# Document Stores, MongoDB Lab*

LSDM

In this lab, we will use Docker to launch MongoDB servers and clients. We will first work on a single node, then on multiple nodes with replication. Note that if you are working on Ensimag computers, the alternative to Docker is Podman. For most cases podman commands are identical to docker commands. Useful documents:

- Docker tutorial: https://docs.docker.com/get-started/
- Docker instructions: https://docs.docker.com/engine/reference/commandline/cli/
- Podman documentation: https://docs.podman.io/en/latest/
- Everything you need to know about MongoDB is in the doc.

## 1 Preparing the Database

1. Launch the server `mongod` in some container that you will name `mongoserv`. If you want to know more about the MongoDB image, you may check https://hub.docker.com/_/mongo/, but you don't have to.

   ```
   docker run -d -p 27017:27017 -name mongoserv mongo
   ```

2. 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...).

3. Check which database you are using and the list of available databases. Switch to database `bdtest`. What happens? is the database created?

4. Insert some arbitrary doc in the `movies` collection. What happens?

5. 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://silviu.maniu.info/teaching/enron.json, and copy that file into the `mongoserv` container. Then, load the file into the `emails` collection of the `bdenron` databases.

   > `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 <base> --collection <collection> --file <fichier.json>
   > ```
   >
   > Had the file been a JSON array, we would have used option `--jsonArray`

*Inspired from labs by B. Groz et al.

## 2 MongoDB Queries

1. Connect to the database. Cout the number of documents in `emails` collection.
2. What does `db.emails.find().pretty()` return? Perform a few cursor iterations.
3. List mails sent by `no.address@enron.com`.
4. Display the $10^{\text{th}}$ mail, by alphabetical order on sender, with the following 2 approaches:
   a) store the cursor returned by `find()` in a javascript variable, then iterate over this cursor (or transform the cursor into an array)
   b) using the `skip()` and `limit()` instructions from MongoDB.
5. List `folder` from which mails have been extracted. Check the type of the result..
6. For each mail, display its author and recipient, sorting the result by author.
7. Display messages received in 2000 from April onwards.
   *Hint: date is stored as a string.*
8. Display messages for which "replyto" is not null.
   *Hint: we may use query operators* `$not` *and* `$type`*, or directly compare to* ***`null`***
9. Number of mails sent by each operator in the above time range. 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.
10. Using the doc, and through experimentation, check what the following query returns:

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

## 3 MongoDB on multiple nodes: replication

Instructions and parameters available on mongo image (mongod server by default, mongos for router)

`mongosh` the mongoDB shell, which can be executed on servers `mongod` or `mongos`

`mongod --replSet <nomset> --port <numport>` launches server (daemon) `mongod`:

`--replSet` allows to connect containers into the replica set.

`--port` specifies on which port the server is listening

In case of sharding:

`--configserv` for a configuration server

`--shardsvr` for a server storing data

`mongos --port 27017 --configdb <nomsetcfg>/<hote>:<numportcfg>` launches a router

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).
- 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 ...`

2. Launch the `mongosh` interpreter in containers, specifying the corresponding port. `docker exec -it mongoserv1 mongosh --port 27018`

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

   ```
   docker inspect -f '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' mongoserv1
   ```

   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"}]
     })
   ```

   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 )`.