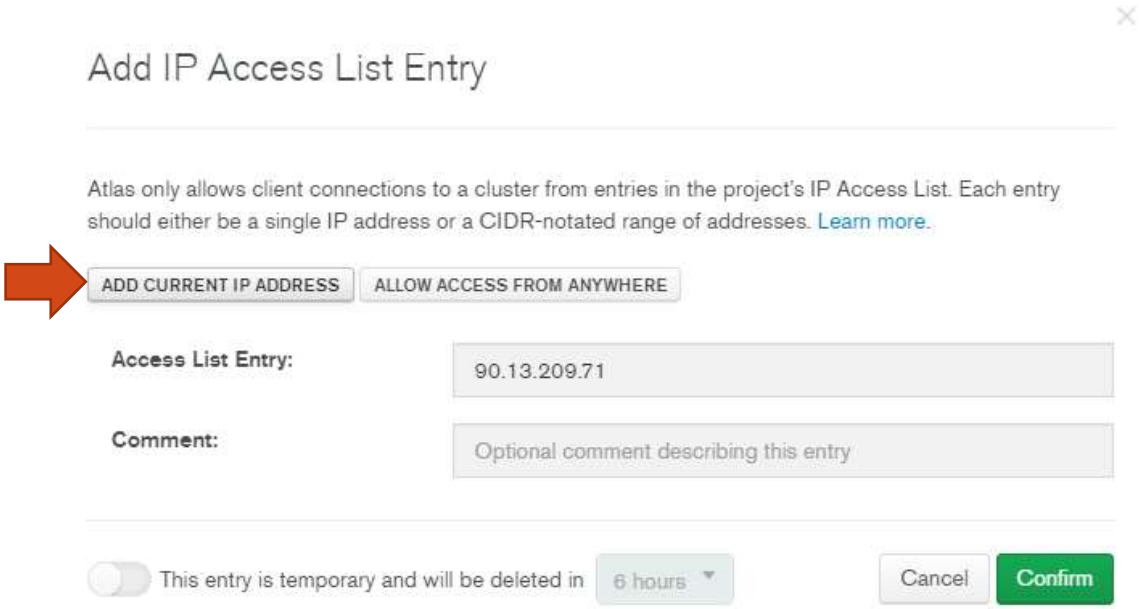


Choix du provider (AWS, Google Cloud, Azure), region …. Ou directement « creae cluster »

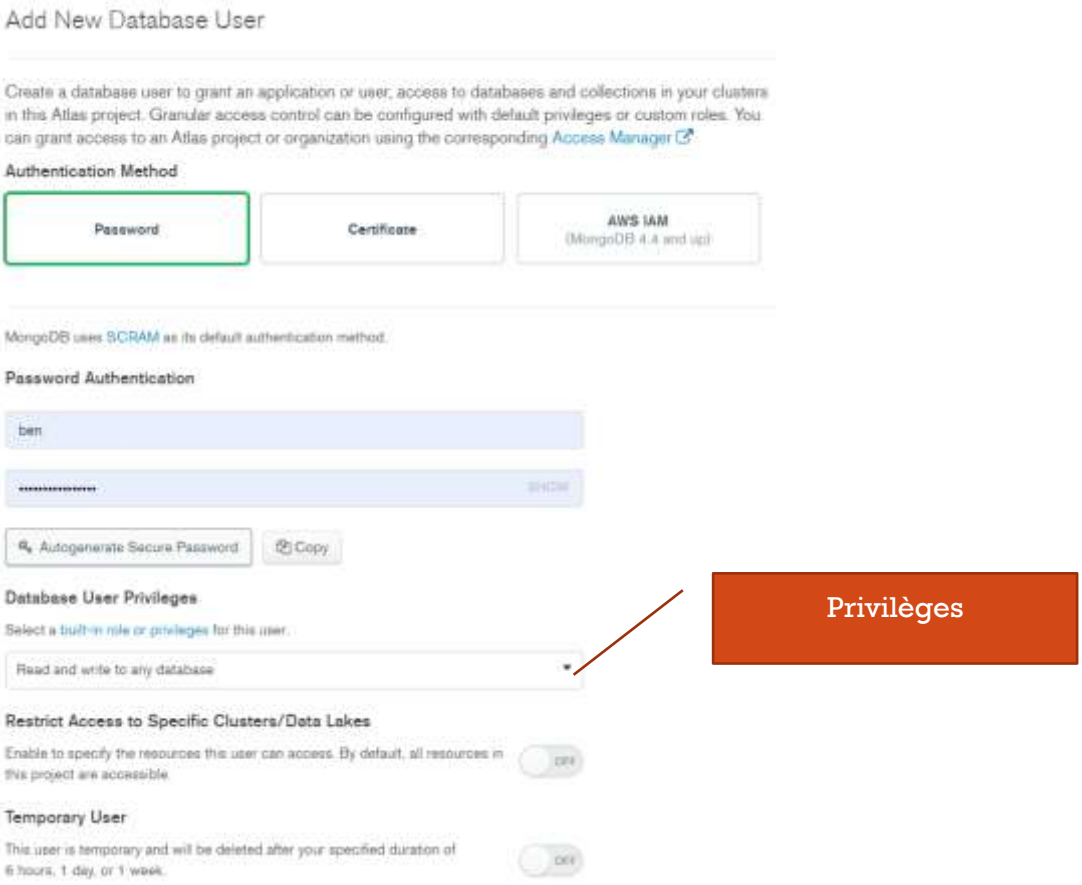


3. Network Access : IP autorisées

Onglet « Network Access » … Add IP Address … Sélectionner IP courante ou autoriser accès depuis partout



4. Database Access : création d'un utilisateur « admin »

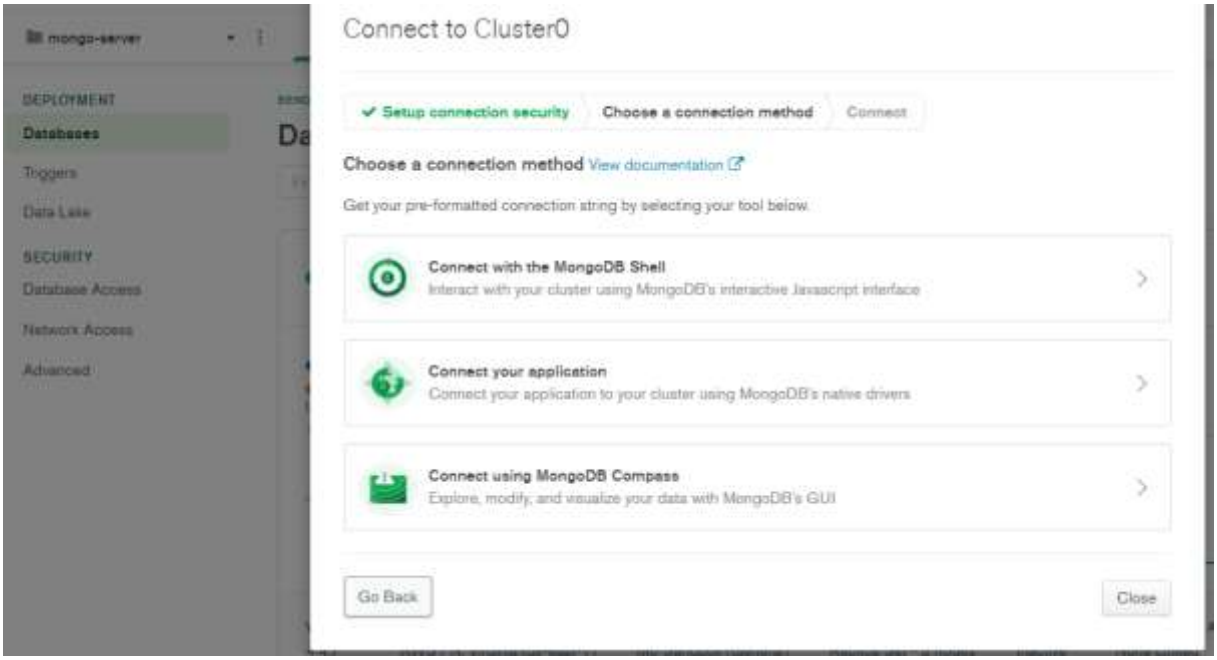


Se connecter à cette base de données

Onglet « Databases » cliquer sur le bouton « connect » de la base désirée. Démarches pour:

- Mongosh

- Son application : choix du driver (Node.js, C#, PHP, etc.)
- MongoDBCompass



L'important est de copier la **chaine de connexion**. Exemple

```
mongodb+srv://ben:<password>@cluster0.l5hoc.mongodb.net/myFirstDatabase?retryWrites=true&w=majority
```

Il faut ensuite remplacer le password (copier celui défini pour l'utilisateur dans « Database Access ») et remplacer le nom de la database « myFirstDatabase »

Deployment

- Atlas (free)
- ~~mLab~~ : partie de atlas
- Clever cloud
- Plus : Compose, Scalegrid, Digital Ocean, Mongo Clusters, A2 Hosting, NodeChef

Sécurité

Création d'un utilisateur avec le rôle admin

https://docs.mongodb.com/manual/tutorial/enable-authentication/

Note : on peut modifier le fichier de configuration (ajout de *authorization enabled*) puis se connecter avec la commande « mongo » (sans identifiants) s'il n'y a pas d'utilisateur créé … le temps d'en ajouter un…. puis redémarrer le service.

Utiliser la database « admin »

```
use admin
```

… puis création d'un utilisateur avec privilèges

```
db.createUser({user:"jerome",pwd:"secret",roles:[{role:"userAdminAnyDatabase",db:"admin"},"readWriteAnyDatabase"]})
```

On ajoute ce rôle pour pouvoir lire/ écrire sur les différentes databases

Une collection users a été créée

```
> show collections
system.users
system.version
```

Espace de nom « system »

Redémarrer le service

Depuis le Shell (mongosh) on peut entrer la commande

```
db.adminCommand( { shutdown: 1 } )
```

Puis pour quitter le Shell

```
exit
```

Editer le fichier de configuration « mongod.cfg ». Partie **Security**

```
#security
security:
   authorization: enabled
```

Redémarrer le service

```
net start mongodb
```

Si on essaie de se connecter avec « mongo » sans être authentifié on ne verra aucune database ni collection.

Se connecter avec cet utilisateur pour de nouveau avoir accès aux databases.

```
mongo -u jerome -p secret
```

Raccourci de

```
mongo --username jerome --password secret
```

User créé avec le scope d'une database autre que « admin »

On peut créer un user avec un « scope » pour une db différente de « admin », celui-ci sera ajouté quand même dans « system.users » (commande « show users »

Exemple : on utilise une database nommée « blog » et on crée un user

```
use blog
db.createUser({user:"marie",pwd:"secret",roles:["readWrite"]})
```

Pour se connecter avec ce user, il faut spécifier la database ou le user est défini

```
mongo -u marie -p secret --authenticationDatabase blog
```

Rôles principaux

https://docs.mongodb.com/manual/reference/built-in-roles/

https://docs.mongodb.com/manual/reference/built-in-roles/#all-database-roles

- **userAdminAnyDatabase** : permet de créer des users, grant permissions (même à lui-même), *mais pas de lire les données des collections/ ni de créer de nouvelles collections/doctuments*
- **read** : read collections
- **readWrite** : tout ce que peut faire « read » + write non-system collections
- **root** : tous les rôles

Méthodes de gestion des users utiles

https://docs.mongodb.com/manual/reference/method/js-user-management/

Afficher les users créés

En étant connecté avec un utilisateur ayant les permissions

```
show users
```

db.auth(), db.logout()

Permet de se connecter/déconnecter avec un user depuis le shell. Exemple

```
mongo
use admin
db.auth("jerome","secret")
db.logout()
```

Souvent il vaudra mieux quitter le shell (commande « exit ») et faire

```
mongo -u jerome -p secret
```

grantRolesToUser

Exemple : on crée un utilisateur admin qui n'a pas les droits de lecture/écriture sur la databases

```
use admin
db.createUser({user:"ben",pwd:"secret",roles:[{role:"userAdminAnyDatabase",db:"admin"}]})
```

En se connectant avec cet utilisateur on a donc des erreurs « unauthorized » dès qu'on essaie de lire une collection ». Comme cet utilisateur a le rôle « userAdminAnyDatabase » il peut lui-même s'attribuer la permission de lecture sur une database

```
use admin
db.grantRolesToUser("ben",[{"db":"blog",role:"read"}])
```

revokeRolesFromUser

Permet de supprimer un role pour un utilisateur

```
db.revokeRolesFromUser("ben",[{"db":"blog",role:"read"}])
```

dropUser

Permet de supprimer un utilisateur

```
db.dropUser("ben")
```

Note il existe une méthode permettant de supprimer tous les users :
« db.dropAllUsers() »

Interagir avec MongoDB

Shell

Pour se connecter au Shell

```
mongo
```

Ou pour s'authentifier avec un utilisateur

```
mongo -u jerome -p secret
```

Pour quitter le shell

```
exit
```

Mongosh

Mongo est deprecated, il est recommandé d'utiliser Mongosh désormais. Il propose une interface plus claire pour les différentes actions.

Télécharger le msi https://www.mongodb.com/try/download/shell

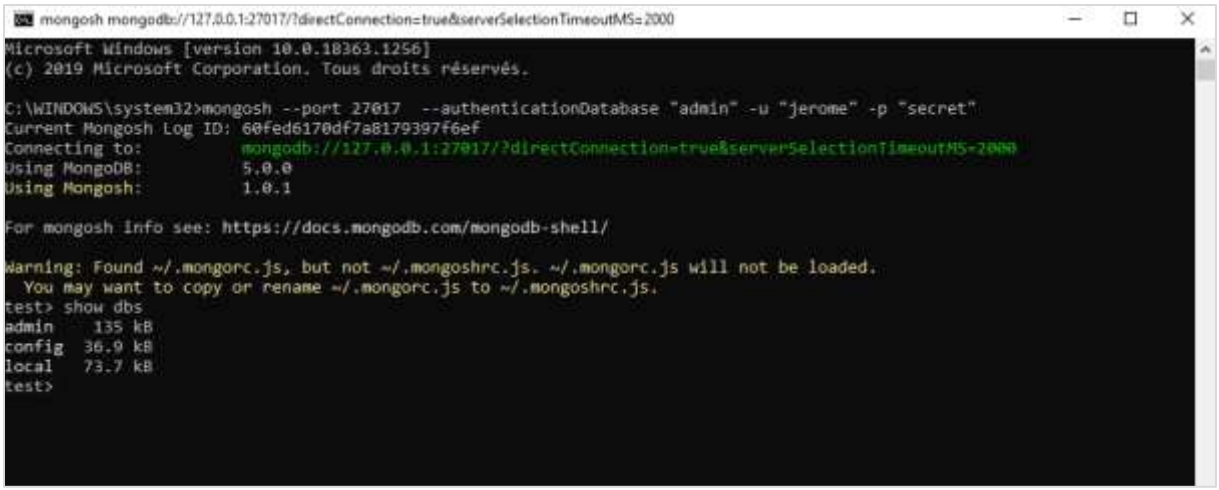
« C:\Users\<username>\AppData\Local\Programs\mongosh\ »

Le dossier a été ajouté dans la variable d'environnement PATH durant l'installation. On peut donc directement accéder à la commande « mongosh » depuis une invite de commandes (en mode administrateur)

```
mongosh
```

Ou pour s'authentifier avec un utilisateur

```
mongosh --port 27017  --authenticationDatabase "admin" -u "jerome" -p "secret"
```



Pour quitter mongosh

```
exit
```

Avec une Tool

MongoDBCompass

Peut être installé directement avec le « msi » de MongoDB. On peut aussi le télécharger https://www.mongodb.com/try/download/compass

Se connecter à une database

Locale

```
mongodb://127.0.0.1:27017/test
```

Locale avec utilisateur + mot de passe

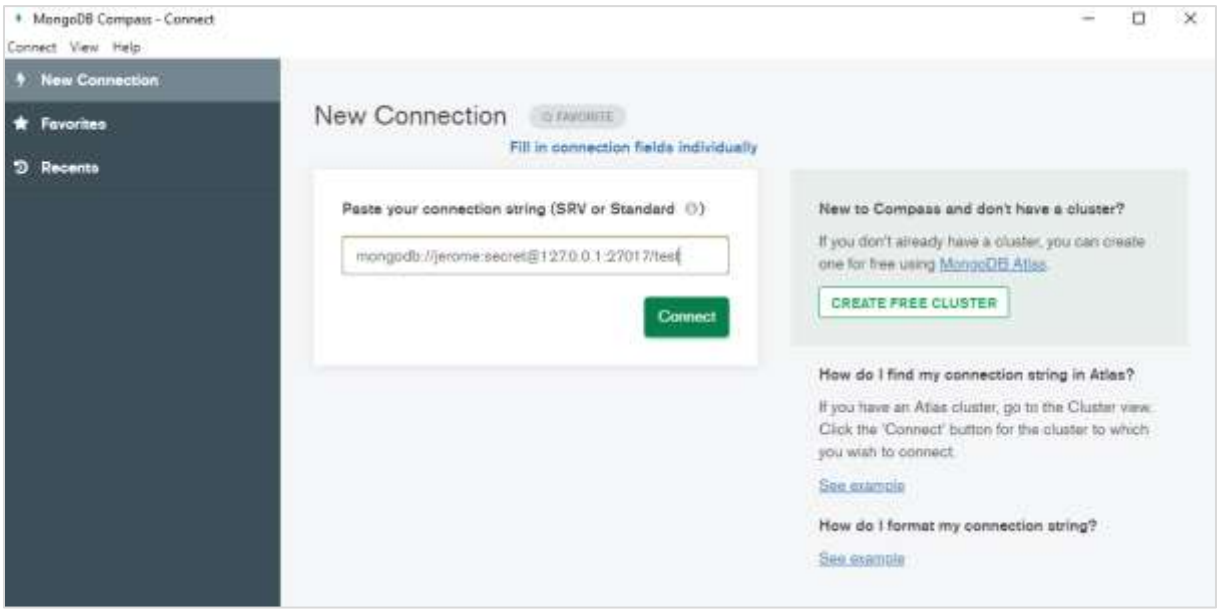
```
mongodb://jerome:secret@127.0.0.1:27017/test
```

Il faudra peut être indiquer la db (avec mongoose par ex)

```
mongodb://jerome:secret@127.0.0.1:27017/test?authSource=admin
```

Distante avec Atlas par exemple

```
mongodb+srv://ben: iAC9np8LLHBQmMpO@cluster0.pruby.mongodb.net/test
```



Comprendre l'url

```
mongodb+srv://ben: iAC9np8LLHBQmMpO@cluster0.pruby.mongodb.net/test
```

Nom d'utilisateur

Mot de passe de l'utilisateur

Database (obligatoire)

Robot 3T

Anciennement Robomongo https://robomongo.org/

Avec son application

Un driver va faire l'intermédiaire entre le langage de programmation et l'accès à la base de données MongoDB.

De nombreux drivers disponibles (Node.js, C#, PHP, etc.) :
https://docs.mongodb.com/drivers/

La documentation donne également les informations et des exemples de code pour savoir comment se connecter.

Par exemple pour Node.js le driver est « mongodb » qu'il faut installer avec NPM dans son projet

```
npm install mongodb
```

Mais dans le cas de Node.js on préfèrera sans doute utiliser mongoose, une librairie pour accéder plus facilement à MongoDB. Le driver « mongodb » est installé en même temps en tant que dépendance.

Comprendre MongoDB

Database → collection (de documents) → document

Exemple

- blog
  - posts
    - title : « post 1 », content : « content 1 »
  - categories
    - name : « categorie 1 »

Database

Collections « posts » et « categories »

Document avec des fileds et values

Management de database avec le Shell

Commandes utiles

Pour voir quelle est la database en cours

```
db
```

Pour afficher les database

```
show dbs
```

Pour utiliser une database (la crée si elle n'existe pas au premier ajout de collection)

```
use products
```

Pour afficher les collections de la database

```
show collections
```

Supprimer un database

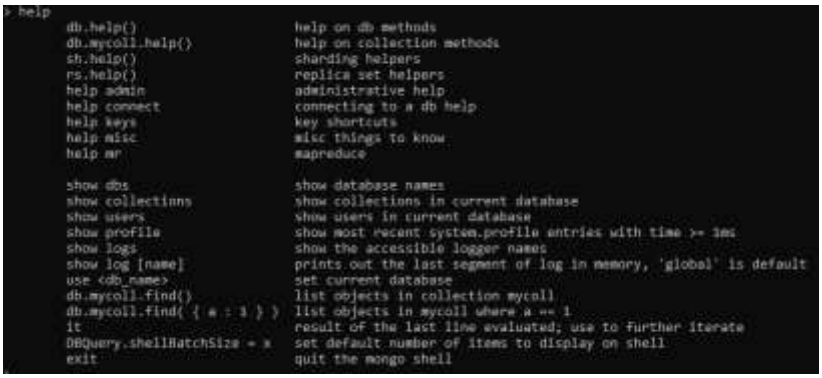
```
use blog
db.dropDatabase()
```

Supprimer une collection

```
use blog
db.posts.drop()
```

Aide

Afficher les commandes avec « help »



Afficher les stats d'une database

```
blog> db.stats()
{
  db: 'blog',
  collections: 1,
  views: 0,
  objects: 1,
  avgObjSize: 66,
  dataSize: 66,
  storageSize: 4096,
  freeStorageSize: 0,
  indexes: 1,
  indexSize: 4096,
  indexFreeStorageSize: 0,
  totalSize: 8192,
  totalFreeStorageSize: 0,
  scaleFactor: 1,
  fsUsedSize: 214858899456,
  fsTotalSize: 488360828928,
  ok: 1
}
```

Query

En général on se positionne sur la database à utiliser (créée au premier ajout si elle n'existe pas). Exemple

```
use blog
```

JSON vs BSON

https://docs.mongodb.com/manual/reference/bson-types/

On passe des informations au format JSON, celles-ci sont converties en base de données en BSON (par exemple l'ObjectId).

MongoDB offre offre des facilités et on peut omettre par exemple les guillemets pour le nom des fields.

ObjectId

https://docs.mongodb.com/manual/reference/method/ObjectId/

Création d'un _id pour un document dont l'_id n'est pas explicitement spécifié.

Cet _id généré respecte un format particulier. Il est possible également d'obtenir divers informations avec des méthodes. Exemple la date à laquelle le document a été créé

```
ObjectId("61001ad88279cc4b59a4195a").getTimestamp()
```

… Affiche par exemple

```
ISODate("2021-07-27T14:40:24Z")
```

Insert

https://docs.mongodb.com/manual/tutorial/insert-documents/

« **insertOne** » (object) pour ajouter un document


```
db.posts.insertOne({title:"Mon article", content:"Mon contenu"})
```

Un _id est généré automatiquement

On peut aussi préciser l'_**id**

```
db.posts.insertOne({_id: 1, title:"Mon article 2", content:"Mon contenu 2"})
```

« **insertMany** » (array) pour insérer plusieurs documents en même temps

```
db.posts.insertMany([{title:"Mon article 3", content:"Mon contenu 3"},{title:"Mon article 4",content:"Mon contenu 4"}])
```

On peut aussi utiliser la méthode « **insert** » qui permet d'ajouter un ou plusieurs documents selon qu'on lui passe un array ou un object en paramètre

Import de données

https://docs.mongodb.com/database-tools/installation/installation/

Requiert de télécharger MongoDB Database Tools (zip ou archive)

https://www.mongodb.com/try/download/database-tools?tck=docs_databasetools

Ajouter à la variable PATH de Windows le chemin vers le « bin » du dossier dézippé pour avoir accès aux tools depuis les invites de commandes

Depuis une invite de command en mode administrateur

```
mongoimport -u jerome -p secret --authenticationDatabase admin --db organizer --collection friends --file C:\Users\romag\Documents\contacts.seed.json --jsonArray
```

On passe les informations de connexion si besoin, puis le nom de la database, de la collection le fichier. Ici on indique jsonArray car le fichier contient des données dans un array. Exemple de fichier json.

```
[{
    "username":"marie",
    "email": "marie123@gmail.com",
    "age":20
},
{
    "username":"patrick",
    "email": "pat123@gmail.com",
    "age":30
},
{
    "username":"claude",
    "email": "claude123@gmail.com",
    "age":35
}]
```

Export

https://docs.mongodb.com/v4.2/reference/program/mongoexport/

Exemple

```
mongoexport -u jerome -p secret --authenticationDatabase admin --db organizer --collection friends --out C:\friends.json
```

Aperçu du fichier généré

```
{"_id":{"$oid":"61001ad88279cc4b59a4195a"},"username":"marie","email":"marie123@gmail.com","age":20}
{"_id":{"$oid":"61001ad88279cc4b59a4195b"},"username":"patrick","email":"pat123@gmail.com","age":30}
{"_id":{"$oid":"61001ad88279cc4b59a4195c"},"username":"claude","email":"claude123@gmail.com","age":35}
```

On peut ajouter « --jsonArray » si on désire que les données soient dans un array

```
mongoexport -u jerome -p secret --authenticationDatabase admin --db organizer --collection friends --out C:\friends.json --jsonArray
```

Read all

Obtenir **tous les documents** avec « **find** »

```
db.posts.find()
```

Note on peut formater l'affichage avec « pretty »

```
db.posts.find().pretty()
```

Filtrer selon une condition. Exemple avec la collection de friends, filtre ceux ayant 20 ans

```
use organizer
db.friends.find({age:20})
```

On peut aussi utiliser les opérateurs de comparaison. Exemple filtre les friends dont l'age est supérieur à 20 et inférieur à 40

```
db.friends.find({age:{$gt:20,$lt:40}})
```

Options

Sélectionner les fields à retourner. « 0 » pour cacher, « 1 » pour afficher

```
db.friends.find({},{_id:0,email:1})
```

Opérateurs de comparaison, logiques, etc.

https://docs.mongodb.com/manual/reference/operator/query/

Opérateurs de comparaison les plus utiles :

$eq	== égal
$gt	> Plus grand que
$gte	>= Plus grand ou égal
$in	Compris entre Ex : db.friends.find({age:{$in:[20,35]}})
$lt	< Plus petit que
$lte	<= Plus petit ou égal

Opérateurs logiques : $and, $or, $not, $nor

Sort

Exemple tri des friends selon l'âge **décroissant** avec « **-1** » (du plus agé au plus jeune)

```
db.friends.find().sort({age:-1})
```

Pour le tri en **ordre croissant** passer « **1** »

Limit

Pour ne retourner qu'un nombre maximum de documents

```
db.friends.find().limit(2)
```

Cursor

La méthode find retourne un cursor (et non pas un array come on pourrait le penser). Exemple

```
> var x = db.friends.find()
> x.next()
{
        "_id" : ObjectId("61001ad88279cc4b59a4195a"),
        "username" : "marie",
        "email" : "mariebellin123@hotmail.com",
        "age" : 20
}
> x.count()
3
> x.toArray()
[
        {
                "_id" : ObjectId("61001ad88279cc4b59a4195b"),
                "username" : "patrick",
                "email" : "pat123@gmail.com",
                "age" : 30
        },
        {
                "_id" : ObjectId("61001ad88279cc4b59a4195c"),
                "username" : "claude",
                "email" : "claude123@gmail.com",
                "age" : 35
        }
]
```

Méthodes disponibles sur les cursors :
https://docs.mongodb.com/manual/reference/method/js-cursor/

Aggregation Framework

https://docs.mongodb.com/manual/aggregation/

Pipeline qui permet de filtrer, trier, grouper etc. les données retournées

Index

Création d'index pour un gain en performance. Un index est créé automatiquement sur l'_id. Mais on peut créer soi-même des index sur d'autres fields, sur une date par exemple.

Afficher les index d'une collection avec « getIndexes ».

```
> db.friends.getIndexes()
[ { "v" : 2, "key" : { "_id" : 1 }, "name" : "_id_" } ]
```

Version de l'index

Nom donné à l'index

Création d'un index sur « age » par exemple

https://docs.mongodb.com/manual/reference/method/db.collection.createIndex/#mongodb-method-db.collection.createIndex

```
db.friends.createIndex({age:1})
```

Pour créer un index sur un **nested document**

```
db.friends.createIndex({"address.country":1})
```

Il faut créer un index sur ce qui est souvent interrogé (filtres et tris) dans des requêtes.

Read one

Avec « **findOne** »

Avec **_id**

```
db.posts.findOne({_id:1})
```

_id avec un ObjectId

```
db.friends.findOne({_id:ObjectId("61001ad88279cc4b59a4195b")})
```

Avec filtre. Retourne le premier document correspondant

```
db.friends.findOne({age:{$gt:20}})
```

Nested / embedded document

Object

```
db.friends.updateOne({_id:ObjectId("61001ad88279cc4b59a4195a")},{$set:{address:{country:"London"}}})
```

Filtre avec les fields d'un embedded document

```
db.friends.find({"address.country":"London"})
```

Array

Ajout d'un array "hobbies" à un document

```
db.friends.updateOne({_id:ObjectId("61001ad88279cc4b59a4195a")},{$set:{hobbies:["Sport","Cook"]}})
```

Accès à l'array

```
db.friends.findOne({username:"marie"}).hobbies
```

Update

https://docs.mongodb.com/manual/tutorial/update-documents/

On utilise « **updateOne** » pour être sur de ne modifier qu'un document

```
db.friends.updateOne({_id:ObjectId("61001ad88279cc4b59a4195a")},{$set:{email:"mariebellin123@hotmail.com"}})
```

Sélection du document à modifier

Utilisation de $set pour ne modifier que les valeurs des champs spécifiés

Attention si on ne précise pas « $set » cela remplace tout simplement le document. Ici ce document n'aurait plus qu'un champ email.

Delete

https://docs.mongodb.com/manual/tutorial/remove-documents/

On utilise « **deleteOne** » pour supprimer un document

```
db.friends.deleteOne({_id:ObjectId("61001ad88279cc4b59a4195a")})
```

On peut utiliser « **deleteMany** » pour supprimer tous les documents selon un filtre ou **tous les documents** avec en filtre un object vide

```
db.products.deleteMany({})
```

Relations

2 solutions possibles :

- Embedded/ nested documents
- Références : un peu comme une base de données SQL, on indique la « clé étrangère »

Selon les cas une solution ou l'autre est mieux adaptée

Plusieurs types de relations : one-to-one, one-to-many, many-to-many

Exemple par « référence » : « relation 1-1/1-n » une collection « categories » et une collection « posts » (chaque post a une et une seule catégorie dans cet exemple)

```
db.categories.insertMany([{_id:1, name:"C#"},{_id:2, name:"MongoDB"}])
db.posts.insertMany([{_id:1, title:"Article sur C#", categoryId: 1},{_id:2, title:"Article sur MongoDB", categoryId:2}])
```

Filtre pour sélectionner les posts d'une catégorie

```
db.posts.find({categoryId:1})
```

Embedded/ nested document :

- Relation 1-1 → object
- Relation 1-n → array par exemple un friends qui peut avoir plusieurs hobbies

Many to many

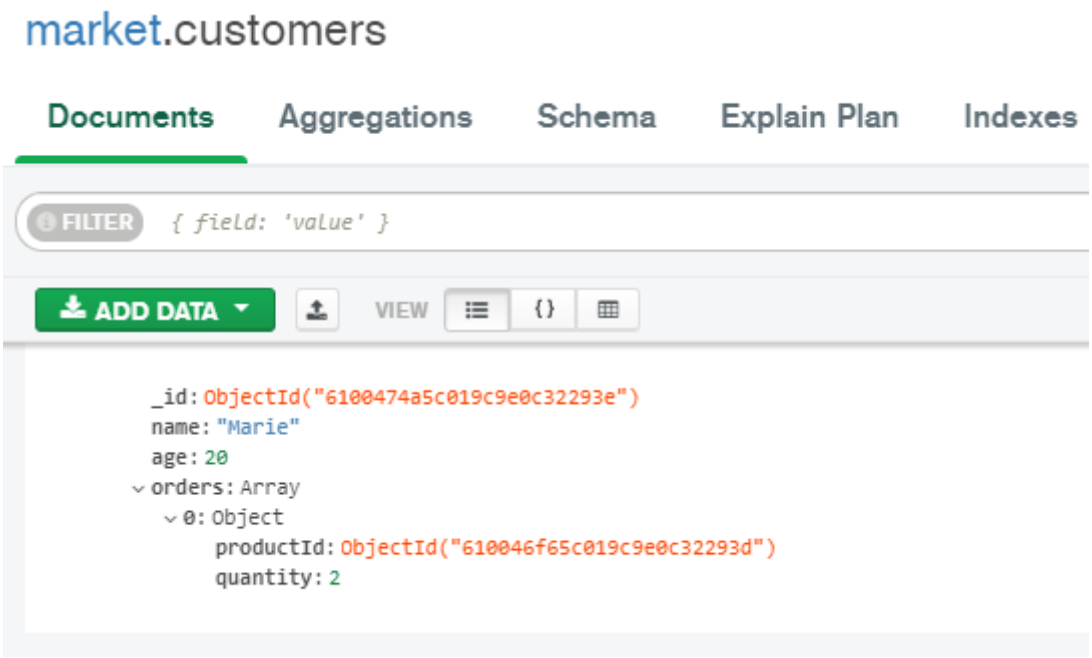
Embedded

Exemple : on inclut les orders dans le customer

```
db.products.insertOne({title:"A book", price: 10.99})
db.customers.insertOne({name:"Marie", age: 20})
db.orders.insertOne({productId:ObjectId("610046f65c019c9e0c32293d"),
customerId:ObjectId("6100474a5c019c9e0c32293e")})
```

Et on modifie le customers pour lui ajouter le tableau d'orders

```
db.customers.updateOne({},{$set:{orders:[{productId:ObjectId("610046f65c019c9e0c3
2293d"),quantity:2}]}})
```



References

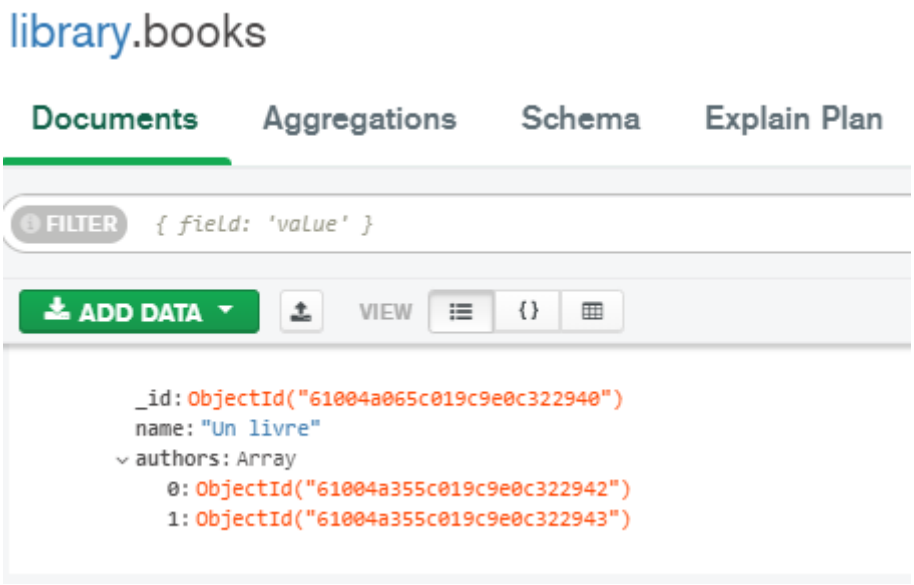
Exemple:

```
db.books.insertMany([{name:"Un livre"},{name:"Un autre livre"}])
db.authors.insertMany([{name:"Author 1"},{name:"Author 2"}])
```

On modifie books pour y inclure les authors. On n'indique que les _id

```
db.books.updateOne({_id:ObjectId("61004a065c019c9e0c322940")},{$set:{authors:[Obj
ectId("61004a355c019c9e0c322942"),ObjectId("61004a355c019c9e0c322943")]}})
```

Pareil pour les authors on peut modifier pour indiquer un array avec les _id des books qu'ils ont écrits



Schema

Mongodb est « schemaless ». Les documents ne respectent pas une structure précise de base. Un document peut par exemple avoir plus de champs ou même avoir des champs complètement différents d'un autre.

On peut quand même définir un schéma pour mieux structurer une collection

https://docs.mongodb.com/manual/core/schema-validation/

Exemple

validator

```
db.createCollection("students", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["name", "year"],
      properties: {
        name: {
          bsonType: "string",
          description: "must be a string and is required",
        },
        year: {
            bsonType: "number",
            minimum: 2017,
            maximum: 3017,
            description: "must be an integer in [ 2017, 3017 ] and is required"
         }
      }
    }
  }
});
```

```
db.students.insertOne({name:"marie",year:2020})
```