NoSQL - B3 INFRA - Équipe BAPIBO

Membres du groupe :
Axel PINTO - Eddy BOUCHOUAREB - Leo-Paul ARON - Axel BARBESIER

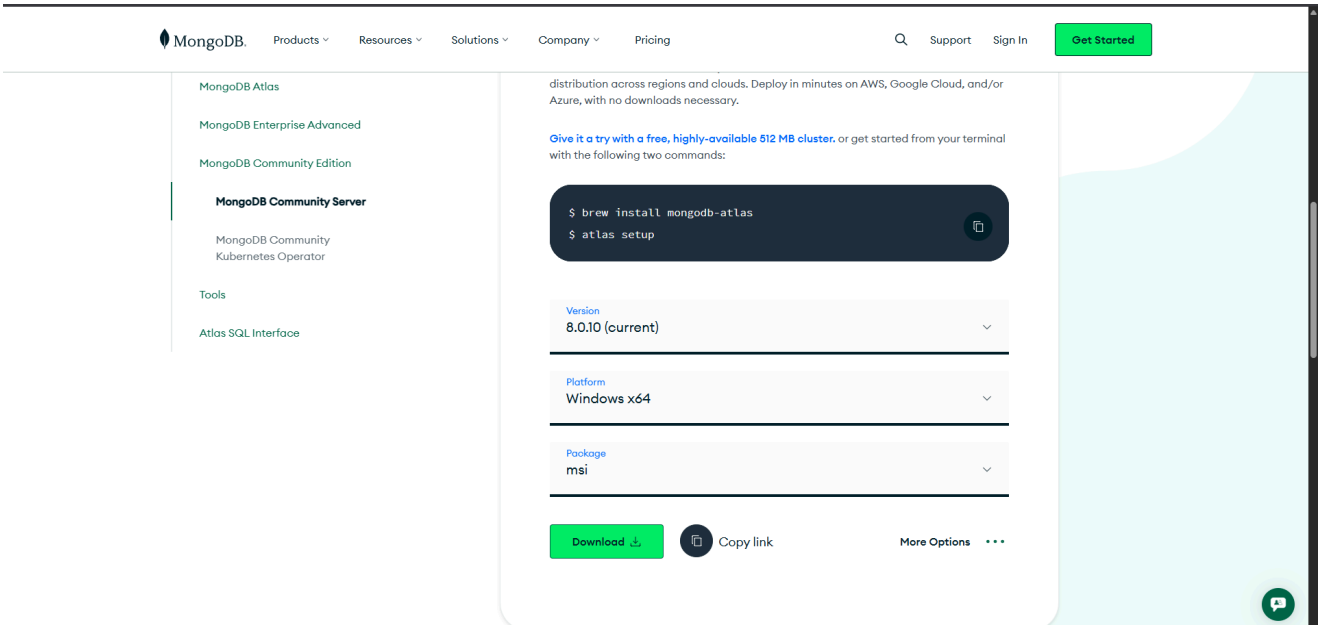
Précision :

Après avoir essayé de faire le TP avec une machine virtuelle Ubuntu 24.04, ce qui, dans un premier temps, nous prenait énormément de temps, puis après être tombé à plusieurs reprises sur l'erreur "core-dump" de MongoDB qui n'était pas compatible avec la version actuelle d'Ubuntu.

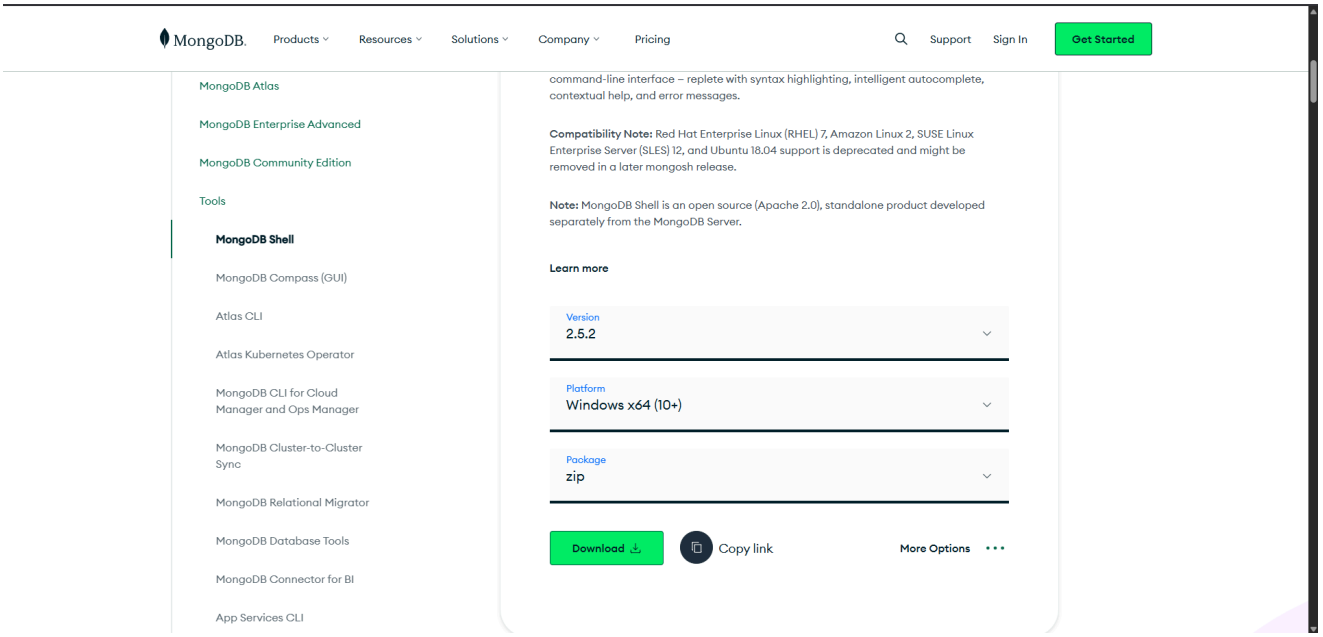
Nous avons chercher d'autre moyen de faire, et si possible, moins contraignante, nous avons trouver des tutos de personnes qui utilisés MongoDB directement via le PowerShell sous Windows et c'est ce que nous avons utilisés également.

Prérequis - Installation de MongoDB et MongoDB Shell :

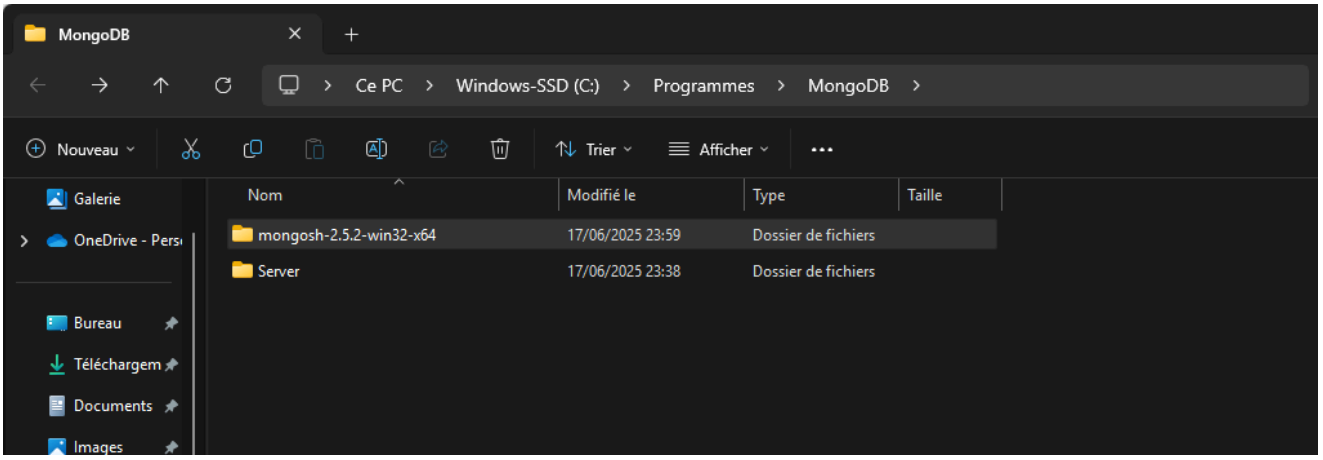
Tout d'abord on se rend sur le site officiel de MongoDB et on télécharge la dernière version du "Community server". https://www.mongodb.com/try/download/community



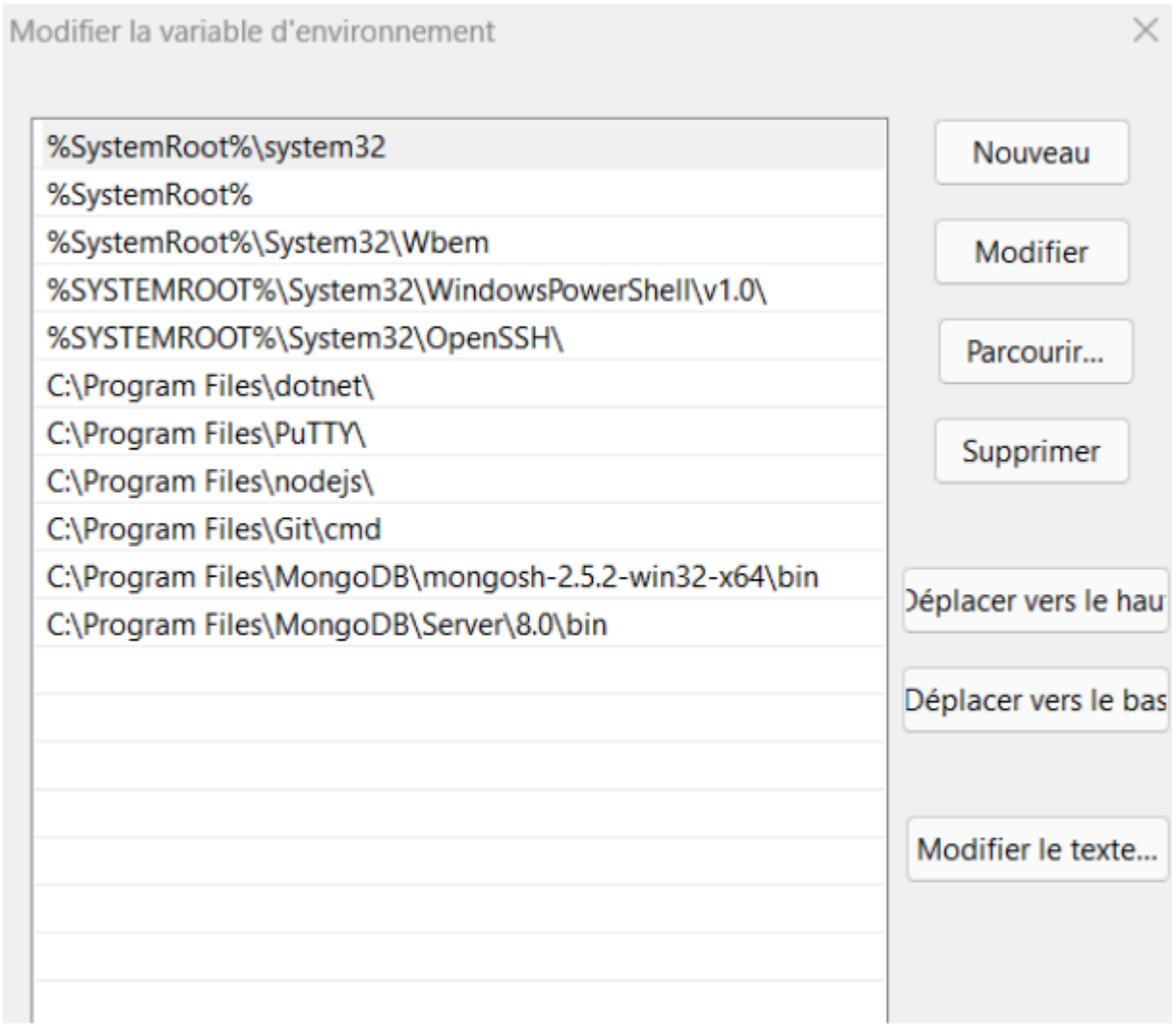
L'installation est simple, c'est seulement cliquer sur "suivant".
Puis nous téléchargeons également le Shell, qui est lui, un dossier compressé.
https://www.mongodb.com/try/download/shell



Une fois télécharger, on extrait le contenu du zip dans le dossier créer par l'installation de MongoDB Community Server.

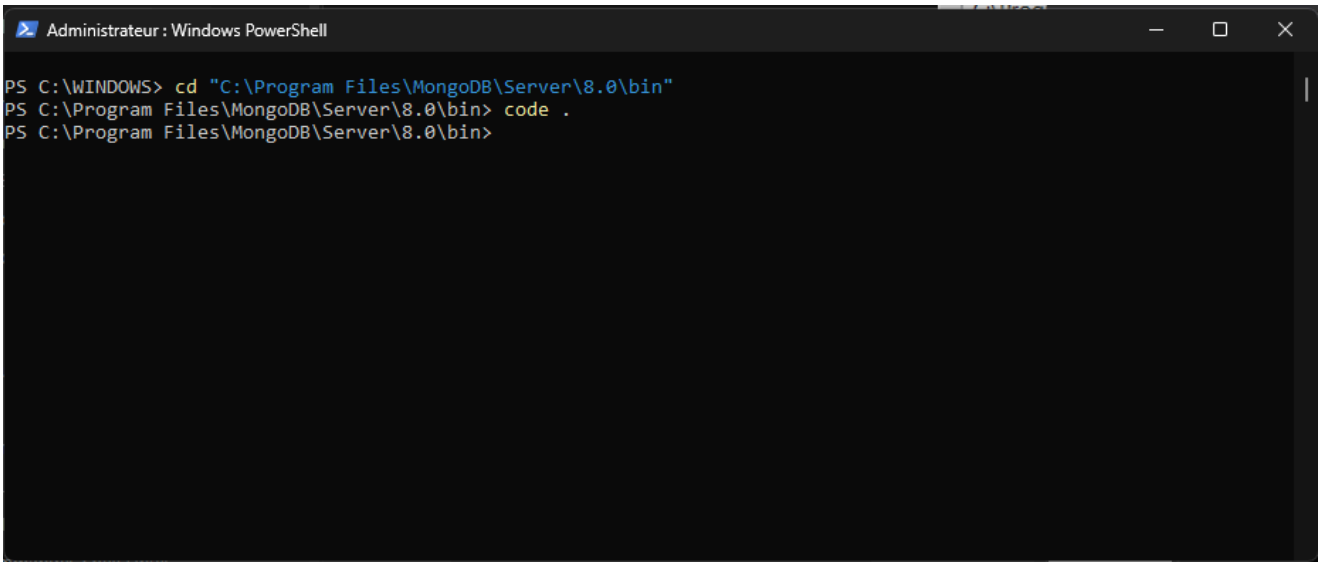


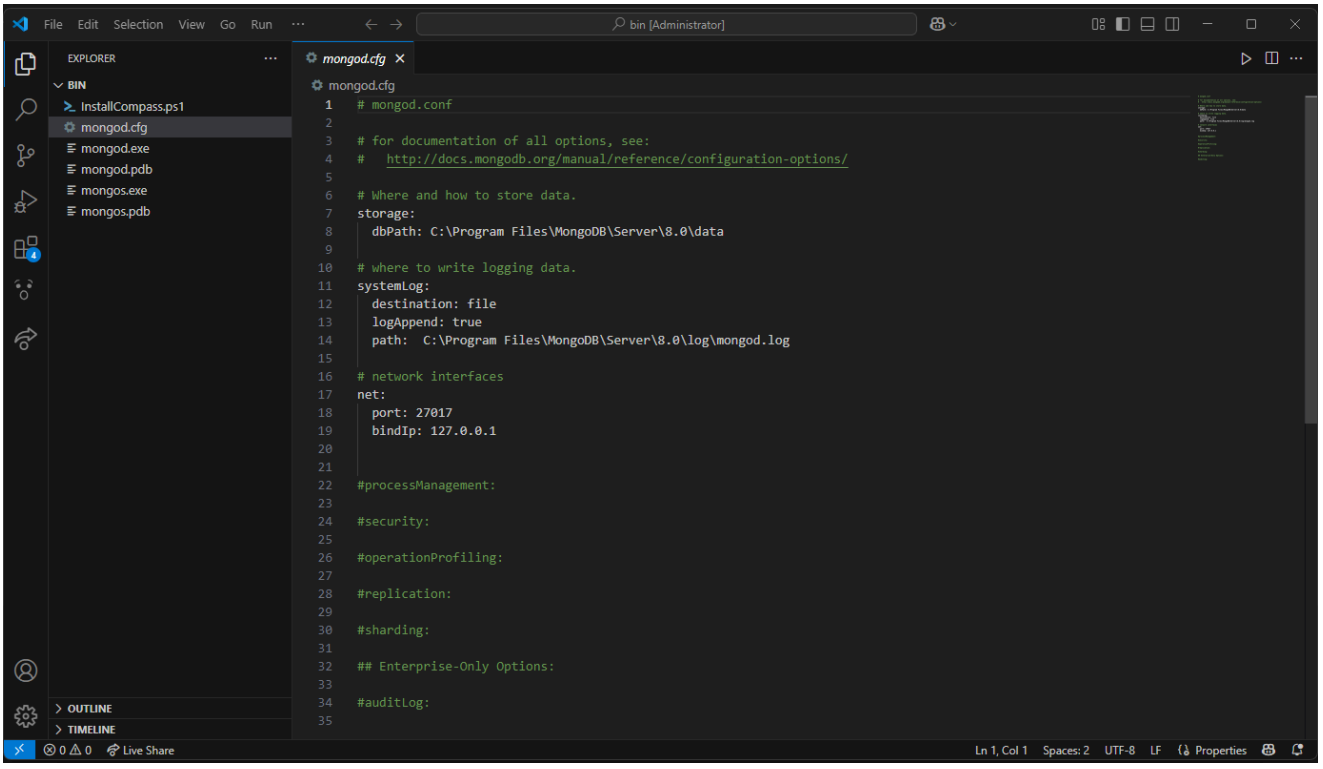
Pour finir on copie le chemin des deux dossiers dans le PATH (Variable d'environnement) afin de pouvoir exécuter les commandes ne n'importes où nous le souhaitons.



Étape 1 - Authentification via MongoDB :

Tout d'abord on ouvre un PowerShell en admin. Puis on se rends dans le dossier \bin de MongoDB
Et pour finir on tape " `code .` " qui nous ouvre notre IDE avec les fichiers de conf de MongoDB.





On enlève le commentaire sur la partie "Security" et on y ajoute l'autorisation en activé.

```
security:
  authorization: enabled
```

⚠ Perte de capture d'écran

A cause d'une mauvais manipulation (Problème de couche 8) nous avons perdu une partie des captures d'écran de l'étape 1.
Nous avons donc retaper les commandes dans la hâte histoire de les avoirs quand même noter

```
db.test.insertMany([{nom: "toto", age: 18}, {nom: "tata", age: 20}])
```

```
db.test.find()

db.test.find({age: { $gt: 24 } })
```

```
db.test.updateOne({nom: "toto"}, {$set: {age: 22}})
```

```
db.createUser({user : "admin", pwd: "123", roles: [{ roles:
"userAdminAnyDatabase"n db: "admin"}, "readWriteDatabase"]})
```

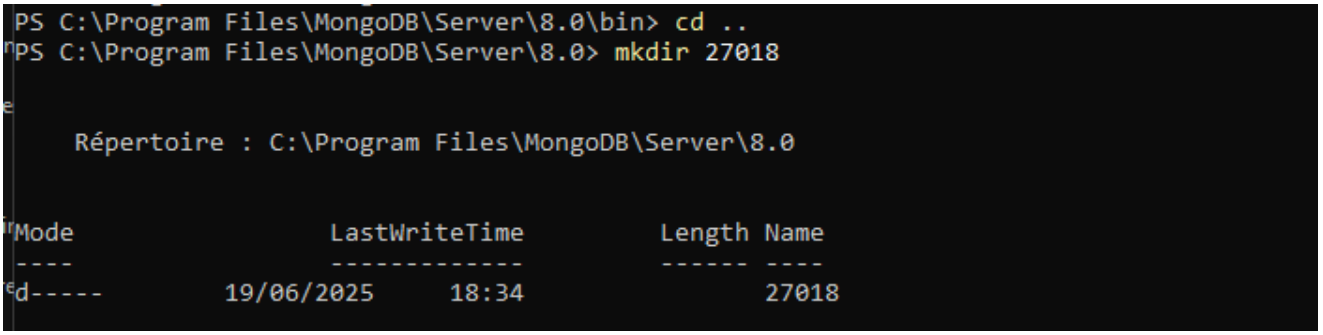
```
mongosh -u admin -p 123 --authenticationDatabase admin test
```

Étape 2 - Création des différentes instances :

> **ⓘ Enregistrement par défaut de la data**
>
> Par défaut, MongoDB stocke la data dans le chemin suivant :
> `C:\Program Files\MongoDB\Server\8.0\data`

Cependant, nous allons changer cela pour que la réplication fonctionne comme nous le souhaitons.

Dans notre invite de commande PowerShell



Nous reculons d'un dossier (avant **bin**) puis nous créons notre premier dossier du nom de **27018**
Qui sera notre **primary**

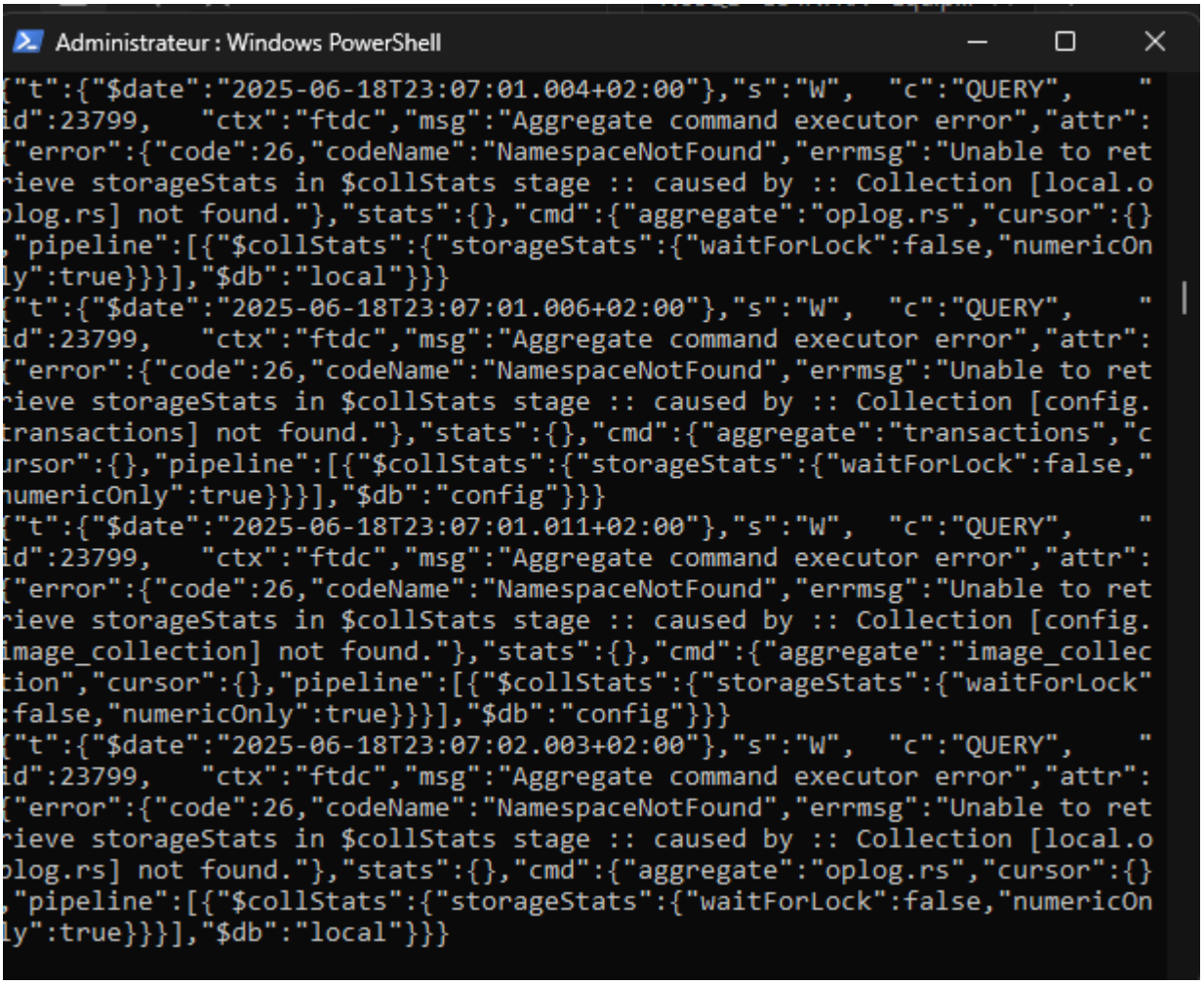
```
PS C:\Program Files\MongoDB\Server\8.0> mkdir 27019


    Répertoire : C:\Program Files\MongoDB\Server\8.0


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        19/06/2025     18:34                27019


PS C:\Program Files\MongoDB\Server\8.0> mkdir 27020


    Répertoire : C:\Program Files\MongoDB\Server\8.0


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        19/06/2025     18:34                27020


PS C:\Program Files\MongoDB\Server\8.0>
```

Puis on fait la même chose pour nos deux **secondary** qui sont **27019** et **27020**

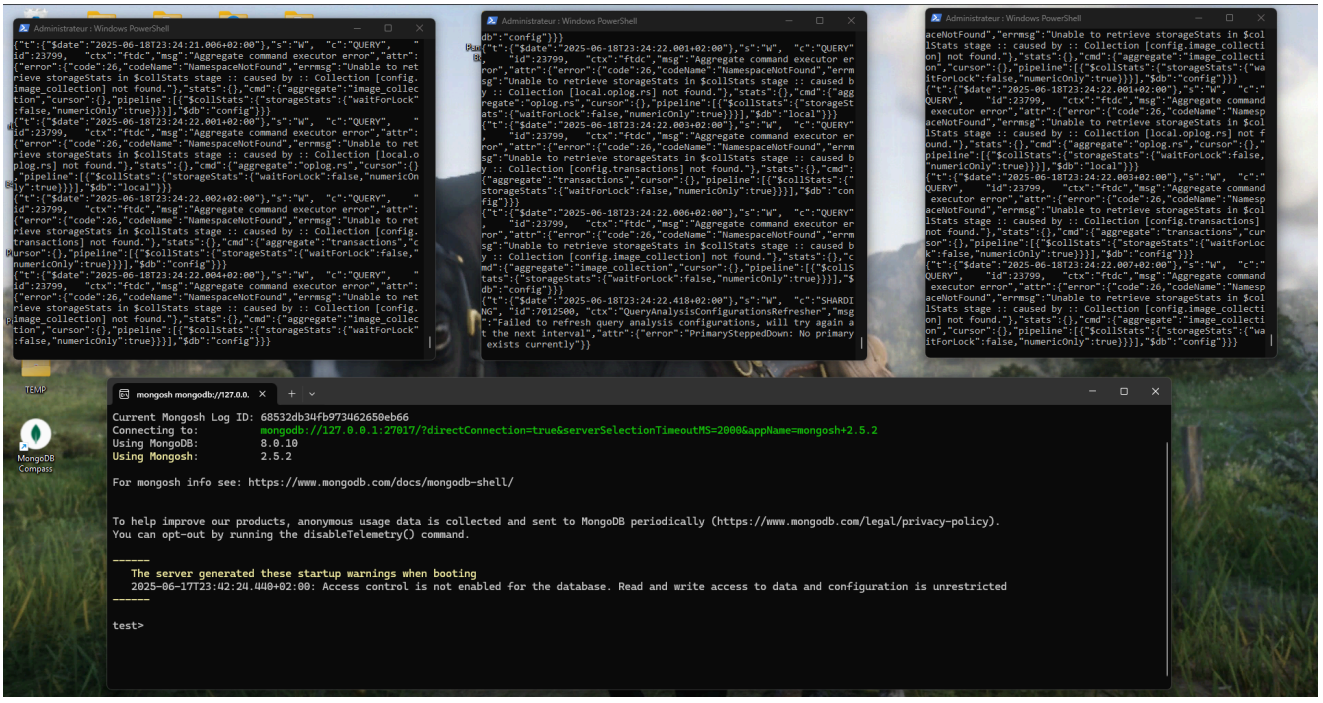
```
PS C:\Program Files\MongoDB\Server\8.0\bin> mongod --depath ../27018 --port 27018 --replSet "rs0"
```

Une fois ceci fait, nous retournons dans \bin et nous créons nos instances.
(Ici l'instance **primary 27018**)
Quelques informations concernant les arguments utilisés :

> ⓘ **Info**
>
> - **--dbpath** -> correspond au chemin où seront stockées les données, par défaut dans le \data mais dans ce TP la elles seront stockées dans les dossiers que nous avons créés au préalable.
> - **--port** -> sur quel port va écouter l'instance, et par moyen de praticité, les instances ont le même port que leurs noms de dossiers.
> - **--replSet "rs0"** -> pour préciser quel réplicatSet nous allons utiliser.



Notre instance est maintenant en train de tourner et en attente de finalisation.
Nous reproduisons donc la manipulation pour les deux autres instances.



Une fois qu'on a nos 3 instances qui tournent, on va pouvoir commencer à créer la BDD de test et à la répliquer.

```
PS C:\Program Files\MongoDB\mongosh-2.5.2-win32-x64\bin> ./mongosh --port 27018
Current Mongosh Log ID: 685466bb5b78c06e0050eb66
Connecting to:          mongodb://127.0.0.1:27018/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2
.5.2
Using MongoDB:          8.0.10
Using Mongosh:          2.5.2

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/

------
   The server generated these startup warnings when booting
   2025-06-19T21:33:30.014+02:00: Access control is not enabled for the database. Read and write access to data and conf
iguration is unrestricted
   2025-06-19T21:33:30.016+02:00: This server is bound to localhost. Remote systems will be unable to connect to this se
rver. Start the server with --bind_ip <address> to specify which IP addresses it should serve responses from, or with --
bind_ip_all to bind to all interfaces. If this behavior is desired, start the server with --bind_ip 127.0.0.1 to disable
 this warning
------

test> rs.status()
MongoServerError[NotYetInitialized]: no replset config has been received
test>
```

On peut voir ici que nous sommes bien sur notre instance dite primary sur le port 27018 et qu'aucun ReplicatSet n'a encore été fait.

```
test> rs.initiate({_id: "rs0", version: 1, members:[{_id: 0, host: "localhost:27018"}, {_id: 1, host : "localhost:27019"}, {_id:2, host: "localhost:27020"}] } )
{
  ok: 1,
```

On tape donc cette commande pour l'initialiser, avec quelques explications des arguments de la commande dans l'ordres de leur apparition :

- _ id: "rs0" : précise quel replicatSet nous souhaitons configurer
- version: 1 : la version que nous choisissons
- members: [{...}] : est le champ où nous déclarons les instances lancer plutôt
  - _ id : est l'identifiant de l'instance
  - host : est le port sur lequel est l'instance en question
- ok 1 : correspond à "ok true" qui indique que la commande est bien passé

```
members: [
  {
    _id: 0,
    name: 'localhost:27018',
    health: 1,
    state: 1,
    stateStr: 'PRIMARY',
    uptime: 1102,
    optime: { ts: Timestamp({ t: 1750362709, i: 1 }), t: Long('1') },
    optimeDate: ISODate('2025-06-19T19:51:49.000Z'),
    optimeWritten: { ts: Timestamp({ t: 1750362709, i: 1 }), t: Long('1') },
    optimeWrittenDate: ISODate('2025-06-19T19:51:49.000Z'),
    lastAppliedWallTime: ISODate('2025-06-19T19:51:49.924Z'),
    lastDurableWallTime: ISODate('2025-06-19T19:51:49.924Z'),
    lastWrittenWallTime: ISODate('2025-06-19T19:51:49.924Z'),
    syncSourceHost: '',
    syncSourceId: -1,
    infoMessage: '',
    electionTime: Timestamp({ t: 1750362269, i: 1 }),
    electionDate: ISODate('2025-06-19T19:44:29.000Z'),
    configVersion: 1,
    configTerm: 1,
    self: true,
    lastHeartbeatMessage: ''
  }
```

En retapant la commande "rs.status()" on peut maintenant avoir un peu plus d'informations, dont le fait que notre 27018 est bien le primary, comme attendu, et que les deux autres sont bien des secondary.


```
{
  _id: 1,
  name: 'localhost:27019',
  health: 1,
  state: 2,
  stateStr: 'SECONDARY',
  uptime: 452,
  optime: { ts: Timestamp({ t: 1750362699, i: 1 }), t: Long('1') },
  optimeDurable: { ts: Timestamp({ t: 1750362699, i: 1 }), t: Long('1') },
  optimeWritten: { ts: Timestamp({ t: 1750362699, i: 1 }), t: Long('1') },
  optimeDate: ISODate('2025-06-19T19:51:39.000Z'),
  optimeDurableDate: ISODate('2025-06-19T19:51:39.000Z'),
  optimeWrittenDate: ISODate('2025-06-19T19:51:39.000Z'),
  lastAppliedWallTime: ISODate('2025-06-19T19:51:49.924Z'),
  lastDurableWallTime: ISODate('2025-06-19T19:51:49.924Z'),
  lastWrittenWallTime: ISODate('2025-06-19T19:51:49.924Z'),
  lastHeartbeat: ISODate('2025-06-19T19:51:49.646Z'),
  lastHeartbeatRecv: ISODate('2025-06-19T19:51:50.566Z'),
  pingMs: Long('0'),
  lastHeartbeatMessage: '',
  syncSourceHost: 'localhost:27018',
  syncSourceId: 0,
  infoMessage: '',
  configVersion: 1,
  configTerm: 1
},
{
  _id: 2,
  name: 'localhost:27020',
  health: 1,
  state: 2,
  stateStr: 'SECONDARY',
  uptime: 452,
  optime: { ts: Timestamp({ t: 1750362699, i: 1 }), t: Long('1') },
  optimeDurable: { ts: Timestamp({ t: 1750362699, i: 1 }), t: Long('1') },
  optimeWritten: { ts: Timestamp({ t: 1750362699, i: 1 }), t: Long('1') },
  optimeDate: ISODate('2025-06-19T19:51:39.000Z'),
  optimeDurableDate: ISODate('2025-06-19T19:51:39.000Z'),
  optimeWrittenDate: ISODate('2025-06-19T19:51:39.000Z'),
  lastAppliedWallTime: ISODate('2025-06-19T19:51:49.924Z'),
  lastDurableWallTime: ISODate('2025-06-19T19:51:49.924Z'),
  lastWrittenWallTime: ISODate('2025-06-19T19:51:49.924Z'),
  lastHeartbeat: ISODate('2025-06-19T19:51:49.646Z'),
  lastHeartbeatRecv: ISODate('2025-06-19T19:51:50.597Z'),
  pingMs: Long('0'),
  lastHeartbeatMessage: '',
  syncSourceHost: 'localhost:27018',
  syncSourceId: 0,
  infoMessage: '',
  configVersion: 1,
```

Étape 3 - Création de la base de donnée test :

Maintenant que nous avons nos 3 instances lancées, nous pouvons commencer à insérer de la data pour voir si la réplication se fait bien correctement.

```
rs0 [direct: primary] test> use BAPIBO_DB
rs0 [direct: primary] BAPIBO_DB> db.person.insertOne({name: 'Leo-Paul', age: 21})
{
  acknowledged: true,
  insertedId: ObjectId('68546d3a5b78c06e0050eb67')
}
rs0 [direct: primary] BAPIBO_DB>
```

On crée donc une nouvelle Base de données du nom de "BAPIBO_BD" et on y insère une entrée avec comme info un prénom et un âge.
De plus, on a bien l'information que l'entrée a bien été prise en compte, et on nous donne également un identifiant unique pour cette entrée.

```
mongosh mongodb://127.0.0.1:27019/?directConnection=true&serverSelectionTimeoutMS=2000
Using MongoDB:          8.0.10
Using Mongosh:          2.5.2

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/

------
   The server generated these startup warnings when booting
   2025-06-19T21:34:31.146+02:00: Access control is not enabled for the database. Re
ad and write access to data and configuration is unrestricted
   2025-06-19T21:34:31.147+02:00: This server is bound to localhost. Remote systems
will be unable to connect to this server. Start the server with --bind_ip <address>
to specify which IP addresses it should serve responses from, or with --bind_ip_all
to bind to all interfaces. If this behavior is desired, start the server with --bind
_ip 127.0.0.1 to disable this warning
------

rs0 [direct: secondary] test> use BAPIBO_DB
switched to db BAPIBO_DB
rs0 [direct: secondary] BAPIBO_DB> show collections
person
rs0 [direct: secondary] BAPIBO_DB> db.person.find()
[
  {
    _id: ObjectId('68546d3a5b78c06e0050eb67'),
    name: 'Leo-Paul',
    age: 21
  }
]
rs0 [direct: secondary] BAPIBO_DB>
```

Si on se rend donc sur la deuxième instance (secondary 27019), qu'on sélectionne notre base de donnée fraîchement créée et qu'on demande à voir ce qu'il y a dedans, on peut voir qu'il y a effectivement des data.