# Guided Practice on MongoDB Atlas

In this document we will work on MongoDB Atlas, a hosted MongoDB service to create a database, load documents and execute queries.

In a second part we will see how to work on larger databases with `mongosh` in the terminal.

At the end of this session, you should have a good understanding of MongoDB's query language (MQL).

Objectives:

- You connect to a MongoDB Atlas server
- You create a database and insert documents
- You build relatively complex queries on a MongoDB database

---

## Several ways to work with MongoDB

On your local machine

- MongoDB Community Edition Self-hosted, runs on your computer
- MongoDB Compass - the GUI tool for visualization
- `mongosh` the CLI (command line interface) in the terminal.

The MongoDB Shell, mongosh, is a JavaScript and Node.js **REPL** (Read-Eval-Print Loop) environment for interacting with MongoDB servers.

You can also work on MongoDB with your preferred scripting language: python, go, ruby, PHP, Java, etc...

Today, we are working on the hosted Atlas version.

- MongoDB Atlas provides a Cloud hosted service with free or paid offerings. **You don't have to install anything**.

MongoDB Atlas gives a hosted cluster on which we can create a database, import sample **collections** and understand how to perform CRUD operations in MongoDB.

> **CRUD** means Create, Read, Update and Delete: the four basic operations for a DBMS

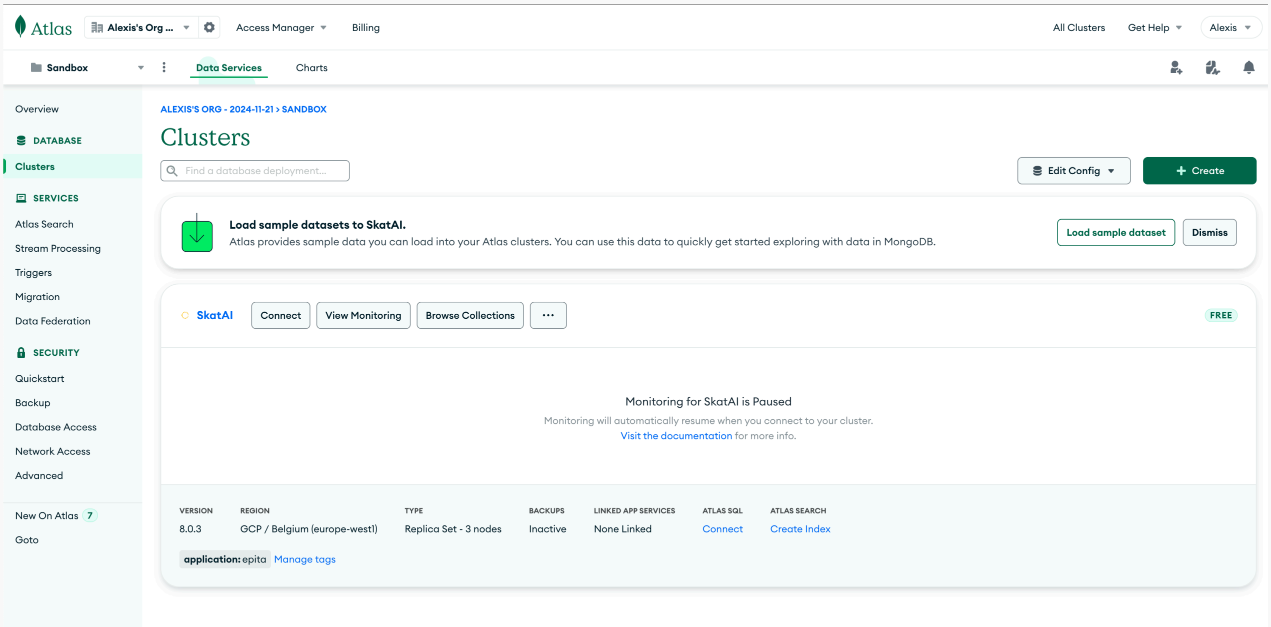
---

## Atlas

Let's start by creating an account on Atlas https://www.mongodb.com/cloud/atlas/register

Then:

- Create a cluster and a project (I called my project `Sandbox`)



ALEXIS'S ORG - 2024-11-21 > SANDBOX

Clusters

Find a database deployment...

**Load sample datasets to SkatAI.**
Atlas provides sample data you can load into your Atlas clusters. You can use this data to quickly get started exploring with data in MongoDB.

○ SkatAI

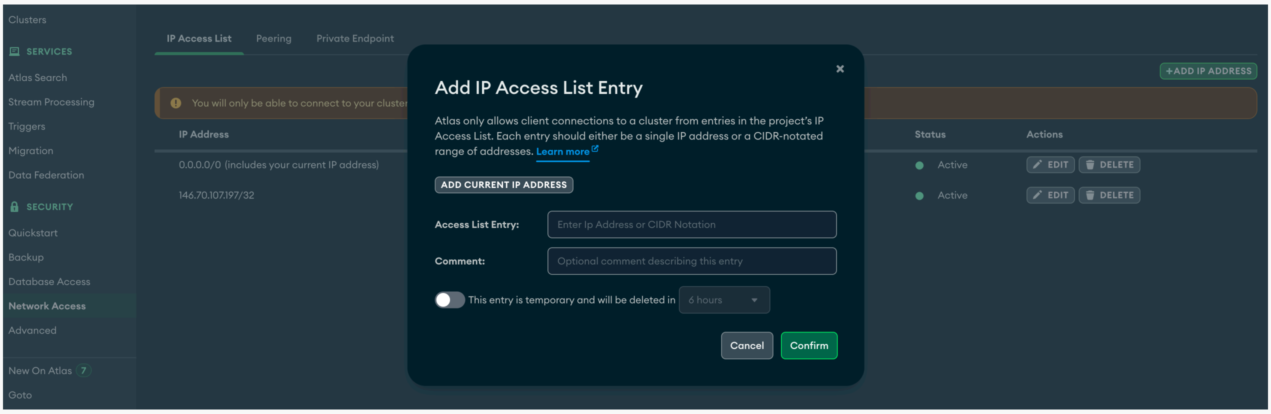
Monitoring for SkatAI is Paused
Monitoring will automatically resume when you connect to your cluster.
Visit the documentation for more info.

| VERSION | REGION | TYPE | BACKUPS | LINKED APP SERVICES | ATLAS SQL | ATLAS SEARCH |
|---|---|---|---|---|---|---|
| 8.0.3 | GCP / Belgium (europe-west1) | Replica Set - 3 nodes | Inactive | None Linked | Connect | Create Index |

**application:** epita  Manage tags

## Important

- **Copy your database password.**
- Check that your **IP** address is authorized. Visit `Network Access` (left navigation) and add it to the list
- To allow access from any IP add 0.0.0.0/0. This is not recommended for production but easier for our context.



You will only be able to connect to your cluster

| IP Address | Status | Actions |
|---|---|---|
| 0.0.0.0/0 (includes your current IP address) | ● Active | EDIT DELETE |
| 146.70.107.197/32 | ● Active | EDIT DELETE |

Add IP Access List Entry

Atlas only allows client connections to a cluster from entries in the project's IP Access List. Each entry should either be a single IP address or a CIDR-notated range of addresses. Learn more

Access List Entry: Enter Ip Address or CIDR Notation

Comment: Optional comment describing this entry

○ This entry is temporary and will be deleted in 6 hours

## Then create a MongoDB database

- Go to `clusters`
- Create a database: add the name `songsdb` and the collection name `songs`,

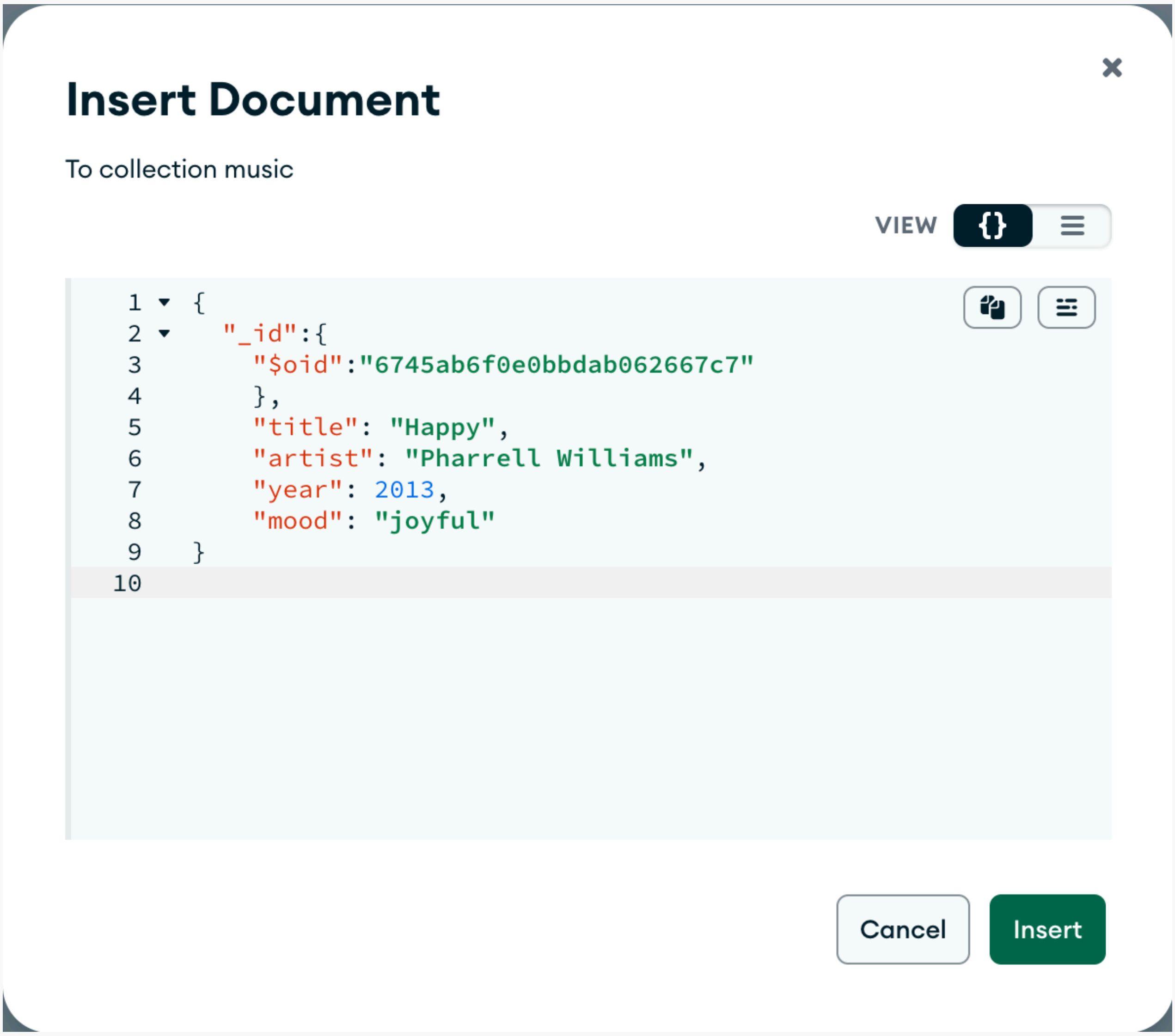
Note: do not confuse

- the project: `Sandbox`

- the database: `songsdb`
- the collection (table): `songs`

Then insert the following Document

```JavaScript
{
  "_id":{
    "$oid":"6745ab6f0e0bbdab062667c7"
    },
    "title": "Happy",
    "artist": "Pharrell Williams",
    "year": 2013,
    "mood": "joyful"
}
```



## Insert Document

To collection music

```
{
  "_id":{
    "$oid":"6745ab6f0e0bbdab062667c7"
    },
    "title": "Happy",
    "artist": "Pharrell Williams",
    "year": 2013,
    "mood": "joyful"
}
```

And insert another one

JavaScript

```
{
  "_id":{
    "$oid":"abcdeab6f0e0bbdab062667aa"
  },
  "title": "Highway to hell",
  "artist": "AC/DC",
  "year": 1981,
  "mood": "energetic"
}
```

In both cases, the value of the `_id` is arbitrary. The tool proposes a default one with another value.

We therefore have 2 documents!

2 documents, wow that's mega exciting 🤣🤣🤣 !

Let's explore this MongoDB database.

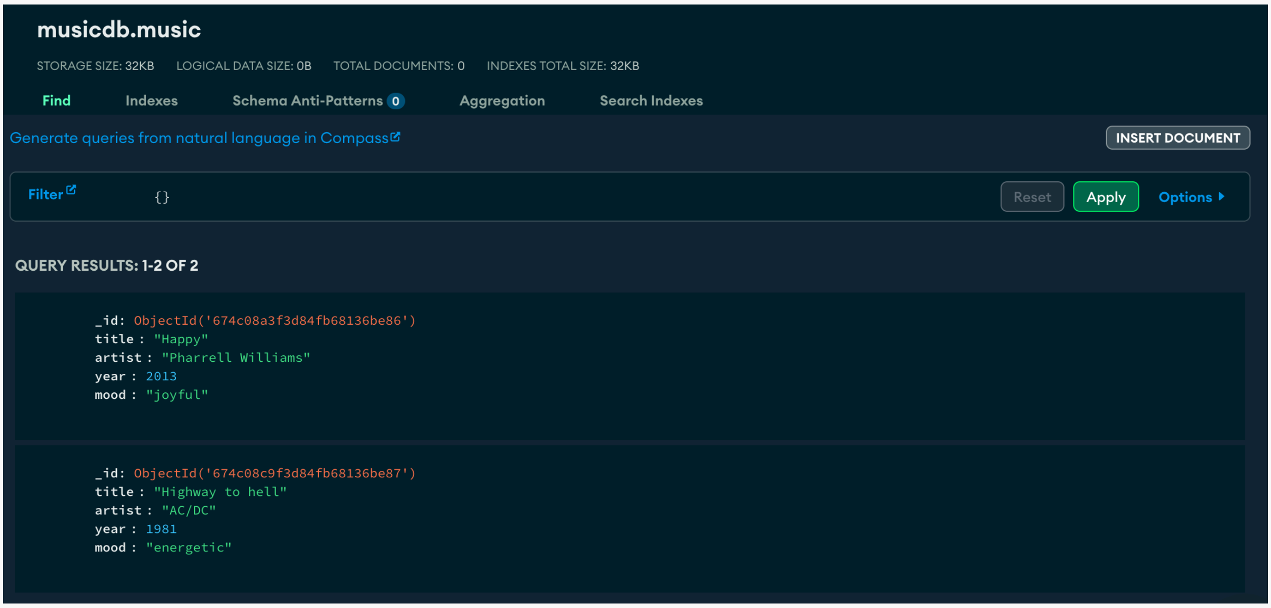
## Queries in MongoDB

In MongoDB, writing a query comes down to writing `JSON`

| json | query |
|---|---|
| `{}` | returns all documents |
| `{ field : value }` | where field = value |
| `{ field : { $lt : value } }` | where field <= value (lt : less than) |

So if we want to find all documents in our `songs` collection, just write `{}` in the query field

This returns the 2 songs, the 2 documents.



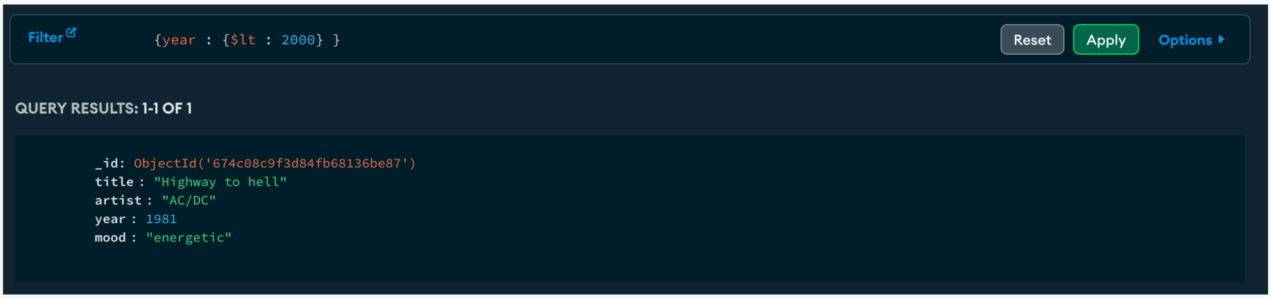
Filter: {}

QUERY RESULTS: 1-2 OF 2

```
_id: ObjectId('674c08a3f3d84fb68136be86')
title : "Happy"
artist : "Pharrell Williams"
year : 2013
mood : "joyful"
```

```
_id: ObjectId('674c08c9f3d84fb68136be87')
title : "Highway to hell"
artist : "AC/DC"
year : 1981
mood : "energetic"
```

And the query `{ year: { $lt: 2000 } }` returns the song that was published before 2000.



QUERY RESULTS: 1-1 OF 1

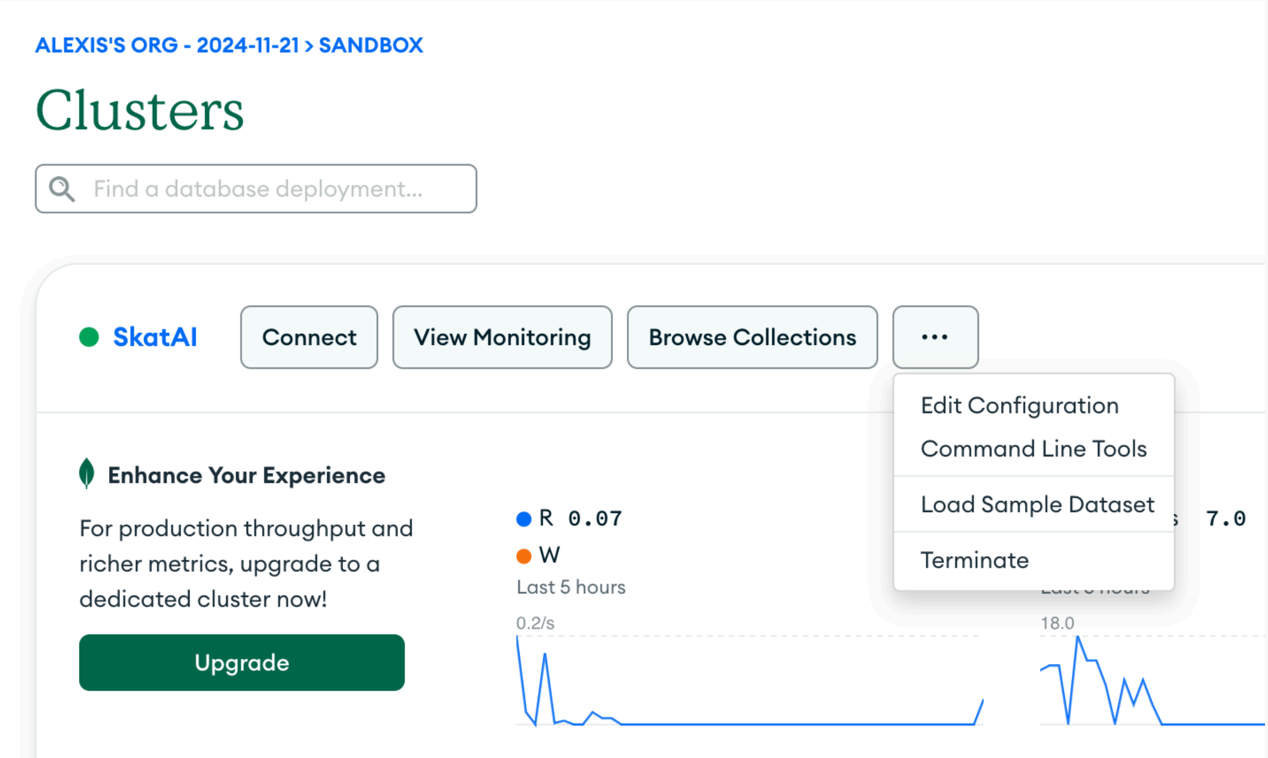
```
_id: ObjectId('674c08c9f3d84fb68136be87')
title : "Highway to hell"
artist : "AC/DC"
year : 1981
mood : "energetic"
```

## Moooooore data! (moaaaaaarrrr data)

2 documents is good but we need more data to really play.

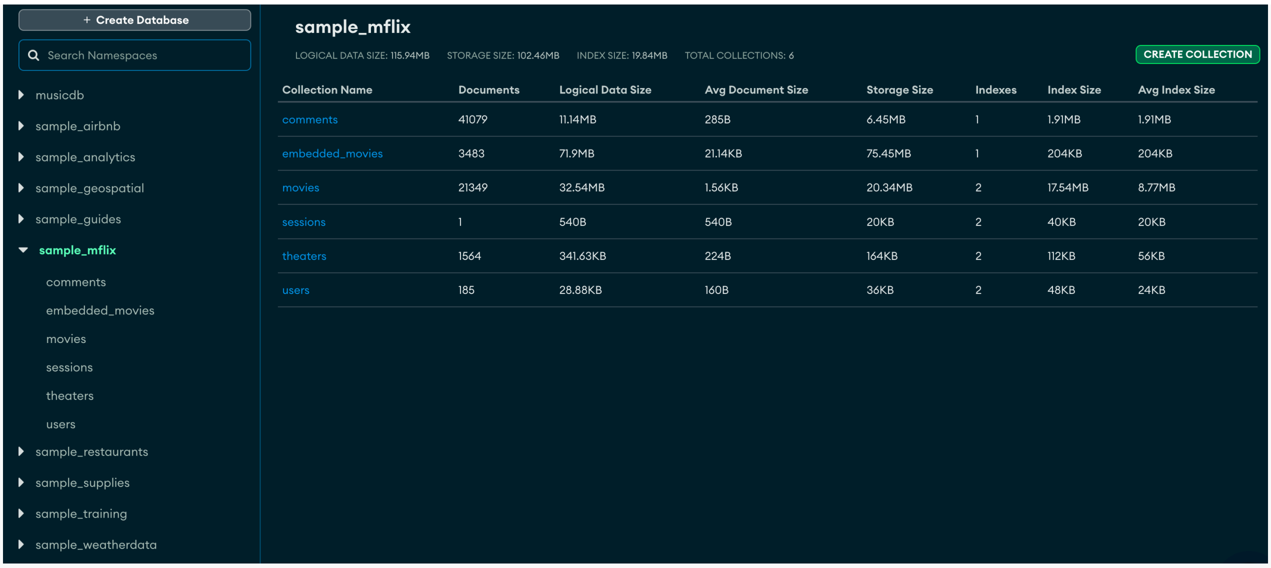
Let's now import the sample datasets provided by ATLAS.

Go to `Clusters > dots > load sample dataset`



Then click on your project name (for me `Sandbox`) and `Browse collections` to see the available databases.

You get a new database `movies_mflix` with 5 collections.



sample_mflix

LOGICAL DATA SIZE: 115.94MB STORAGE SIZE: 102.46MB INDEX SIZE: 19.84MB TOTAL COLLECTIONS: 6

| Collection Name | Documents | Logical Data Size | Avg Document Size | Storage Size | Indexes | Index Size | Avg Index Size |
|---|---|---|---|---|---|---|---|
| comments | 41079 | 11.14MB | 285B | 6.45MB | 1 | 1.91MB | 1.91MB |
| embedded_movies | 3483 | 71.9MB | 21.14KB | 75.45MB | 1 | 204KB | 204KB |
| movies | 21349 | 32.54MB | 1.56KB | 20.34MB | 2 | 17.54MB | 8.77MB |
| sessions | 1 | 540B | 540B | 20KB | 2 | 40KB | 20KB |
| theaters | 1564 | 341.63KB | 224B | 164KB | 2 | 112KB | 56KB |
| users | 185 | 28.88KB | 160B | 36KB | 2 | 48KB | 24KB |

Look at the `embedded_movies` collection which contains 1525 documents and observe the structure of a document

### A document is a JSON record

It can have:

- **nested JSON**: look at the nested dictionaries `imdb` and `tomatoes` fields.
- **arrays**: look at genres, cast, languages, writers, ...

### The primary key of a collection is always "`_id`".

JavaScript

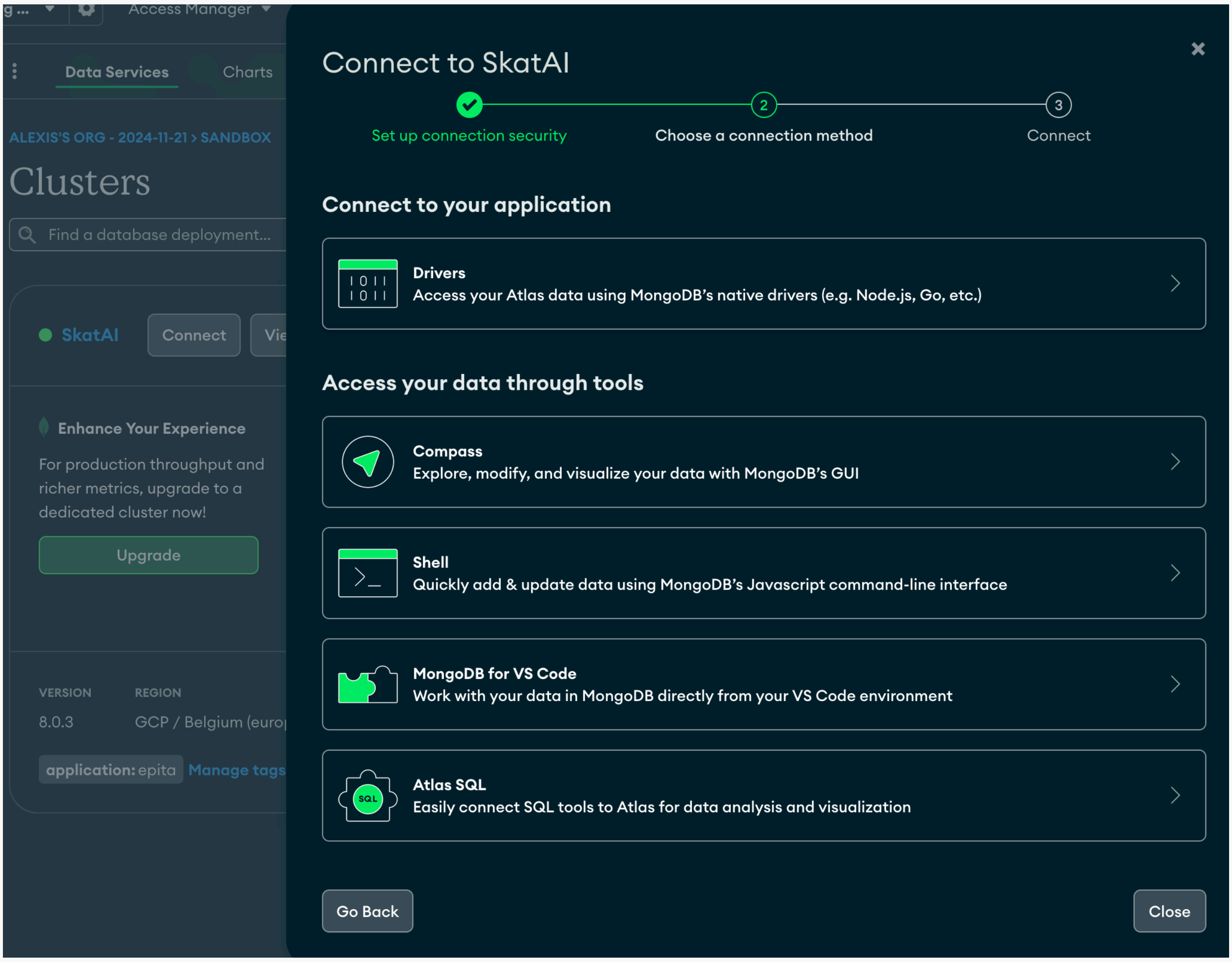
```
{
    "_id": {
        "$oid": "573a1390f29313caabcd5293"
    },
    "plot": "Young Pauline is left a lot of money when her wealthy uncle dies. H
    "genres": [
        "Action"
    ],
    "runtime": {
        "$numberInt": "199"
    },
    "cast": [
        "Pearl White",
        "Crane Wilbur",
        "Paul Panzer",
        "Edward Josè"
    ],
    "num_mflix_comments": {
        "$numberInt": "0"
    },
    "poster": "https://m.media-amazon.com/images/M/MV5BMzgxODk1Mzk2Ml5BMl5BanBnX
    "title": "The Perils of Pauline",
    "fullplot": "Young Pauline is left a lot of money when her wealthy uncle die
    "languages": [
        "English"
    ],
    "released": {
        "$date": {
            "$numberLong": "-1760227200000"
        }
    },
    "directors": [
        "Louis J. Gasnier",
        "Donald MacKenzie"
    ],
    "writers": [
        "Charles W. Goddard (screenplay)",
        "Basil Dickey (screenplay)",
        "Charles W. Goddard (novel)",
        "George B. Seitz",
        "Bertram Millhauser"
    ],
    "awards": {
        "wins": {
            "$numberInt": "1"
        },
        "nominations": {
            "$numberInt": "0"
        },
        "text": "1 win."
    },
    "lastupdated": "2015-09-12 00:01:18.647000000",
    "year": {
        "$numberInt": "1914"
    },
    "imdb": {
        "rating": {
            "$numberDouble": "7.6"
        },
        "votes": {
            "$numberInt": "744"
        },
        "id": {
            "$numberInt": "4465"
        }
    },
    "countries": [
        "USA"
    ],
    "type": "movie",
    "tomatoes": {
        "viewer": {
            "rating": {
                "$numberDouble": "2.8"
            },
            "numReviews": {
                "$numberInt": "9"
            }
        },
        "production": "Pathè Frères",
        "lastUpdated": {
            "$date": {
                "$numberLong": "1441993579000"
            }
        }
    }
}
```

The ATLAS user interface is well done but I prefer working with code rather than a web interface.

Let's move to python and `mongosh` to explore this movie database and learn how to make queries in MongoDB.

## Connecting via your language

Atlas allows you to connect to your cluster



Connect to SkatAI

✓ Set up connection security — 2 Choose a connection method — 3 Connect

Connect to your application

Drivers
Access your Atlas data using MongoDB's native drivers (e.g. Node.js, Go, etc.)

Access your data through tools

Compass
Explore, modify, and visualize your data with MongoDB's GUI

Shell
Quickly add & update data using MongoDB's Javascript command-line interface

MongoDB for VS Code
Work with your data in MongoDB directly from your VS Code environment

Atlas SQL
Easily connect SQL tools to Atlas for data analysis and visualization

Enhance Your Experience

For production throughput and richer metrics, upgrade to a dedicated cluster now!

## Reset and password management

If you forgot your cluster password

## Connecting with Mongosh

Save the connection string in an environment variable

```
mongodb+srv://username:password@cluster_id.mongodb.net/
```

For example

```bash
export MONGO_ATLAS_URI=mongodb+srv://alexis:password@skatai.w932a.mongodb.net/
```

Then connect with

```bash
mongosh ${MONGO_ATLAS_URI}
```

### Connecting in Python

We need the pymongo package

```bash
pip install pymongo
```

The connection string is

```python
connection_string = "mongodb+srv://alexis:<db_password>@skatai.w932a.mongodb.net
```

Note: If you work in python, it's better to put the connection string as an environment variable (`$MONGO_ATLAS_URI`), in a `.env` file. then load it with `dotenv`

```python
import os
from dotenv import load_dotenv

load_dotenv()

MONGO_ATLAS_URI = os.getenv('MONGO_ATLAS_URI')
```

And then instantiate the client with

```python
from pymongo import MongoClient

client = MongoClient(MONGO_ATLAS_URI)
```

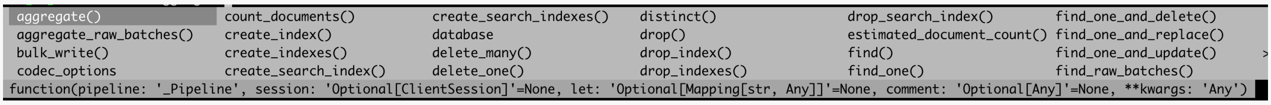
Once we have a client, we can connect to the database

```python
db = client["sample_mflix"]
```

then instantiate a collection

```python
collection = db["movies"]
```

The collection is of class `pymongo.synchronous.collection.Collection` and has many methods:



```
 aggregate()               count_documents()         create_search_indexes()   distinct()                drop_search_index()         find_one_and_delete()
 aggregate_raw_batches()   create_index()            database                  drop()                    estimated_document_count()  find_one_and_replace()
 bulk_write()              create_indexes()          delete_many()             drop_index()              find()                      find_one_and_update()    >
 codec_options             create_search_index()     delete_one()              drop_indexes()            find_one()                  find_raw_batches()
function(pipeline: '_Pipeline', session: 'Optional[ClientSession]'=None, let: 'Optional[Mapping[str, Any]]'=None, comment: 'Optional[Any]'=None, **kwargs: 'Any')
```

## Using the terminal with `mongosh`

How to install `mongosh`: https://www.mongodb.com/docs/mongodb-shell/install/

The language used in the MongoDB shell (`mongosh`) is **JavaScript**.

`mongosh` is an interactive JavaScript interface for MongoDB, allowing you to interact with your MongoDB instances, execute queries or perform administrative tasks in JavaScript.

`mongosh` also supports many JavaScript features including the use of variables, loops and functions.

Here are some examples of using JavaScript in `mongosh`:

- **Connect to a MongoDB instance:**

Local

```bash
mongosh "mongodb://localhost:27017"
```

On Atlas

```bash
mongosh mongodb+srv://alexis:password@skatai-ipsa.w932a.mongodb.net/
```

or

```Bash
mongosh ${MONGO_ATLAS_URI}
```

Once connected to an instance

- **see the databases:**

```JavaScript
show dbs
```

- **Change database:**

```JavaScript
use myDatabase
```

- **see the collections:**

```JavaScript
show collections
```

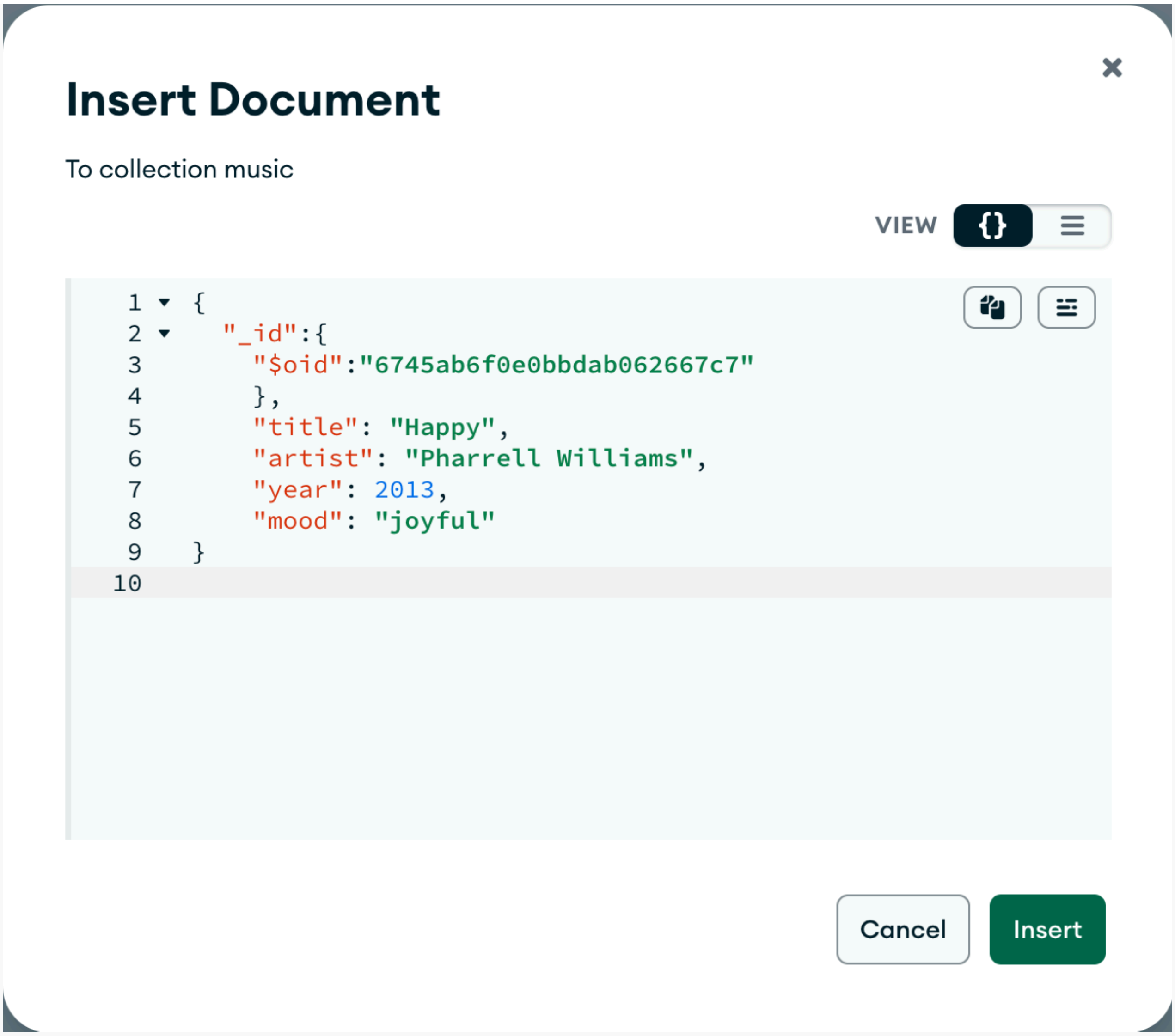
- **Insert a document**

```JavaScript
db.songs.insertOne(
    {
        "title": "Green Onions",
      "artist": "Booket T and the MGs",
      "year": 1967,
      "mood": "[blues, soul]"
  }
)
```

```JavaScript
db.users.updateOne(
{ "artist": "Booket T and the MGs" },
{ $set: { "artist": "Booker T and the MGs" } }
);
```

```JavaScript

db.songs.insertOne(
    {
    "title": "Happy",
    "artist": "Pharrell Williams",
    "year": 2013,
    "mood": "joyful"
})
```



Insert Document

To collection music

```
{
  "_id":{
    "$oid":"6745ab6f0e0bbdab062667c7"
    },
    "title": "Happy",
    "artist": "Pharrell Williams",
    "year": 2013,
    "mood": "joyful"
}
```

And insert another one

```javascript
db.songs.insertOne(
    {
  "title": "Highway to hell",
  "artist": "AC/DC",
  "year": 1981,
  "mood": "energetic"
})
```

- **Execute a query**

```javascript
db.songs.find({ year: { $lt: 2000 } })
```

- **Update a document**

```javascript
db.myCollection.updateOne({ name: "Alice" }, { $set: { age: 31 } })
```

- **Delete a document:**

JavaScript

```
db.myCollection.deleteOne({ name: "Alice" })
```

- **Use JavaScript variables and functions**

JavaScript

```
var name = "Alice";
db.myCollection.find({ name: name }).forEach(printjson);
```

`mongosh` is a powerful tool for querying and managing your MongoDB databases.

### Difference between queries in python and javascript

python: double quotes around fields and operators

Python

```
db.movies.find(
    {"runtime": {"$gt" : 180}},  // Filter on movie duration
    { "_id": 0, "title": 1, "runtime": 1, "imdb.rating": 1 }  // Projection to i
)
```

`mongosh`: no need for quotes

JavaScript

```
db.movies.find(
    {runtime: {$gt : 180}},  // Filter on movie duration
    { _id: 0, title: 1, runtime: 1, "imdb.rating": 1 }  // Projection to include
)
```

### Filtering

https://www.mongodb.com/docs/manual/reference/glossary/

The JSON that specifies the filtering arguments is called a **query predicate**. It's an expression that returns a boolean indicating whether a document matches the specified query.

For example, `{ title: { $eq: "Top Gun" } }`, which returns documents that have a "`title`" field whose value is "`Top Gun`".

An empty query predicate (`{ }`) returns all documents in the collection.

Main functions on collections

| function | returns |
|---|---|
| `find()` | all documents |
| `find_one()` | the 1st document |
| `distinct("<field>")` | list of distinct values for the `<field>` |
| `count_documents({})` | number of documents for the collection or returned by the filter in the query predicate |

Also note

- `find_one_and_replace()`
- `find_one_and_update()` and
- `delete_many()`
- `delete_one()`
- `drop_index()`
- etc ...

Note: You can also query the collection directly from the client with `db.<collection_name>.find()` without having to instantiate a `collection` object.

```bash
collection.find({})
# or
db.movies.find({})
```

You can chain these methods with `limit` and `sort`

```bash
db.movies.find({runtime: {$gt: 120}}).limit(3)
```

### Cursor

The result returned is a **cursor**.

```javascript
cursor = db.movies.find({})
```

A cursor is a pointer to a MongoDB query result set.

### Projection

In database language, **projecting** means selecting a subset of all possible fields.

In SQL, you simply list the column names

```SQL
select genres, plot from movies;
```

In MongoDB, you must specify the fields in a JSON object, right after the query predicate

```JavaScript
db.movies.find(
    {runtime: {$gt : 180}},  // Filter on movie duration
    { _id: 0, title: 1, runtime: 1, "imdb.rating": 1 }  // Projection to include
)
```

Here the projection is expressed by: `({ _id: 0, title: 1, runtime: 1, "imdb.rating": 1 })`

- `title: 1`: includes the title field.
- `runtime: 1,` includes the runtime
- `"imdb.rating": 1`: Includes the `imdb.rating` field.
- `_id: 0`: Excludes the `_id` field from the result (the default value is 1 if not specified).

The query returns

```JavaScript
  { runtime: 240, title: 'Napoleon', imdb: { rating: 7.4 } },
  { runtime: 281, title: 'Les Misèrables', imdb: { rating: 7.9 } },
  { runtime: 245, title: 'Flash Gordon', imdb: { rating: 7.3 } },
  { runtime: 238, title: 'Gone with the Wind', imdb: { rating: 8.2 } },
```

## Exercises

### In python

Let's execute some queries in Python or `mongosh` on the `movies` database

```Python
import os
from pymongo import MongoClient

connection_string = os.getenv('MONGO_ATLAS_URI')
client = MongoClient(connection_string)
db = client["sample_mflix"]
```

Then

- Retrieve the title and genres of movies that have the "Action" genre.

```Python
cursor = db.movies.find(
    {"genres": "Action"},  # Filter: movies with 'Action' in the genres array
    {"_id": 0, "title": 1, "genres": 1}  # Projection: include title and genres,
)
```

to see the results with python

```Python
for movie in cursor:
    print(movie)
```

**with `mongosh`:**

once connected

```JavaScript
cursor = db.movies.find(
    {"genres": "Action"},  # Filter: movies with 'Action' in the genres array
    {"_id": 0, "title": 1, "genres": 1}  # Projection: include title and genres,
)
```

### Number of documents

The fastest way to count documents is to use `count_documents`

```Python
count = db.movies.count_documents({"imdb.rating": {"$gt": 8.0}})
```

> Note: in Python, you can clone the cursor to get its length and the number of returned documents. Cloning the cursor does not consume it

```Python
len(list(cursor.clone()))
```

**with mongosh**

```JavaScript
db.movies.countDocuments({ "imdb.rating": { $gt: 8.0 } })
```

### Your turn

with

```Python
cursor = db.movies.find( filter, projection).limit(5)
for movie in cursor:
    print(movie)
```

Write the filter and projection for the following queries and also return the number of documents with `db.movies.count_documents(filter)`

- use projection to return only relevant fields or at minimum "title"
- limit results to 5 documents

1. Find movies with an IMDb rating greater than 8
   - filter: `{"imdb.rating": {"$gt" : 8}}`
   - projection: `{"_id": 0, "title": 1, "imdb.rating": 1}`
2. Movies released after 2000
3. Movies with a specific director: "Christopher Nolan". Show title, director and year
4. Retrieve movies with `tomatoes.viewer.rating > 4.0`, showing title and viewer rating.
5. Find movies that contain `"Comedy"` and `"Drama"` in the `genres` array. Use `{$all: [list of genres]}`
6. Combine a query with sorting: Retrieve the top 5 movies with the highest IMDb rating, showing title and rating. (you should only retrieve `imdb.rating` with `double` data type)
7. Movie query on a year range: Retrieve movies released between 1990 and 2000, showing title and year.
8. Movie query with missing fields: Find movies where the `fullplot` field does not exist. Use `$exists`.
9. Find all distinct genres
   - use `db.movies.distinct("genres")`
10. Movies with at least 2 genres
    - use `{"genres": {$size: 2}}`
11. Action movies, after 1950 with imdb ratings > 8, sort by year desc, imdb rating desc
    - use `{ "year": {$gt: 1950}, "imdb.rating": {$gt: 8}, "genres": "Action" }`
12. Movies with both genres: Action and Drama
    - `$and: [ {"genres": "Action"}, {"genres": "Drama"} ]`
13. Movies with either Action or Drama
    - `$or: [ {"genres": "Action"}, {"genres": "Drama"} ]`
14. Movies after 1950 with either imdb.rating > 0 or awards.wins > 5
    - use: `{ "year": {$gt: 1950}, $or: [ {"imdb.rating": {$gt: 8}}, {"awards.wins": {$gt: 5}} ] }`

---

## Conclusion

In this session you practiced:

- Setting up MongoDB Atlas, a cloud-hosted database service, including cluster creation and security configuration

- Writing basic MongoDB queries using JSON format:
  - Basic syntax: `{}` for all documents, `{field: value}` for equality, `{field: {$lt: value}}` for comparisons
  - How to query nested fields and arrays in complex documents
- Connecting to MongoDB Atlas using Python and `pymongo`:
  - Using basic operations: `find()`, `find_one()`, `distinct()`, `count_documents()`
  - Implementing projections to select specific fields
- Working with sample datasets (particularly the movies database) to practice:
  - Filtering and sorting data
  - Working with nested fields
  - Using operators like `$gt`, `$lt`, `$all`, `$exists`
  - Writing combined queries with multiple conditions

In the next session, we will dive deeper into MongoDB and look at more complex ways to query data using **aggregation pipelines**. We will also cover schema design and validation.

## To go further

For next time, you can:

- explore more Atlas sample databases and practice writing queries
- follow https://www.mongodb.com/docs/languages/python/pymongo-driver/current/read/ for more practice
- There are many free courses and tutorials in MongoDB University
  - Intro to MongoDB
  - CRUD in python

and much more