Getting Started with Python and MongoDB

Get Started with MongoDB Atlas

Robert Walters

April 27, 2017 | Updated: June 24, 2021
#Technical

You can get started with MongoDB and your favorite programming language by leveraging one of its drivers, many of which are maintained by MongoDB engineers, and others which are maintained by members of the community. MongoDB has a native Python driver, PyMongo, and a team of Driver engineers dedicated to making the driver fit the Python community's needs, making sure MongoDB and Python work together flawlessly.

In this article, which is aimed at Python developers who are new to MongoDB, you will learn how to do the following:

- Create a free hosted MongoDB database using MongoDB Atlas
- Install PyMongo, the Python Driver
- Connect to MongoDB
- Explore MongoDB Collections and Documents
- Perform basic Create, Retrieve, Update and Delete (CRUD) operations using PyMongo

Let's get started!

You can start working immediately with MongoD by using a free MongoDB cluster via MongoDB Atlas. MongoDB Atlas is a hosted database s llows you to choose your database size and get a connection string! If you are interested i free tier follow the instructions in the Appendix section at the end of this article.

Install the Python Driver

For this article, we will install the Python driver called, "PyMongo".

Although there are other drivers written by the community, PyMongo is the official Python driver for MongoDB. For a detailed documentation on the driver check out the documentation here.

The easiest way to install the driver is through the pip package management system. Execute the following on a command line:

```
python -m pip install pymongo
```

Note: If you are using the Atlas M0 (Free Tier) cluster, you must use Python 2.7.9+ and use a Python 3.4 or newer. You can check which version of Python and PyMongo you have installed by issuing "`python --version`" and "`pip list`" commands respectively.

For variations of driver installation check out the complete documentation:

Once PyMongo is installed we can write our first application that will return information about the MongoDB server. In your Python development environment or from a text editor enter the following code.

```
from pymongo import MongoClient
# pprint library is used to make the output look more pretty
from pprint import pprint
# connect to MongoDB, change the << MONGODB URL >> to reflect your own connection string
client = MongoClient(<<MONGODB URL>>)
db=client.admin
# Issue the serverStatus command and print the results
serverStatusResult=db.command("serverStatus")
pprint(serverStatusResult)
```

Replace the "< >" with your connection string to MongoDB. Save this file as "mongodbtest.py" and run it from the command line via, "python mongodbtest.py"

An example output appears as follows:

```
{u'asserts': {u'msg': 0,
              u'regular': 0,
              u'rollovers': 0,
              u'user': 0,
              u'warning': 0},
 u'connections': {u'available': 96, u'[illegible], u'totalCreated': 174L},
 u'extra_info': {u'note': u'fields var[illegible]rm', u'page_faults': 0},
 u'host': u'cluster0-shard-00-00-6czvq.m[illegible].net:27017',
 u'localTime': datetime.datetime(2017, 4, 4, 0, 18, 45, 616000),
.
.
.
}
```

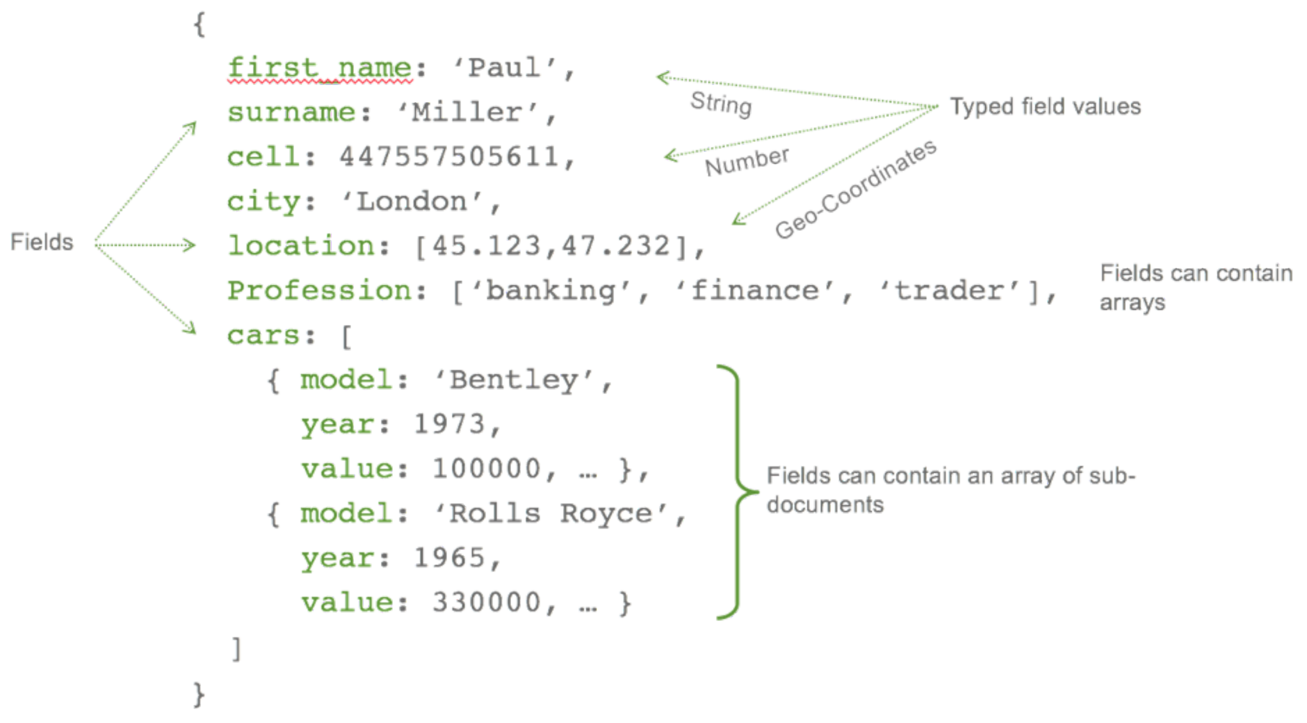
Note that the **'u'** character comes from the python output and it means that the strings are stored in unicode. This example also uses the `pprint` library which is not related to MongoDB but is used here only to make the output structured and visually appealing from a console.

In this example we are connecting to our MongoDB instance and issuing the "db.serverStatus()" command (reference). This command returns information about our MongoDB instance and is used in this example as a way to execute a command against MongoDB.

If your application runs successfully, you are ready to continue!

Exploring Collections and Documents

MongoDB stores data in documents. Documents are not like Microsoft Word or Adode PDF documents but rather JSON documents based on the JSON specification.
An example of a JSON document would be as follows:



```
{
  first_name: 'Paul',
  surname: 'Miller',
  cell: 447557505611,
  city: 'London',
  location: [45.123,47.232],
  Profession: ['banking', 'finance', 'trader'],
  cars: [
    { model: 'Bentley',
      year: 1973,
      value: 100000, … },
    { model: 'Rolls Royce',
      year: 1965,
      value: 330000, … }
  ]
}
```

Fields
String
Number
Geo-Coordinates
Typed field values
Fields can contain arrays
Fields can contain an array of sub-documents

Figure 1: Sample document

Notice that documents are not just key/value pairs but can include arrays and subdocuments. The data itself can be different data types like geospatial, decimal, and ISODate to name a few. Internally

MongoDB stores a binary representation of JSO[illegible]wn as BSON. This allows MongoDB to provide data types like decimal that are not defined i[illegible]pecification. For more information on the BSON spec check out the following URL: http[illegible]ec.org.

A collection in MongoDB is a container for documents. A database is the container for collections. This grouping is similar to relational databases and is pictured below:

Relational concept	MongoDB equivalent
Database	Database
Tables	Collections
Rows	Documents
Index	Index

There are many advantages to storing data in documents. While a deeper discussion is out of the scope of this article, some of the advantages like dynamic, flexible schema, and the ability to store arrays can be seen from our simple Python scripts. For more information on MongoDB document structure take a look at the online documentation at the following URL: https://docs.mongodb.com/manual/core/document/.

Let's take a look at how to perform basic CRUD operations on documents in MongoDB using PyMongo.

Performing basic CRUD operations using PyMongo

To establish a connection to MongoDB with PyMongo you use the MongoClient class.

```
from pymongo import MongoClient
client = MongoClient('<<MongoDB URL>>')
```

The "`<<MongoDB URL>>`"is a placeholder for the connection string to MongoDB. See the connection string documentation for detail information on how to create your MongoDB connection string. If you are using Atlas for your MongoDB database, refer to the "testing your connection" section for more information on obtaining the connection string for MongoDB Atlas.

We can now create a database object referencing a new database, called "`business`", as follows:

```
db = client.business
```

Once we create this object we can perform our CRUD operations. Since we want something useful to query let's start by building a sample data ge[illegible]cation.

Generating sample data code example

Create a new file called createsamples.py using your development tool or command line text editor and copy the following code:

```
from pymongo import MongoClient
from random import randint
#Step 1: Connect to MongoDB - Note: Change connection string as needed
client = MongoClient(port=27017)
db=client.business
#Step 2: Create sample data
names = ['Kitchen','Animal','State', 'Tastey', 'Big','City','Fish', 'Pizza','Goat', 'Sal
company_type = ['LLC','Inc','Company','Corporation']
company_cuisine = ['Pizza', 'Bar Food', 'Fast Food', 'Italian', 'Mexican', 'American', '
for x in range(1, 501):
    business = {
        'name' : names[randint(0, (len(names)-1))] + ' ' + names[randint(0, (len(names)-
        'rating' : randint(1, 5),
        'cuisine' : company_cuisine[randint(0, (len(company_cuisine)-1))]
    }
    #Step 3: Insert business object directly into MongoDB via insert_one
    result=db.reviews.insert_one(business)
    #Step 4: Print to the console the ObjectID of the new document
    print('Created {0} of 500 as {1}'.format(x,result.inserted_id))
#Step 5: Tell us that you are done
print('finished creating 500 business reviews')
```

Be sure to change the MongoDB client connection URL to one that points to your MongoDB database instance. Once you run this application, 500 randomly named businesses with their corresponding ratings will be created in the MongoDB database called, "business". All of these businesses are created in a single collection called, "reviews". Notice that we do not have to explicitly create a database beforehand in order to use it. This is different from other databases that require statements like, "CREATE DATABASE" to be performed first.

The command that inserts data into MongoDB in this example is the insert_one() function. A bit self-explanatory, `insert_one` will insert one document into MongoDB. The result set will return the single ObjectID that was created. This is one of a few methods that insert data. If you wanted to insert multiple documents in one call you can use the `insert_many` function. In addition to an acknowledgement of the insertion, the result set for insert_many will include a list of the ObjectIDs that were created. For more information on `insert_many` see the documentation located here.

For details on the result set of `insert_many` che[illegible] this section of documentation as well.

We are now ready to explore querying and ma[illegible]a in MongoDB using Python. To guide this exploration we will create another application that will manage our business reviews.

Exploring business review data

Now that we have a good set of data in our database let's query for some results using PyMongo.

In MongoDB the find_one command is used to query for a single document much like select statements are used in relational databases. To use the find_one command in PyMongo we pass a Python dictionary that specifies the search criteria. For example, let's find a single business with a review score of 5 by passing the dictionary, "`{ 'rating' : 5 }` ".

```
fivestar = db.reviews.find_one({'rating': 5})
print(fivestar)

The result will contain data similar to the following:

{u'rating': 5,
 u'_id': ObjectId('58e65383ea0b650c867ef195'),
 u'name': u'Fish Salty Corporation',
u'cuisine': u'Sushi Bar'}
```

Given we created 500 sample pieces of data there is more than one business with rating 5. The `find_one` method is just one in a series of find statements that support querying MongoDB data. Another statement, called "`find`", will return a cursor over all documents that match the search criteria. These cursors also support methods like count() which returns the number of results in the query. To find the total count of businesses that are rated with a 5 we can use the `count()` method as follows:

```
fivestarcount = db.reviews.find({'rating': 5}).count()
print(fivestarcount)
```

Your results may vary since the data was randomly generated but in a test run the value of 103 was returned.

MongoDB can easily perform these straightforward queries. However, consider the scenario where you want to sum the occurrence of each rating across the entire data set. In MongoDB you could create 5 separate find queries, execute them and present the results, or you could simply issue a single query using the MongoDB aggregation pipeline as follows:

```
from pymongo import MongoClient
# Connect to the MongoDB, change the connection string per your MongoDB environment
client = MongoClient(port=27017)
```

```
# Set the db object to point to the business database
db=client.business
# Showcasing the count() method of fin[illegible]e total number of 5 ratings
print('The number of 5 star reviews:')
fivestarcount = db.reviews.find({'rating': 5}).count()
print(fivestarcount)
# Now let's use the aggregation framework to sum the occurrence of each rating across the
print('\nThe sum of each rating occurance across all data grouped by rating ')
stargroup=db.reviews.aggregate(
# The Aggregation Pipeline is defined as an array of different operations
[
# The first stage in this pipe is to group data
{ '$group':
    { '_id': "$rating",
     "count" :
                 { '$sum' :1 }
    }
},
# The second stage in this pipe is to sort the data
{"$sort":  { "_id":1}
}
# Close the array with the ] tag
] )
# Print the result
for group in stargroup:
    print(group)
```

A deep dive into the aggregation framework is out of scope of this article, however, if you are interested in learning more about it check out the following URL: https://docs.mongodb.com/manual/aggregation/.

Updating data with PyMongo

Similar to `insert_one` and `insert_many` there exists functions to help you update your MongoDB data including `update_one`, `update_many` and `replace_one`. The `update_one` method will update a single document based on a query that matches a document. For example, let's assume that our business review application now has the ability for users to "like" a business. To illustrate updating a document with this new "likes" field, let's first take a look at what an existing document looks like from our previous application's insertion into MongoDB. Next, let's update the document and requery the document and see the change.

```
from pymongo import MongoClient
#include pprint for readabillity of the
from pprint import pprint

#change the MongoClient connection string to your MongoDB database instance
client = MongoClient(port=27020)
```

```
db=client.business

ASingleReview = db.reviews.find_one({})
print('A sample document:')
pprint(ASingleReview)

result = db.reviews.update_one({'_id' : ASingleReview.get('_id') }, {'$inc': {'likes': 1}
print('Number of documents modified : ' + str(result.modified_count))

UpdatedDocument = db.reviews.find_one({'_id':ASingleReview.get('_id')})
print('The updated document:')
pprint(UpdatedDocument)
```

When running the sample code above you may see results similar to the following:

```
A sample document:
{'_id': ObjectId('58eba417ea0b6523b0fded4f'),
 'cuisine': 'Pizza',
 'name': 'Kitchen Goat Corporation',
 'rating': 1}

Number of documents modified : 1

The updated document:
{'_id': ObjectId('58eba417ea0b6523b0fded4f'),
 'cuisine': 'Pizza',
 'likes': 1,
 'name': 'Kitchen Goat Corporation',
 'rating': 1}
```

Notice that the original document did not have the “likes” field and an update allowed us to easily add the field to the document. This ability to dynamically add keys without the hassle of costly Alter_Table statements is the power of MongoDB’s flexible data model. It makes rapid application development a reality.

If you wanted to update all the fields of the document and keep the same ObjectID you will want to use the `replace_one` function. For more details on `replace_one` check out the pymongo documentation here.

The update functions also support an option called, “upsert”. With upsert you can tell MongoDB to create a new document if the document you are trying to update does not exist.

Deleting documents

Much like the other command discussed so far the delete_one and delete_many command takes a query that matches the document to delete a [illegible] rameter. For example, if you wanted to delete all documents in the reviews collection [illegible] category was "Bar Food" issue the following:

```
result = db.restaurants.delete_many({"category": "Bar Food"})
```

If you are deleting a large number of documents it may be more efficient to drop the collection instead of deleting all the documents.

Where to go next

There are lots of options when it comes to learning about MongoDB and Python. MongoDB University is a great place to start and learn about administration, development and other topics such as analytics with MongoDB. One course in particular is MongoDB for Developers (Python). This course covers the topics of this article in much more depth including a discussion on the MongoDB aggregation framework. For more information go to the following URL:

https://university.mongodb.com/courses/M101P/about

Appendix: Creating a free tier MongoDB Atlas database

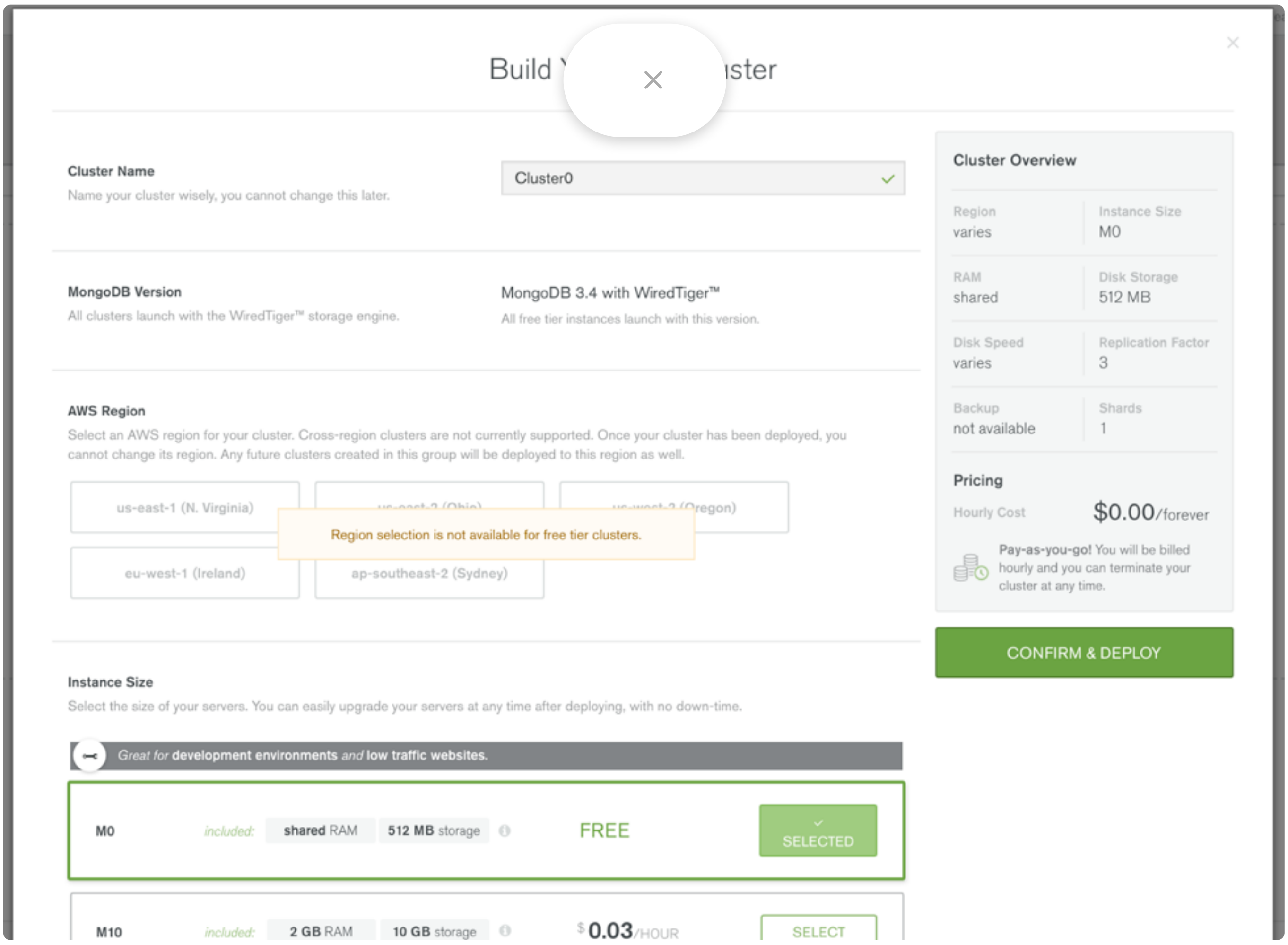
MongoDB Atlas is a hosted database service that allows you to choose your database size and get a connection string! Follow the steps below to start using your free

Build your cluster for free

Follow the below steps to create a free MongoDB database:

1. Go to the following URL: https://www.mongodb.com/cloud/atlas.
2. Click the "Start Free" button
3. Fill out the form to create an account. You will use this information to later login and manage your MongoDB.

Once you fill out the form, the website will create your account and you will be presented with the "Build Your New Cluster" pop up as shown in Figure 1.



Build ister

Cluster Name
Name your cluster wisely, you cannot change this later.

Cluster0 ✓

MongoDB Version
All clusters launch with the WiredTiger™ storage engine.

MongoDB 3.4 with WiredTiger™
All free tier instances launch with this version.

AWS Region
Select an AWS region for your cluster. Cross-region clusters are not currently supported. Once your cluster has been deployed, you cannot change its region. Any future clusters created in this group will be deployed to this region as well.

us-east-1 (N. Virginia) | us-east-2 (Ohio) | us-west-2 (Oregon)
eu-west-1 (Ireland) | ap-southeast-2 (Sydney)

Region selection is not available for free tier clusters.

Instance Size
Select the size of your servers. You can easily upgrade your servers at any time after deploying, with no down-time.

Great for **development environments** *and* **low traffic websites.**

M0	*included:*	**shared** RAM	**512 MB** storage	FREE	✓ SELECTED
M10	*included:*	**2 GB** RAM	**10 GB** storage	$0.03/HOUR	SELECT

Cluster Overview

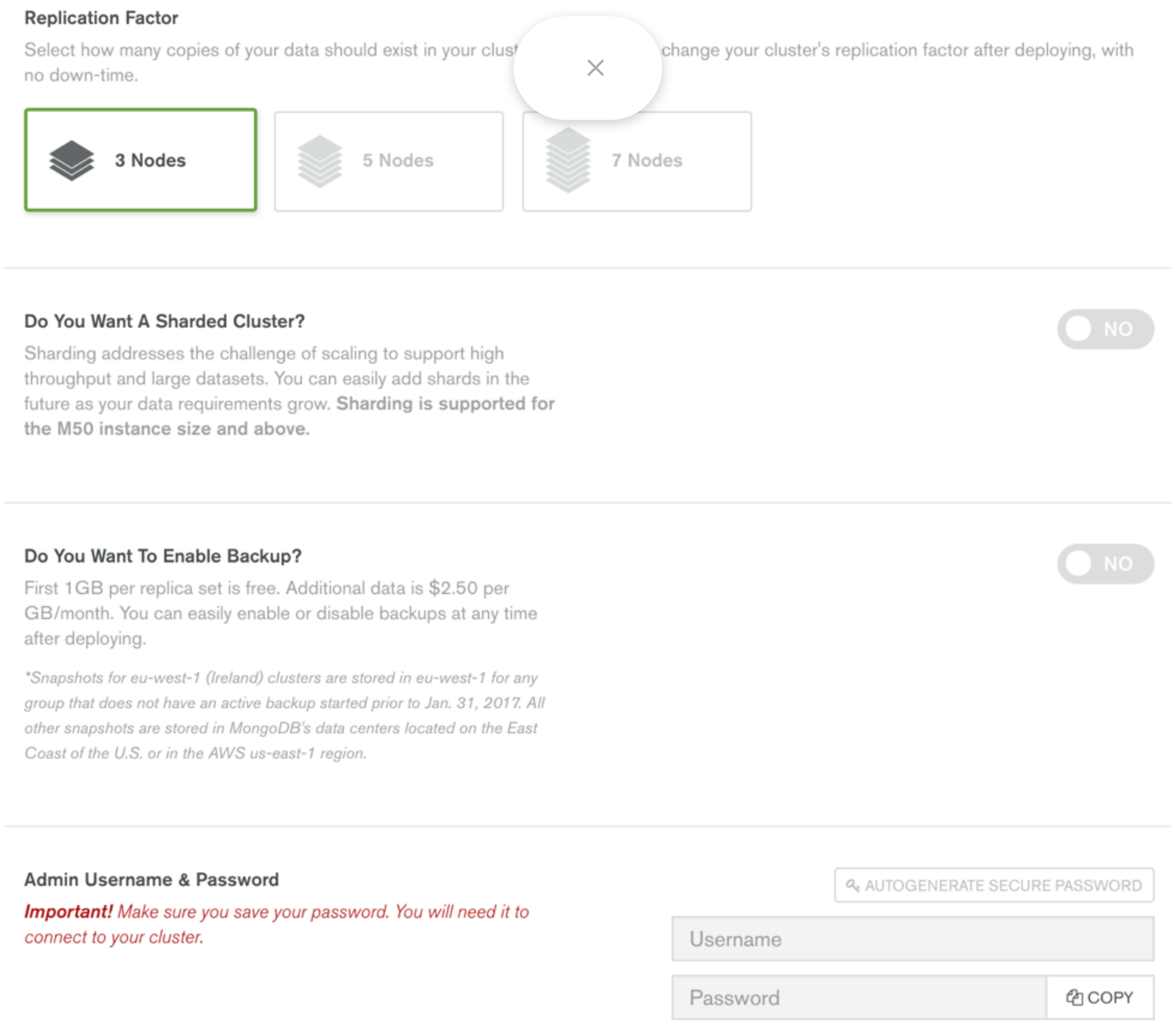
Region: varies	Instance Size: M0
RAM: shared	Disk Storage: 512 MB
Disk Speed: varies	Replication Factor: 3
Backup: not available	Shards: 1

Pricing
Hourly Cost $0.00/forever

Pay-as-you-go! You will be billed hourly and you can terminate your cluster at any time.

CONFIRM & DEPLOY

To use the free tier scroll down and select, “M0”. When you do this the regions panel will be disabled. The free tier has some restrictions with the ability to select a region being one of them and your database size will be limited to 512MB of storage. Given that, when you are ready to use MongoDB for more than just some simple operations you can easily create another instance by choosing a size from the “Instance Size” list. Before you click “Confirm & Deploy” scroll down the page and notice the additional options shown in Figure 2.


Replication Factor

Select how many copies of your data should exist in your clus change your cluster's replication factor after deploying, with no down-time.

3 Nodes | 5 Nodes | 7 Nodes

Do You Want A Sharded Cluster? NO

Sharding addresses the challenge of scaling to support high throughput and large datasets. You can easily add shards in the future as your data requirements grow. **Sharding is supported for the M50 instance size and above.**

Do You Want To Enable Backup? NO

First 1GB per replica set is free. Additional data is $2.50 per GB/month. You can easily enable or disable backups at any time after deploying.

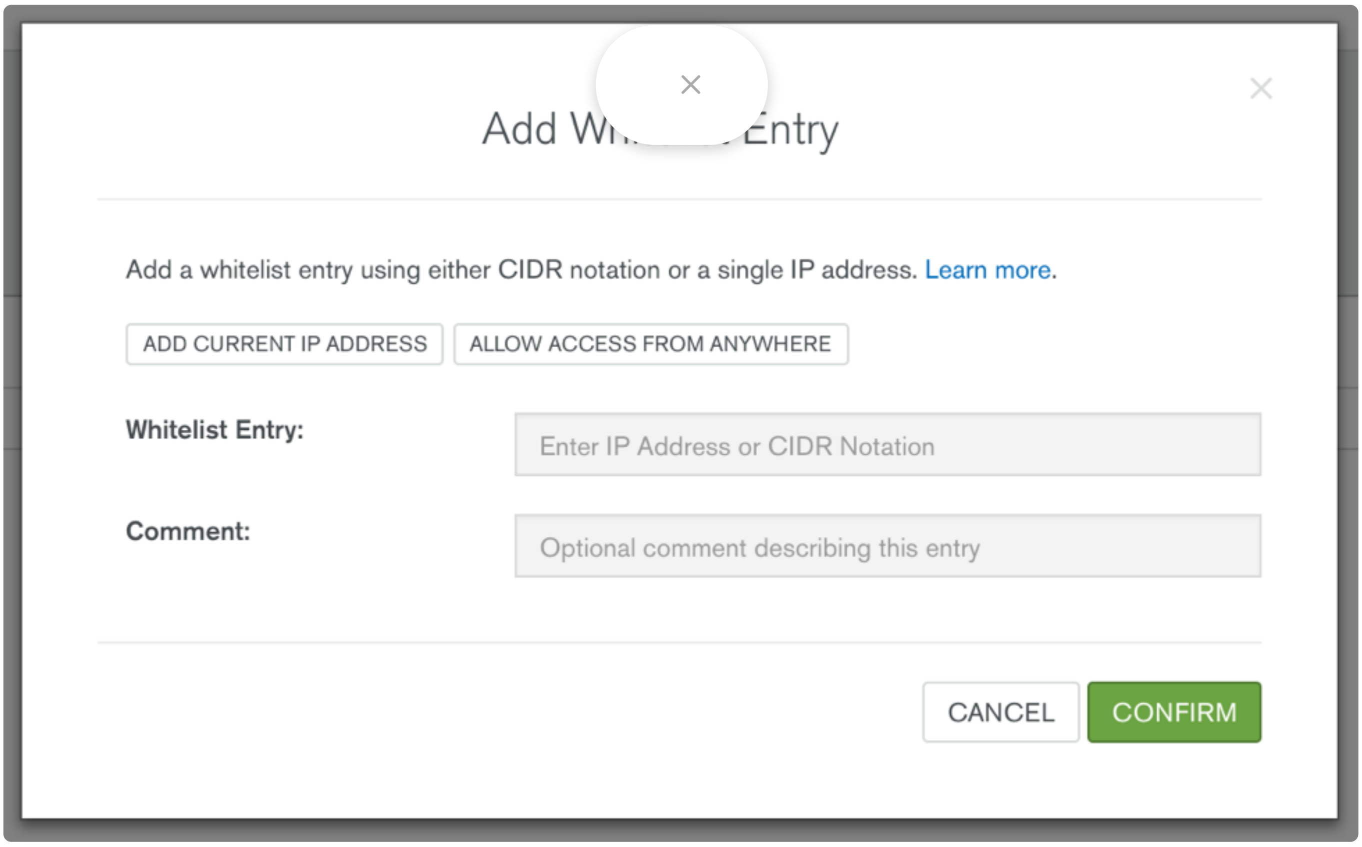
**Snapshots for eu-west-1 (Ireland) clusters are stored in eu-west-1 for any group that does not have an active backup started prior to Jan. 31, 2017. All other snapshots are stored in MongoDB's data centers located on the East Coast of the U.S. or in the AWS us-east-1 region.*

Admin Username & Password

Important! *Make sure you save your password. You will need it to connect to your cluster.*

From the “Build Your New Cluster” pop up you can see that there are other options available including choosing a 3, 5 or 7 node replica set and up to a 12 shard cluster. Note that the free tier does not allow you to chose anything more than the 3 node cluster, but if you move into other sizes these options will become available. At this point we are almost ready; the last thing to address is the admin username and password. You may also choose to have a random password generated for you by clicking the “Autogenerate Secure Password” button. Finally, click the “Confirm & Deploy” button to create your Atlas cluster. Setting up your IP Whitelist

While Atlas is creating your database you will need to define which IP's are allowed access to your new database since MongoDB Atlas does not allow access from the internet by default. This list of granted IP addresses is called the “IP Whitelist”. To add the IP of your machine to this list click on the “Security” tab, then “IP Whitelist” then click the “+ ADD IP ADDRESS” button. This will pop up another dialog shown in Figure 3 below. You can click the “Add current IP Address” button to add your IP or provide a specific IP address or enable access to the world by not restricting IPs at all (not a fantastic idea but there in case you have no other choice and need to allow authentication from any IP).



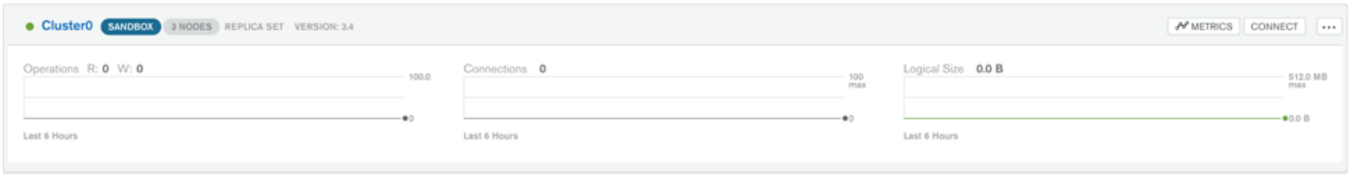
Add Wh… Entry

Add a whitelist entry using either CIDR notation or a single IP address. Learn more.

Whitelist Entry: Enter IP Address or CIDR Notation

Comment: Optional comment describing this entry

Once you have filled out this dialog click “Confirm” and this will update the firewall settings on your MongoDB Atlas cluster. Next, click on the “Clusters” tab and you should see your new MongoDB database ready for action!



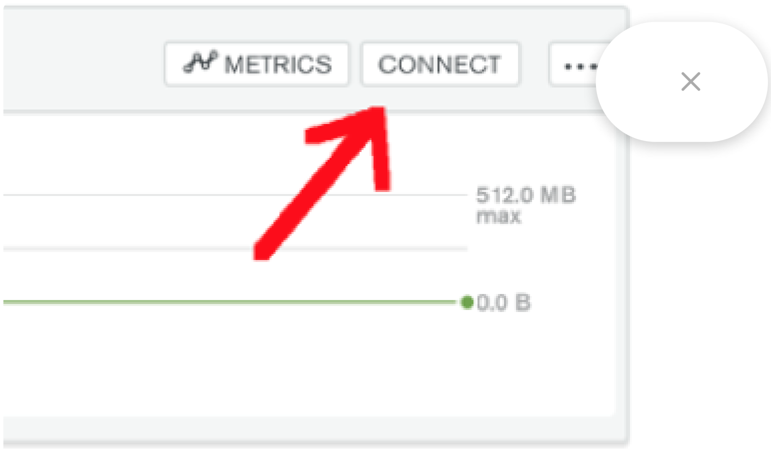
Operations R: 0 W: 0 — 100.0 — 0 — Last 6 Hours

Connections 0 — 100 max — 0 — Last 6 Hours

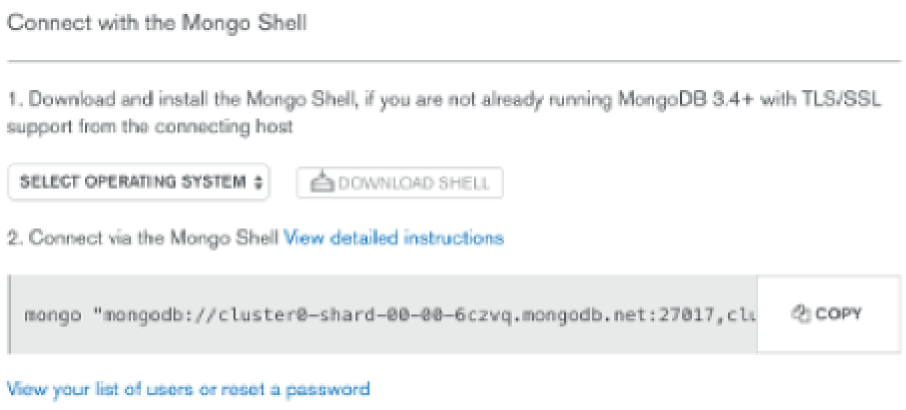
Logical Size 0.0 B — 512.0 MB max — 0.0 B — Last 6 Hours

Testing your connection

We want to make sure the MongoDB database is accessible from our development box before we start typing in code. A quick way to test is to make a connection using the Mongo Shell command line tool. Be sure to have your MongoDB connection information available. If you are using MongoDB Atlas you can obtain the connection information by clicking on the “Connect” button on the Clusters tab as shown in Figure 5.



The Connect button will launch a dialog that provides connection information. At the bottom of this dialog you will see a prepared command line ready for you to simply copy and paste in a command prompt.


Connect with the Mongo Shell

1. Download and install the Mongo Shell, if you are not already running MongoDB 3.4+ with TLS/SSL support from the connecting host

SELECT OPERATING SYSTEM DOWNLOAD SHELL

2. Connect via the Mongo Shell View detailed instructions

```
mongo "mongodb://cluster0-shard-00-00-6czvq.mongodb.net:27017,clu
```

COPY

View your list of users or reset a password

Note that if you copy the connection text as-is you will have to replace with the password for the **admin** user, and with the name of the database to which you wish to connect.

The command text that comes from this dialog is lengthy. For clarity, let's take a look at each of the parameters individually.

```
mongo
"mongodb://cluster0-shard-00-00-2ldwo.mongodb.net:27017,cluster0-shard-00-01-2ldwo.mongod
 --authenticationDatabase admin
--ssl
--username myadmin
--password S$meComPLeX1!
```

The first parameter is a string containing the list of all the nodes in our cluster including the definition of a replica set called, "`Cluster0-shard-0`". The next parameter, "`--authenticationDatabase`" tells which database contains the user we want to authenticate. The "`--ssl`" forces the connection to be encrypted via SSL/TLS protocol. Finally we provide the username and password, and we are connected! Note that if you are not using MongoDB Atlas, your MongoDB deployment may not have security enabled or require SSL. Thus, connecting to it could be as simple as typing “mongo” in the command prompt.

You are now ready to use MongoDB!

If you're interested in learning everything you need to know to get started building a MongoDB-based app you can sign up for one of our free online MongoDB University courses.

← Previous

Automated MongoDB Updates, No Problem with Atlas

As a developer, you have a lot of different options to run the MongoDB database for your application. But as you plan to launch your ap…

April 25, 2017

Next →

×

August 26, 2021

Resources

NoSQL Database Explained

MongoDB Architecture Guide

MongoDB Enterprise Advanced

MongoDB Atlas

MongoDB Realm

MongoDB Engineering Blog

Education & Support

View Course Catalog

Certification

MongoDB Manual

Installation

Support

Community

FAQ

Popular Topics

MongoDB on AWS

MongoDB on Google Cloud

Run MongoDB on Multiple Clouds with MongoDB Atlas

Migrate to MongoDB Atlas

What is a Cloud Database?

Building a REST API with MongoDB Realm

About

MongoDB, Inc.

Leadership

Press Room

Careers

Investors

Legal Notices

Privacy Notice

Security Information

Trust Center

Office Locations

Code of Conduct

Follow Us

Facebook

Github

Youtube

Twitter

LinkedIn

StackOverflow

Twitch