

TP MongoDB

1. Preparing the database

1. Check that *docker is correctly installed* by launching the hello-world image (no need to name the container).

docker run hello-world

2. *Pull the mongodb image* : mongo:latest.

docker pull mongo:latest

3. *Launch the server* mongod in some container that you will name mongoserv.

docker run -d --name mongoserv mongo

4. *Launch the Mongo shell* mongosh, in the very same docker container.

docker exec -it mongoserv mongosh

The MongoDB shell is a javascript interpreter. So we can send both javascript and MongoDB instructions. It admits many Unix shortcuts (Tab, Up, Ctrl+a, Ctrl+r, Ctrl+k, Ctrl+c. . .).

5. *Check which database you are using* and the *list of available databases*.
Switch to database bdtest. What happens? Is the database created?

- **db** – show current database
- **show dbs** – shows current databases
- **use <database_name>** - to switch the database

(In MongoDB, when you switch to a non-existing database using the **use <database_name>** command, the database is not immediately created. Instead, MongoDB creates the database only when you insert some data into it. This means that simply running `use bdtest` does not create the bdtest database unless you subsequently perform an operation that requires the database to store data, such as creating a collection and inserting a document.)

6. *Insert some arbitrary doc* in the movies collection. What happens?

- **show collections** – to view the collections within a database.
- **db.<collection_name>.insertOne**({name: 'Star Wars', year: '1970', genre: 'sci-fi'}) – to insert an arbitrary document into specified collection. To check if a collections appears: show collections. Now after show dbs command the database appears, since we inserted some data into it.
- **db.<collection_name>.insertMany**([{ name: 'Jane Doe', age: 25, address: '456 Elm St' }, { name: 'Jim Beam', age: 40, address: '789 Oak St' }]) – to insert multiple docs inside a specific collection.

7. Inserting a lot of data with individual insertion instructions is not an option. So we will use the dedicated program mongoimport which can import json and csv files. Download the file with enron mails available at <https://www.lri.fr/~maniu/enron.json>, and copy that file into the mongoserv container. Then, load the file into the emails collection of the bdenron databases

- **docker cp C:\Users\pmoll\Desktop\enron.json mongoserv:/enron.json** : After having downloaded the JSON file, you copy the file from its path “C:\Users\pmoll\Desktop\enron.json” to the container “mongoserv” and you paste it in the new path “/enron.json” (within the container).
- **docker exec mongoserv mongoimport --db bdenron --collection emails --file enron.json** :
 - mongoimport is one of the programs available on the mongo docker image. It assumes that the input file contains one JSON document per line, without any comma at the end of a line. Our document satisfies that syntax. We can therefore use the syntax: mongoimport --db --collection --file Had the file been a JSON array, we would have used option --jsonArray
 - The command executes in the container mongoserv, the program mongoimport on the file enron.json located within the collection email which is within the database bdenron. Basically, the enron.json stays in the home directory of the container mongoserv but its content goes to bdenron/emails/enron.json.
- **db.emails.find()** : Shows all the documents within the collection emails
- **db.emails.find().pretty()** : Shows all the documents within the collection emails in a pretty format (more readable)

2. MongoDB Queries

1. Connect to the database. [Count the number of documents in emails collection.](#)


- `docker start mongoserv` – To start the mongoserv
- `docker exec -it mongoserv mongosh` – To open the command line of mongoserv
- `db.emails.countDocuments()` – Counts the number of documents within the collection “emails”.

2. [What does db.emails.find\(\).pretty\(\) return?](#) Perform a few cursor iterations.

- It displays the documents within the emails collection in a pretty way.
- To perform a few cursor iterations you can execute just “it”, it will show you more content of the documents because it doesn’t show the whole content of the collection “emails”.

3. [List mails sent by no.address@enron.com.](#)

- `db.emails.find({“sender”:”no.address@enron.com”})` : Lists all documents within in the emails collection where the “sender” is “no.address@enron.com”.

4. [Display the 10th mail, by alphabetical order on sender, with the following 2 approaches:](#)  Comments

(a) [store the cursor returned by find\(\) in a javascript variable, then iterate over this cursor \(or transform the cursor into an array\).](#)

```
let sorted_senders = db.emails.find( { }, { _id:0, sender:1 } ).sort( { sender:1 } );
```

[//stores the object of sorted documents by senders in accending order](#)

```
let count = 0; // count variable to iterate the loop
```

```
while(sorted_senders.hasNext()){ //hasNext\(\) – checks if the object has next value to access
```

```
if (count === 10){
```

```
  printjson(sorted_senders.next()) // .next\(\) access the value of given doc and printjson – prints result in json format
```

```
  break; }
```

```
count++;}
```

(b) using the `skip()` and `limit()` instructions from MongoDB.

- `db.emails.find({},{}).sort({"sender":1}).skip(9).limit(1) = db.emails.find().sort({"sender":1}).skip(9).limit(1)` – Within the “emails” collection, we look for the documents (mails) sorted on alphabetical order with respect to “sender” (ascending = 1, descending = -1) and we skip the first 9 results and display only one (the 10th)
- `db.emails.find({}, {_id:0, sender:1}).sort({"sender":1}).skip(9).limit(1)` – This command line returns the same result except that instead of returning the whole document (mail), it returns just the “sender” field. If not mentioned in the projection, the rest of attributes won’t appear, however the “_id” needs to be explicitly set to 0 in order to remove it from the display.

5. List folder from which mails have been extracted. Check the type of the result.

- `db.emails.distinct("folder")` :
Lists the unique field (“folder”) values of all documents within “emails” collection.
- `typeof(db.emails.distinct("folder"))` :
Returns the types of all the previous selected documents.

```
bdenron> db.emails.distinct("folder")
[
  '_sent',
  'business',
  'compaq',
  'discussion_threads',
  'enron',
  'inbox',
  'sec_panel',
  'sent_items',
  'all_documents',
  'calendar',
  'deleted_items',
  'elizabeth',
  'family',
  'notes_inbox',
  'sent'
]
```

7. For each mail, display its author and recipient, sorting the result by author.

- `db.emails.find({}, {"_id":0,"sender":1,"recipients":1}).sort({"sender":1})`:

8. Display messages received in 2000 from April onwards.

Hint: date is stored as a string.

- `db.emails.find({"date": {$gte: "2000-04-01"}})` – gte means greater than or equal to.
- `db.emails.find({"date": {$gte: "2000-04-01", $lt: "2001-12-31"}})` – Displays messages received between those dates.

9. Display messages for which "replyto" is not null.

Hint: we may use query operators \$not and \$type, or directly compare to null

- `db.emails.find({ replyto : { $exists : true } })` - \$exists:true checks if the value of certain attribute is not null
- `db.emails.find({"replyto": {$ne : null}})` - This query will return all documents in the "emails" collection that have a replyto field, including those where replyto might be present but does not contain a useful value (an empty string).

There are none.

10. Number of mails sent by each operator in the above time range(2000 from April onward). Sort the result by decreasing number of mails.

Hint: sorting will be one step in the aggregation pipeline. One can indeed not perform a sort() on the somewhat specific cursor returned by aggregate(). If you want to know more, check the doc about aggregate.

- `db.emails.aggregate([
 {$match : { "date": {$gte: "2000-04-01"} }},
 {$group : { "_id" : "$sender", "nb_emails" : {$sum: 1} }},
 {$sort: {nb_emails : -1} }
])`

```
[
  { _id: 'rosalee.fleming@enron.com', nb_emails: 757 },
  { _id: 'brown_mary_jo@lilly.com', nb_emails: 82 },
  { _id: 'leonardo.pacheco@enron.com', nb_emails: 78 },
  { _id: 'savont@email.msn.com', nb_emails: 66 },
  { _id: 'tori.wells@enron.com', nb_emails: 57 },
  { _id: 'elizabeth.davis@compaq.com', nb_emails: 50 },
  { _id: 'katherine.brown@enron.com', nb_emails: 47 },
  { _id: 'no.address@enron.com', nb_emails: 38 },
  { _id: 'lizard_ar@yahoo.com', nb_emails: 36 },
  { _id: 'karen.denne@enron.com', nb_emails: 35 },
  { _id: 'svarga@kudlow.com', nb_emails: 34 },
  { _id: 'mrslinda@lplpi.com', nb_emails: 34 },
  { _id: 'rob.bradley@enron.com', nb_emails: 32 },
  { _id: 'jeffrey.garten@yale.edu', nb_emails: 32 },
  { _id: 'enron.announcements@enron.com', nb_emails: 30 },
  { _id: 'listadmin@client-mail.com', nb_emails: 27 },
  { _id: 'shea_dugger@i2.com', nb_emails: 26 },
  { _id: 'perfmgmt@enron.com', nb_emails: 25 },
  { _id: 'sally.keepers@enron.com', nb_emails: 25 },
  { _id: 'joe.hillings@enron.com', nb_emails: 25 }
]
```

10. Using the doc, and through experimentation, check what the following query returns:

- `db.emails.find({ text: {$regex: '^(?si).*P(?-si)resident Bush.*$'} })`

The regular expression `{ $regex: '^(?si).*P(?-si)resident Bush.*$' }` is used in MongoDB to match strings that contain a specific pattern. Let's break down this regular expression to understand what it's looking for:

- `^`: Anchors the match at the beginning of the string.
- `(?si)`: Sets flags for the regular expression. `s` makes the dot (`.`) match newline characters, and `i` makes the match case-insensitive.
- `.*`: Matches any character (including newline due to the `s` flag) zero or more times.
- `P(?-si)resident`: Matches the word "President" with a case-sensitive 'P'. The `(?-si)` turns off the case-insensitive and dot-all flags for this part of the pattern.
- `Bush`: Matches the word "Bush".
- `.*`: Again, matches any character (including newline) zero or more times.
- `$`: Anchors the match at the end of the string.

Overall, this regular expression looks for strings that contain the word "President" (with a capital 'P') followed by the word "Bush", anywhere in the string. The rest of the string can contain any characters, and the match is case-insensitive except for the capital 'P' in "President".

Examples of strings that match this regular expression:

- **Match**: "Our meeting with President Bush was enlightening."

Explanation: The string contains "President Bush" with a capital 'P'.

- **Match**: "president Bush discussed important matters."

Explanation: The string contains "president Bush". The case-insensitive flag allows "president" to match, but the 'P' is not capitalized.

- **Match**: "The agenda includes a speech by President Bush\nnon environmental policies."

Explanation: The string contains "President Bush" and spans multiple lines, which is allowed due to the dot-all flag.

Examples of strings that do not match:

- **No Match**: "President bush is expected to arrive."

Explanation: Although it contains "President", "bush" is not capitalized, and the match is case-sensitive for "Bush".

- **No Match:** "The Presidential address was well-received."

Explanation: The string does not contain "President Bush".

- **No Match:** "We spoke with the president, Mr. Bush."

Explanation: "President" and "Bush" are not adjacent in the string; there are other words in between.

3. MongoDB on multiple nodes: replication

1. Launch 3 mongod servers, each in its own container:

- We want the servers to be listening on ports 27018, 27019, 27020 respectively (that's for the sake of the exercise: actually we could pick any 3 available ports).
For this, mongod must specify the port with --port 27018.
- The servers must have replication enabled : mongod must specify --replSet repl.

Remarks:

– Specifying mongod is optional as it's the default program in the mongo image.

– replSet is mandatory: without replication enabled, you will be prevented from initializing the replica set in the next questions.

– The ports are optional: without specifying ports we would use the same default port for 27017 for all servers (each container has a different IPs so having the same port is not an issue).

– If you are running docker on your personal laptop and NOT on the servers of PUIO, you could additionally put the containers in a dedicated docker network that you may call my-mongo-network, which makes it easier to communicate between containers docker network create my-mongo-network. Then you may connect the containers you create to the network with: docker run --network my-mongo-network ...

- **docker network create my-mongo-network** - To create the network, otherwise you can't integrate containers to a non-existent network.
- **docker network ls** – To check the networks which are created

- `docker inspect container_name` – Displays all the information of the container and in the field “Networks” you can check to which networks belongs.
- `docker run --name mongo27018 -d --network my-mongo-network -p 27018:27017 mongo --replSet repl --port 27018`
- `docker run --name mongo27019 -d --network my-mongo-network -p 27019:27017 mongo --replSet repl --port 27019`
- `docker run --name mongo27020 -d --network my-mongo-network -p 27020:27017 mongo --replSet repl --port 27020`

2. Launch the mongosh interpreter in containers, specifying the corresponding port.

- `docker exec -it mongo27018 mongosh --port 27018`
- `docker exec -it mongo27019 mongosh --port 27019`
- `docker exec -it mongo27020 mongosh --port 27020`

3. On the host, check the IP of each container:

- `docker inspect -f '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' mongo27018`
- `docker inspect -f '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' mongo27019`
- `docker inspect -f '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' mongo27020`

```
C:\Users\pmoll>docker inspect -f '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' mongo27018
'172.19.0.2'

C:\Users\pmoll>docker inspect -f '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' mongo27019
'172.19.0.3'

C:\Users\pmoll>docker inspect -f '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' mongo27020
'172.19.0.4'
```

From one of the servers, initialize the replica set with the 3 servers:

- `rs.initiate({ _id:"repl", members:[{ _id:0, host:"ip1:port1"}, { _id:1, host:"ip2:port2"}, { _id:2, host:"ip2:port3"}] })`
- `rs.initiate({ _id:"repl", members:[{ _id:0, host:"172.19.0.2:27018"}, { _id:1, host:" 172.19.0.3:27019"}, { _id:2, host:" 172.19.0.4:27020"}] })`

Remark: if you are running the containers on your own laptop and in a dedicated network, no need to identify the containers by their ip; you may directly indicate the container name or id (with the corresponding port) as an host.

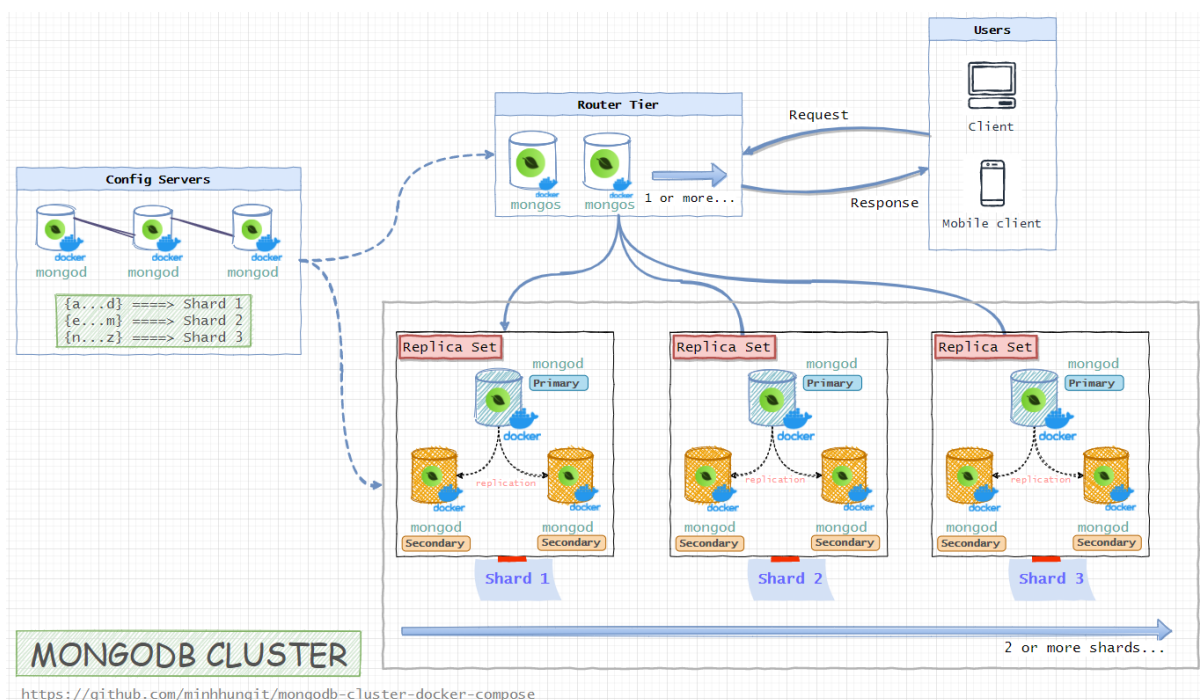
4. Check the nodes taking part in the replica set with:

■ `rs.conf()`.

For more information about synchronization (check the primary node and secondary nodes), use `rs.status()` or equivalently `db.adminCommand(replSetGetStatus : 1)`.

4. On your own computer only: MongoDB on multiple nodes: Sharding

1. Setup a sharded cluster with docker-compose by following the instructions at <https://github.com/minhhungit/mongodb-cluster-docker-compose>.



- Step 1: Start all of the containers

`docker-compose up -d`

- Step 2: Initialize the replica sets (config servers and shards)

Run these command one by one:

docker-compose exec configsvr01 sh -c "mongosh < /scripts/init-configserver.js"

docker-compose exec shard01-a sh -c "mongosh < /scripts/init-shard01.js"

docker-compose exec shard02-a sh -c "mongosh < /scripts/init-shard02.js"

docker-compose exec shard03-a sh -c "mongosh < /scripts/init-shard03.js"

- Step 3: Initializing the router

docker-compose exec router01 sh -c "mongosh < /scripts/init-router.js"

- Step 4: Enable sharding and setup sharding-key

docker-compose exec router01 mongosh --port 27017

2. Check the status of your database with

sh.status() - it provides information about the sharded clusters, such as the configuration of shards, databases, collections, and the distribution of data among them.

db.printShardingStatus() - is an alternative to **sh.status()**

db.stats() - This command provides statistics about the current database. It returns data like the number of collections, documents, indexes, the size of the database, and other storage-related information.

3. Create a database bdpart, with a collection dblp. [Authorize partitioning\(sharding\)](#) of collections in database bdpart:

db.adminCommand({ enableSharding: "bdpart" })

Partition through hashing the collection dblp on the field "ident":

sh.shardCollection("bdpart.dblp", {"ident": "hashed"})

If the command above doesn't work initially, use:

db.dblp.createIndex({ "ident": "hashed" }) – to create hashed index on the shard key

4. Retrieve and unzip the data:

wget <http://b3d.bdpedia.fr/files/dblp.json.zip>;

unzip dblp.json.zip

You will see that the file already contains a field `_id` whose values are unfit for a mongodb identifier.

Consequently, you should rename this field into ident in PowerShell.

(Get-Content dblp.json) -replace '"_id"', '"ident"' | Set-Content dblp.json

5. Copy the file in the router container, then load the data with mongoimport: you need to use --jsonArray given the format of that document. Observer.

docker exec router-01 mongoimport --db bdpart --collection dblp --file /dblp.json --jsonArray

6. Write a query that returns in alphabetical order (on authors) all documents in collection dblp for which "type" = "Article". No need to run the query, but analyze its execution plan with explain(): .explain()

db.dblp.find({"type": "Article"}).sort({"authors": 1}).explain()

DATOS:

In MongoDB, a NoSQL document-oriented database, there are various commands and methods for managing replicas and shards, which are crucial for scalability and high availability. Here's an explanation of what `rs.add`, `rs.initiate`, and `sh.addShard` do, along with examples:

1. rs.initiate()

- **What It Does:** This command initiates a new replica set. A replica set in MongoDB is a group of `mongod` processes that maintain the same set of data. `rs.initiate()` configures the replica set according to the provided options or with default settings if no options are given.

- **Example:**

```
rs.initiate(  
  {  
    _id: "myReplicaSet",  
    members: [  
      { _id: 0, host: "mongodb0.example.com:27017" },  
      { _id: 1, host: "mongodb1.example.com:27017" },  
      { _id: 2, host: "mongodb2.example.com:27017" }  
    ]  
  })
```

2. `rs.add()`

- **What It Does:** This command adds a new member to an existing replica set. You can specify a host (domain name and port) and, optionally, additional options for the new replica set member.

- **Example:**

```
rs.add("mongodb3.example.com:27017")
```

This command would add a new member with the specified host to the current replica set.

3. `sh.addShard()`

- **What It Does:** In a MongoDB environment with sharding enabled, this command adds a new shard to the cluster. A shard can be a single `mongod` or an entire replica set. Sharding is a methodology for distributing data across multiple servers, and shards are the horizontal partitions that host subsets of the data.

- **Example:**

- For a single mongod server as a shard:

```
sh.addShard("mongodb4.example.com:27017")
```

- For a replica set as a shard:

```
sh.addShard("myReplicaSet/mongodb5.example.com:27017,mongodb6.example.com:27017")
```

In the second example, the shard being added is a replica set named `myReplicaSet` with specified members.

These commands are essential for MongoDB cluster administration, allowing database administrators and developers to horizontally scale their applications and ensure data availability through replicas.