MongoDB Quick Start, in Docker

These instructions configure a server instance named mongo-test, running MongoDB version 4.4.0 in a basic Ubuntu Bionic image. My host machine is running Ubuntu. I'm assuming you've already installed Docker. If not, you might want to check out this article.

You'll probably need to sudo your docker commands.

Basics

Start a server instance:

```
docker run -p 27017:27017 --name mongo-test -d mongo:4.4.0-bionic
```

The port mapping (-p 27017:27017) is important. It allows you to connect to the running instance from your host machine.

A running instance can stopped with this:

```
docker stop mongo-test
```

And then started (or restarted) with this:

```
docker restart mongo-test
```

Open a bash shell in the running instance:

```
docker exec -it mongo-test bash
```

View MongoDB log files for the running instance:

```
docker logs mongo-test
```

Running Mongo Shell

You can run an interactive Mongo Shell a couple of ways.

Inside the running instance

First, open a bash shell inside the instance:

```
docker exec -it mongo-test bash
```

Then, run Mongo Shell:

```
mongo
```

From the host machine

First, install the MongoDB client tools:

```
sudo apt install mongodb-clients
```

Then, you can do this:

```
mongo --host localhost
```

Using Mongo Shell

List Databases

```
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
```

Use Database

MongoDB doesn't provide an explicit command for creating databases. A database will automatically be created the first time you try to use it (and add data).

Use a database called 'testdb':

```
> use testdb
switched to db testdb
```

In its simplest form, a database in MongoDB consists of two items:

1. A **document**, which contains data, and,
2. A **collection**, which is a container of documents.

A document is a data structure composed of field and value pairs. It's a JSON object that MongoDB

stores on disk in binary (BSON) format.

Drop Database

If you need to drop a database that's already been created, you switch to it ('use'), then issue a dropDatabase command:

```
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
testdb  0.000GB

> use testdb
switched to db testdb

> db.dropDatabase()
{ "dropped" : "testdb", "ok" : 1 }

> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
```

Add Data

Let's add some data documents to testdb, in a collection called 'people':

```
> use testdb
switched to db testdb

> db.people.insert( {firstName: 'John', lastName: 'Smith'} )
WriteResult({ "nInserted" : 1 })

> db.people.insert( {firstName: 'Bob', lastName: 'Jones'} )
WriteResult({ "nInserted" : 1 })

> db.people.find()
{ "_id" : ObjectId("5f4bc4dd54e2c67896143097"), "firstName" : "John",
"lastName" : "Smith" }
{ "_id" : ObjectId("5f4bc4e354e2c67896143098"), "firstName" : "Bob",
"lastName" : "Jones" }
```

The data in the insert commands is formatted as JSON, but quotes around key names are not required, and data can be single-quoted:

```
{
  firstName: 'John',
```

```
  lastName: 'Smith'
}
```

Update Data

To modify existing data, you pass two sets of data to update(): a filter, and an update action. The filter locates the document, and the update action specifies the data to modify.

In this example, we'll change the "Bob Jones" record to "Robert Jones":

```
> db.people.find()
{ "_id" : ObjectId("5f4bc4dd54e2c67896143097"), "firstName" : "John",
"lastName" : "Smith" }
{ "_id" : ObjectId("5f4bc4e354e2c67896143098"), "firstName" : "Bob",
"lastName" : "Jones" }

> db.people.update({ firstName: "Bob", lastName: "Jones" }, { $set:
{firstName: "Robert" } } )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.people.find()
{ "_id" : ObjectId("5f4bc4dd54e2c67896143097"), "firstName" : "John",
"lastName" : "Smith" }
{ "_id" : ObjectId("5f4bc4e354e2c67896143098"), "firstName" : "Robert",
"lastName" : "Jones" }
```

Remove Data

To remove data, you pass a filter to remove(), specifying the document (or documents) you want to remove.

In this example, we'll add a new document to the people collection, and then remove it.

```
> db.people.insert( { firstName: "To", lastName: "Remove" } )
WriteResult({ "nInserted" : 1 })

> db.people.find()
{ "_id" : ObjectId("5f4bc4dd54e2c67896143097"), "firstName" : "John",
"lastName" : "Smith" }
{ "_id" : ObjectId("5f4bc4e354e2c67896143098"), "firstName" : "Robert",
"lastName" : "Jones" }
{ "_id" : ObjectId("5f4bf7595402b299ee512fd8"), "firstName" : "To",
"lastName" : "Remove" }

> db.people.remove( { firstName: "To", lastName: "Remove"} )
WriteResult({ "nRemoved" : 1 })

> db.people.find()
```

```
{ "_id" : ObjectId("5f4bc4dd54e2c67896143097"), "firstName" : "John",
"lastName" : "Smith" }
{ "_id" : ObjectId("5f4bc4e354e2c67896143098"), "firstName" : "Robert",
"lastName" : "Jones" }
```

Managing Collections

To see the collections in a database:

```
> use testdb
switched to db testdb

> show collections
people
```

You can also use getCollectionNames(), which returns results as BSON:

```
> db.getCollectionNames()
[ "people" ]
```

Add a collection explicitly with createCollection:

```
> show collections
people

> db.createCollection("things")
{ "ok" : 1 }

> show collections
people
things
```

Drop a collection:

```
> show collections
people
things

> db.things.drop()
true

> show collections
people
```

Count of documents in a collection:

```
> db.people.count()
2
```

Retrieving Data

We've already employed a simple find in our add/update/delete examples: db.<collection_name>.find().

Find also accepts two optional parameters:

- Query filter: Describes how to filter the results, similar to a WHERE clause in SQL.
- Projection: Specifies which key/values from the document we want to see.

A find with no arguments retrieves up to the first 20 documents in a collection:

```
> db.people.find()
{ "_id" : ObjectId("5f4bc4dd54e2c67896143097"), "firstName" : "John",
"lastName" : "Smith" }
{ "_id" : ObjectId("5f4bc4e354e2c67896143098"), "firstName" : "Robert",
"lastName" : "Jones" }
```

A filter with an exact match on one key looks like this:

```
> db.people.find( {firstName: "John"} )
{ "_id" : ObjectId("5f4bc4dd54e2c67896143097"), "firstName" : "John",
"lastName" : "Smith" }
```

Matching on multiple keys, similar to an AND in a SQL WHERE clause, looks like this:

```
> db.people.find(
... {
... $and: [
... { firstName: "John" },
... { lastName: "Smith" }
... ]
... });
{ "_id" : ObjectId("5f4bc4dd54e2c67896143097"), "firstName" : "John",
"lastName" : "Smith" }
```

MongoDB supports the following query comparison operators: $eq, $gt, $gte, $lt, $lte, $ne, $in, and $nin, along with the following logical operators: $or, $and, $not, and $nor. Regex is also supported.

Projections can be used to limit the keys returned. For example, here's how to return just the last names:

```
> db.people.find( { }, { _id: 0, lastName: 1 } );
{ "lastName" : "Smith" }
{ "lastName" : "Jones" }
```

The numeric values indicate whether to include (1) or exclude (0) a given field. The _id field is always returned, unless specifically excluded.

Results can also be sorted:

```
> db.people.find( { }, { } ).sort( { lastName: 1 });
{ "_id" : ObjectId("5f4bc4e354e2c67896143098"), "firstName" : "Robert",
"lastName" : "Jones" }
{ "_id" : ObjectId("5f4bc4dd54e2c67896143097"), "firstName" : "John",
"lastName" : "Smith" }
```

The numeric value controls whether to sort ascending (1) or descending (-1).

For large result sets, the number of results to return can be specified:

```
> db.people.find( { }, { } ).limit( 1 );
{ "_id" : ObjectId("5f4bc4dd54e2c67896143097"), "firstName" : "John",
"lastName" : "Smith" }
```

MongoDB in .NET (C#)

Data in MongoDB can be accessed and manipulated in .NET (Standard and Core) applications using MongoDB.Driver. This is a simple introduction to connecting to MongoDB, retrieving data, and displaying it.

Create a .NET Core console application:

```
dotnet new console -o DotMongo
```

Add a reference to MongoDB.Driver:

```
cd DotMongo

dotnet add package MongoDB.Driver --version 2.11.1
```

Open Program.cs in your editor of choice, and replace the contents with this:

Program.cs

```
using System;
using MongoDB.Driver;
using MongoDB.Bson;

namespace DotMongo
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                var databaseName = "testdb";
```

```
                // Get database reference.
                var mongoDatabase = GetDatabaseReference("localhost",
27017, databaseName);
                Console.WriteLine($"Connected to database
{databaseName}");

                // Get a reference to the "people" collection inside
testdb.
                var collection =
mongoDatabase.GetCollection<BsonDocument>("people");

                // We're retrieving all documents in the collection,
                // but we still need an empty filter.
                var filter = new BsonDocument();
                var count = 0;

                // Open a cursor with all the matching documents.
                using (var cursor =
collection.FindSync<BsonDocument>(filter))
                {
                    // Iterate through the cursor
                    while (cursor.MoveNext())
                    {
                        // Get documents at the current cursor
location.
                        var batch = cursor.Current;

                        foreach (var document in batch)
                        {
                            // Get values from the current document,
then display them.
                            var firstName =
document.GetElement("firstName").Value.ToString();
                            var lastName =
document.GetElement("lastName").Value.ToString();

                            Console.WriteLine($"Full name: {firstName}
{lastName}");
                            count++;
                        }
                    }
                }
                Console.WriteLine($"Total records: {count}");

            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }
```

```
        public static IMongoDatabase GetDatabaseReference(string
hostName, int portNumber, string databaseName)
        {
            string connectionString =
$"mongodb://{hostName}:{portNumber}";

            // Connect to MongoDB
            var mongoClient = new MongoClient(connectionString);

            // Get a reference to the specified database
            var mongoDatabase = mongoClient.GetDatabase(databaseName);

            return mongoDatabase;
        }


    }
}
```

Run the application:

```
dotnet run
```

You should see output that looks like this:

```
Connected to database testdb
Full name: John Smith
Full name: Robert Jones
Total records: 2
```

You can find the full project here.

Learn More

MongoDB home page

MongoDB on GitHub

MongoDB on Docker Hub

MongoDB 3 Succinctly

database, docker