mongoDB® Atlas User Guide

Table of Contents

Introduction

What is Atlas?

MongoDB Atlas is a fully-managed cloud database developed by the same people that build MongoDB. Atlas handles all the complexity of deploying, managing, and healing your deployments on the cloud service provider of your choice (AWS , Azure, and GCP).

What this guide covers?

This guide focuses on the simple basic operations needed to create and manage an Atlas cluster, and covers the most important MQL (MongoDB Query Language) commands required to manipulate data in MongoDB.

Chapter 1

Get Started With Atlas

Create an Atlas Account

1. Register new Atlas account at : *https://account.mongodb.com/account/register*
2. Login to Your Atlas Account
3. Create an Atlas organization and then create a project in this organization. You will deploy your first cluster in this project.

Deploy a Free Tier Cluster

Atlas Free Tier clusters provide a small-scale development environment to host your data. Free Tier clusters never expire, and provide access to a subset of Atlas features and functionality.

1. Click Build a Cluster
2. Select Starter Clusters and click create a Cluster.
3. Select your preferred Cloud Provider & Region.
4. Select M0 Sandbox for cluster tier
5. Enter a name for your cluster in the Cluster Name field
6. Click Create Cluster to deploy the cluster

Add your connection IP

You must add your IP address to the IP access list before you can connect to your cluster. To add your IP address to the IP access list

1. Click Connect
2. Click Add Your Current IP Address
3. Click Add IP Address

Create a Database User for Your Cluster

You must create a database user to access your cluster. For security purposes, Atlas requires clients to authenticate as MongoDB database users to access clusters.
To add a database user to your cluster:

4. Click Connect
5. Set the new user's Username and Password
6. Click Create Database User

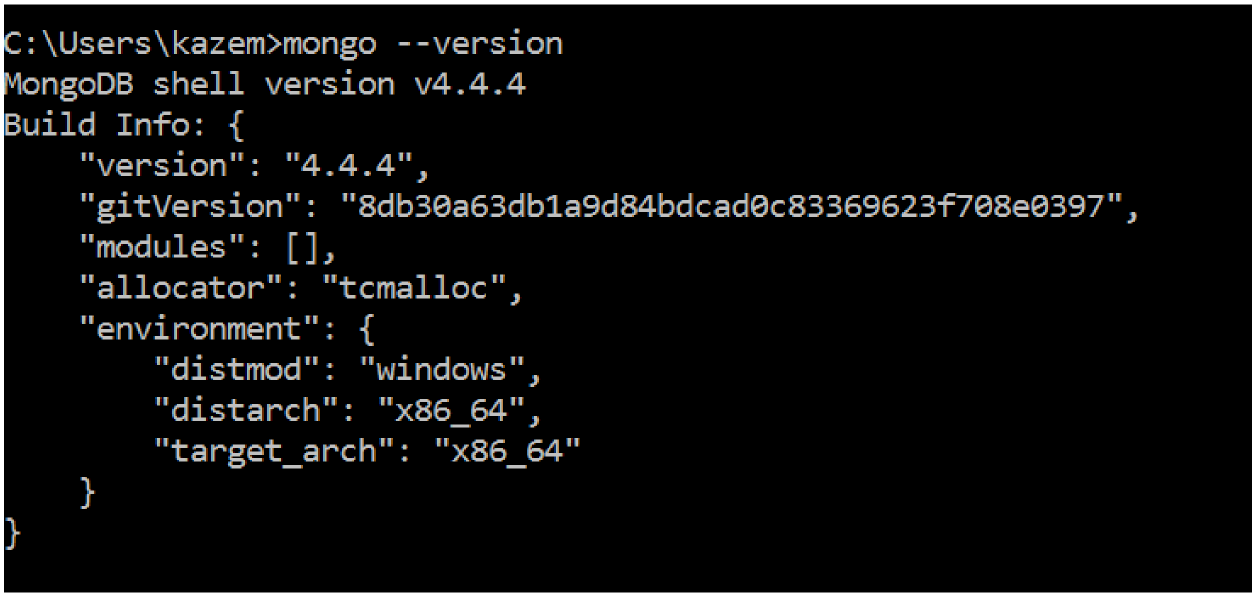
Connect to Your Cluster

You can connect to your cluster in a variety of ways. This tutorial describes how to connect to your cluster using the mongo shell. However, there are also other ways to connect to your cluster either, by using programming languages, like c, c++, c#, Java, python, php, ruby, etc.. Or by using mongoDB GUI’s tool mongoDB compas

1. Click connect
2. Click Choose a connection method
3. Click Connect with mongo shell
4. Click I do not have the mongo shell installed.
5. Download the mongo shell.
6. Add the mongo shell to your system path
7. Test your mongo shell installation, by running this command:

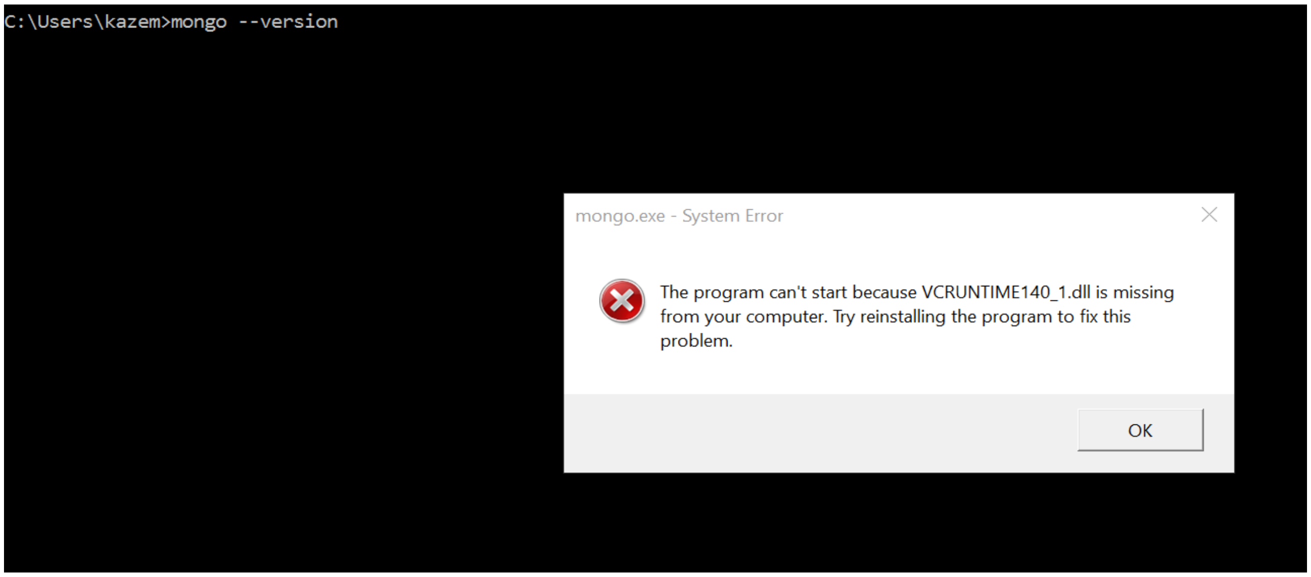
   mongo –version

You should see an output similar to the following:



```
C:\Users\kazem>mongo --version
MongoDB shell version v4.4.4
Build Info: {
    "version": "4.4.4",
    "gitVersion": "8db30a63db1a9d84bdcad0c83369623f708e0397",
    "modules": [],
    "allocator": "tcmalloc",
    "environment": {
        "distmod": "windows",
        "distarch": "x86_64",
        "target_arch": "x86_64"
    }
}
```

Note: in case an error message appears saying "VCRUNTIME140_1.dll" is missing. You can downloaded it from this link https://www.dll-files.com/vcruntime140_1.dll.html and move the file to the bin folder of the mongoDB shell



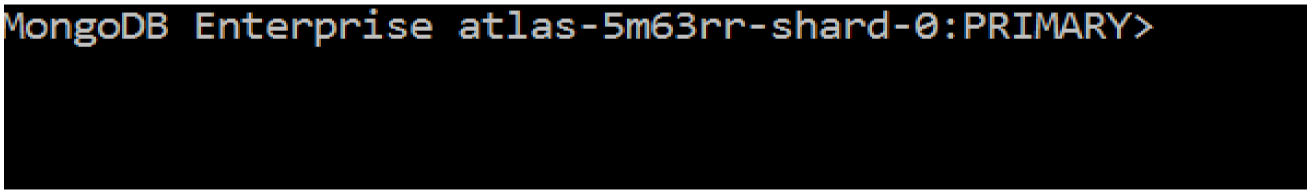
```
C:\Users\kazem>mongo --version
```

mongo.exe - System Error

The program can't start because VCRUNTIME140_1.dll is missing from your computer. Try reinstalling the program to fix this problem.

8. Copy the provided connection string to your clipboard
9. Paste and run your connection string in your terminal then enter your password

You should see the following:



```
MongoDB Enterprise atlas-5m63rr-shard-0:PRIMARY>
```

Chapter 2

Interact with Cluster Data

Create a Database

Use DATABASE_NAME is used to create database. The command will create a new database if it doesn't exist, otherwise it will return the existing database.

```
>use mydb
switched to db mydb
```

To check your currently selected database, use the command db

```
>db
Mydb
```

If you want to check your databases list, use the command

show dbs.

```
>show dbs
local     0.78125GB
test      0.23012GB
```

Your created database (mydb) is not present in list. To display database, you need to insert at least one document into it.

```
>db.movie.insert({"name":"Star Wars"})
>show dbs
local      0.78125GB
mydb       0.23012GB
test       0.23012GB
```

In MongoDB default database is test. If you didn't create any database, then collections will be stored in test database.

Drop a Database

db.dropDatabase() command is used to drop a existing database.

This will delete the selected database. If you have not selected any database, then it will delete default 'test' database

Example

First, check the list of available databases by using the command, show dbs.

```
>show dbs
local      0.78125GB
mydb       0.23012GB
test       0.23012GB
```

If you want to delete new database <mydb>, then dropDatabase() command would be as follows

```
>use mydb
switched to db mydb
>db.dropDatabase()
>{ "dropped" : "mydb", "ok" : 1 }
>
```

Now check list of databases.

```
>show dbs
local      0.78125GB
test       0.23012GB
>
```

Create a collection

db.createCollection(name, options) is used to create collection.

In the command, name is name of collection to be created. Options is a document and is used to specify configuration of collection (Specify options about memory size and indexing).

Options parameter is optional, so you need to specify only the name of the collection. Following is the list of options you can use

Field	Type	Description
capped	Boolean	(Optional) If true, enables a capped collection. Capped collection is a fixed size collection that automatically overwrites its oldest entries when it reaches its maximum size. If you specify true, you need to specify size parameter also.
autoIndexId	Boolean	(Optional) If true, automatically create index on _id field.s Default value is false.
size	number	(Optional) Specifies a maximum size in bytes for a capped collection. If capped is true, then you need to specify this field also.
max	number	(Optional) Specifies the maximum number of documents allowed in the capped collection.

Examples

```
>use test
switched to db test
>db.createCollection("mycollection")
{ "ok" : 1 }
>
```

You can check the created collection by using the command

show collections.

```
>show collections
mycollection
system.indexes
```

The following example shows the syntax of createCollection() method with few important options

```
> db.createCollection("mycol", { capped : true, autoIndexID : true, size :
6142800, max : 10000 } ){
"ok" : 0,
"errmsg" : "BSON field 'create.autoIndexID' is an unknown field.",
"code" : 40415,
"codeName" : "Location40415"
}
>
```

In MongoDB, you don't need to create collection. MongoDB creates collection automatically, when you insert some document.

```
>db.newCollection.insert({"name" : "newDocument"})
WriteResult({ "nInserted" : 1 })
>show collections
mycol
mycollection
system.indexes
newCollection
>
```

Drop a Collection

db.collection.drop() is used to drop a collection from the database.

Example

First, check the available collections into your database mydb.

```
>use mydb
switched to db mydb
>show collections
mycol
mycollection
system.indexes
newCollection
>
```

Now drop the collection with the name mycollection.

```
>db.mycollection.drop()
true
>
```

drop() method will return true, if the selected collection is dropped successfully, otherwise it will return false.

Data Types

MongoDB supports many datatypes. Some of them are:

- String: String in MongoDB must be UTF-8 valid.
- Integer: Integer can be 32 bit or 64 bit depending upon your server.
- Boolean: used to store a Boolean (true/ false) value.
- Double: used to store floating point values.

- Min/ Max keys: used to compare a value against the lowest and highest BSON elements.
- Arrays: used to store arrays or list or multiple values into one key.
- Timestamp: ctimestamp. This can be handy for recording when a document has been modified or added.
- Object: used for embedded documents.
- Null: is used to store a Null value.
- Symbol: used identically to a string; however, it's generally reserved for languages that use a specific symbol type.
- Date: used to store the current date or time in UNIX time format. You can specify your own date time by creating object of Date and passing day, month, year into it.
- Object ID: used to store the document’s ID.
- Binary data: used to store binary data.
- Code: used to store JavaScript code into the document.
- Regular expression: used to store regular expression.

Insert Document

To insert data into MongoDB collection, you need to use MongoDB's insert() or save() method.

Example

```
> db.users.insert({
... _id : ObjectId("507f191e810c19729de860ea"),
... title: "MongoDB Overview",
... description: "MongoDB is no sql database",
... tags: ['mongodb', 'database', 'NoSQL'],
... likes: 100
... })
WriteResult({ "nInserted" : 1 })
```

The insertOne() method

If you need to insert only one document into a collection you can use this method.

Example

Following example creates a new collection named empDetails and inserts a document using the insertOne() method.

```
> db.createCollection("empDetails")
{ "ok" : 1 }
> db.empDetails.insertOne(
        {
                First_Name: "John",
                Last_Name: "Doe",
                Date_Of_Birth: "1995-09-26",
                e_mail: " John _ Doe.123@gmail.com",
                phone: "9848022338"
        })
{
        "acknowledged" : true,
        "insertedId" : ObjectId("5dd62b4070fb13eec3963bea")
}
>
```

The insertMany() method

you can insert multiple documents using the insertMany() method. To this method you need to pass an array of documents.

Example

Following example inserts three different documents into the empDetails collection using the insertMany() method.

```
> db.empDetails.insertMany(
	[
		{
			First_Name: "Radhika",
			Last_Name: "Sharma",
			Date_Of_Birth: "1995-09-26",
			e_mail: "radhika_sharma.123@gmail.com",
			phone: "9000012345"
		},
		{
			First_Name: "Rachel",
			Last_Name: "Christopher",
			Date_Of_Birth: "1990-02-16",
			e_mail: "Rachel_Christopher.123@gmail.com",
			phone: "9000054321"
		},
		{
			First_Name: "Fathima",
			Last_Name: "Sheik",
			Date_Of_Birth: "1990-02-16",
			e_mail: "Fathima_Sheik.123@gmail.com",
			phone: "9000054321"
		}
	]
)
{
	"acknowledged" : true,
	"insertedIds" : [
		ObjectId("5dd631f270fb13eec3963bed"),
		ObjectId("5dd631f270fb13eec3963bee"),
		ObjectId("5dd631f270fb13eec3963bef")
	]
}
>
```

Query Document

The find() Method

To query data from MongoDB collection, you need to use MongoDB's find() method.

Example

```
> db.users.find()
{ "_id" : ObjectId("507f191e810c19729de860ea"), "title" : "MongoDB Overview",
"description" : "MongoDB is no sql database", "tags" : [ "mongodb",
"database", "NoSQL" ], "likes" : 100 }
>
```

The pretty() Method

Example

Following example retrieves all the documents from the collection named mycol and arranges them in an easy-to-read format.

```
> db.users.find().pretty()
{
        "_id" : ObjectId("507f191e810c19729de860ea"),
        "title" : "MongoDB Overview",
        "description" : "MongoDB is no sql database",
        "tags" : [
                "mongodb",
                "database",
                "NoSQL"
        ],
        "likes" : 100
}
```

The findOne() method

Example

Following example retrieves the document with title MongoDB Overview.

```
> db.mycol.findOne({title: "MongoDB Overview"})
{
        "_id" : ObjectId("5dd6542170fb13eec3963bf0"),
        "title" : "MongoDB Overview",
        "description" : "MongoDB is no SQL database",
        "tags" : [
                "mongodb",
                "database",
                "NoSQL"
        ],
        "likes" : 100
}
```

AND in MongoDB

Example

Following example will show all the tutorials whose description 'MongoDB is no SQL database' and whose title is 'MongoDB Overview'.

```
> db.mycol.find({$and:[{ "description": "MongoDB is no SQL
database"},{"title": "MongoDB Overview"}]}).pretty()
{
        "_id" : ObjectId("5dd4e2cc0821d3b44607534c"),
        "title" : "MongoDB Overview",
        "description" : "MongoDB is no SQL database",
        "tags" : [
                "mongodb",
                "database",
                "NoSQL"
        ],
        "likes" : 100
}
>
```

OR in MongoDB

Example

Following example will show all the tutorials whose description 'MongoDB is no SQL database' or whose title is 'MongoDB Overview'.

```
> db.mycol.find({$or:[{ “description”: "MongoDB is no SQL
database"},{"title": "MongoDB Overview"}]}).pretty()
{
   "_id": ObjectId(7df78ad8902c),
   "title": "MongoDB Overview",
   "description": "MongoDB is no sql database",
   "tags": ["mongodb", "database", "NoSQL"],
   "likes": "100"
}
```

NOT in MongoDB

Example

Following example will retrieve the document(s) whose age is not greater than 25

```
> db.empDetails.find( { "Age": { $not: { $gt: "25" } } } )
{
        "_id" : ObjectId("5dd6636870fb13eec3963bf7"),
        "First_Name" : "Fathima",
        "Last_Name" : "Sheik",
        "Age" : "24",
        "e_mail" : "Fathima_Sheik.123@gmail.com",
        "phone" : "9000054321"
}
```

Update Document

MongoDB Update() Method

Example

Consider the mycol collection has the following data.

```
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Atlas overview"}
```

Following example will set the new title 'New MongoDB Tutorial' of the documents whose title is 'MongoDB Overview'.

```
>db.mycol.update({'title':'MongoDB Overview'},{$set:{'title':'New MongoDB
Tutorial'}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>db.mycol.find()
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"New MongoDB Tutorial"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Atlas overview"}
>
```

By default, MongoDB will update only a single document. To update multiple documents, you need to set a parameter 'multi' to true.

```
>db.mycol.update({'title':'MongoDB Overview'},
   {$set:{'title':'New MongoDB Tutorial'}},{multi:true})
```

MongoDB updateOne() method

Example

```
> db.empDetails.updateOne(
        {First_Name: 'Radhika'},
        { $set: { Age: '30',e_mail: 'radhika_newemail@gmail.com'}}
)
```

```
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 0 }
>
```

MongoDB updateMany() method

Example

```
> db.empDetails.updateMany(
      {Age:{ $gt: "25" }},
      { $set: { Age: '00'}}
)
{ "acknowledged" : true, "matchedCount" : 2, "modifiedCount" : 2 }
```

You can see the updated values if you retrieve the contents of the document using the find method as shown below:

```
> db.empDetails.find()
{ "_id" : ObjectId("5dd6636870fb13eec3963bf5"), "First_Name" : "Radhika",
"Last_Name" : "Sharma", "Age" : "00", "e_mail" :
"radhika_newemail@gmail.com", "phone" : "9000012345" }
{ "_id" : ObjectId("5dd6636870fb13eec3963bf6"), "First_Name" : "Rachel",
"Last_Name" : "Christopher", "Age" : "00", "e_mail" :
"Rachel_Christopher.123@gmail.com", "phone" : "9000054321" }
{ "_id" : ObjectId("5dd6636870fb13eec3963bf7"), "First_Name" : "Fathima",
"Last_Name" : "Sheik", "Age" : "24", "e_mail" :
"Fathima_Sheik.123@gmail.com", "phone" : "9000054321" }
>
```

Delete Document

The remove() Method

Example

Consider the mycol collection has the following data.

```
{_id : ObjectId("507f191e810c19729de860e1"), title: "MongoDB Overview"},
{_id : ObjectId("507f191e810c19729de860e2"), title: "NoSQL Overview"},
{_id : ObjectId("507f191e810c19729de860e3"), title: "Atlas overview"}
```

Following example will remove all the documents whose title is 'MongoDB Overview'.

```
>db.mycol.remove({'title':'MongoDB Overview'})
WriteResult({"nRemoved" : 1})
> db.mycol.find()
{"_id" : ObjectId("507f191e810c19729de860e2"), "title" : "NoSQL Overview" }
{"_id" : ObjectId("507f191e810c19729de860e3"), "title" : "Atlas overview}
```

Remove Only One

If there are multiple records and you want to delete only the first record, then set justOne parameter in remove() method.

```
>db.COLLECTION_NAME.remove(DELETION_CRITERIA,1)
```

Remove All Documents

If you don't specify deletion criteria, then MongoDB will delete whole documents from the collection.

```
> db.mycol.remove({})
WriteResult({ "nRemoved" : 2 })
> db.mycol.find()
>
```

Projection

In MongoDB, projection means selecting only the necessary data rather than selecting whole of the data of a document. If a document has 5 fields and you need to show only 3, then select only 3 fields from them

Following example will display the title of the document while querying the document.

```
>db.mycol.find({},{"title":1,_id:0})
```

```
{"title":"MongoDB Overview"}
{"title":"NoSQL Overview"}
{"title":"Tutorials Point Overview"}
>
```

Please note _id field is always displayed while executing find() method, if you don't want this field, then you need to set it as 0.

Limiting Records

The Limit() Method

To limit the records in MongoDB, you need to use limit() method. The method accepts one number type argument, which is the number of documents that you want to be displayed.

Example

Consider the collection mycol has the following data.

```
{_id : ObjectId("507f191e810c19729de860e1"), title: "MongoDB Overview"},
{_id : ObjectId("507f191e810c19729de860e2"), title: "NoSQL Overview"},
{_id : ObjectId("507f191e810c19729de860e3"), title: "Atlas Overview"}
```

Following example will display only two documents while querying the document.

```
>db.mycol.find({},{"title":1,_id:0}).limit(2)
{"title":"MongoDB Overview"}
{"title":"NoSQL Overview"}
>
```

MongoDB Skip() Method

Apart from limit() method, there is one more method skip() which also accepts number type argument and is used to skip the number of documents

Example

Following example will display only the second document.

```
>db.mycol.find({},{"title":1,_id:0}).limit(1).skip(1)
{"title":"NoSQL Overview"}
>
```

Sorting Records

The sort() Method

To sort documents in MongoDB, you need to use sort() method. The method accepts a document containing a list of fields along with their sorting order. To specify sorting order 1 and -1 are used. 1 is used for ascending order while -1 is used for descending order.

Example

Consider the collection mycol has the following data.

```
{_id : ObjectId("507f191e810c19729de860e1"), title: "MongoDB Overview"}
{_id : ObjectId("507f191e810c19729de860e2"), title: "NoSQL Overview"}
{_id : ObjectId("507f191e810c19729de860e3"), title: "Atlas Overview"}
```

Following example will display the documents sorted by title in the descending order.

```
>db.mycol.find({},{"title":1,_id:0}).sort({"title":-1})
```

```
{"title":"Atlas Overview"}
{"title":"NoSQL Overview"}
{"title":"MongoDB Overview"}
>
```

References

[1] "mongoDB university," [Online]. Available: https://university.mongodb.com/. [Accessed 08 March 2021].

[2] "MongoDB Documentation," [Online]. Available: https://docs.atlas.mongodb.com. [Accessed 07 March 2021].

[3] "Tutorials Point," [Online]. Available: https://www.tutorialspoint.com/mongodb/mongodb_create_database.htm. [Accessed 09 March 2021].