ECGR 4090/5090 Cloud Native Application Architecture
Lab 8 : MongoDB

MongoDB is among the most popular of the NoSQL document stores for Cloud Native applications.

Installation

Now that you are familiar with Docker from Lab 7, we can run MongoDB in a Docker container, conveniently downloaded from DockerHub

$ sudo docker run -d -p 27017-27019:27017-27019 --name mongodb mongo:latest

Verify container is running
$ sudo docker container ls

MongoDB shell

To access Mongo shell
$ sudo docker exec -it mongodb mongo

Let's try out a few of the MongoDB CRUD operations (from https://docs.mongodb.com/manual/) MongoDB stores data records as documents (specifically BSON documents) which are gathered together in collections. BSON stands for Binary Javascript Object Notation. It is a binary-encoded serialization of JSON documents. A database stores one or more collections of documents.

To select a database to use, in the mongo shell, issue the use <db> statement, as in the following example:
Note: > is the mongo shell prompt
> use myDB

Create (Insert) -
db.collection.insertOne() inserts a single document into a collection. If the collection does not currently exist, insert operations will create the collection.

```
> db.inventory.insertOne(
   { item: "canvas", qty: 100, tags: ["cotton"], size: { h: 28, w: 35.5, uom: "cm" } }
)
```

Insert multiple documents -
```
> db.inventory.insertMany([
   { item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" },
   { item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "A" },
```

```
   { item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" },
   { item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D" },
   { item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" }, status: "A" }
]);
```

Query (Read) -

To select all documents in the collection, pass an empty document as the query filter parameter to the find method. The query filter parameter determines the select criteria:

```
> db.inventory.find( {} )
```

Equality condition

The following example selects from the inventory collection all documents where the status equals "D":

```
> db.inventory.find( { status: "D" } )
```

AND conditions

```
> db.inventory.find( { status: "A", qty: { $lt: 30 } } )
```

OR conditions

```
> db.inventory.find( {
    status: "A",
    $or: [ { qty: { $lt: 30 } }, { item: /^p/ } ]
} )
```

Update -

The following example uses the db.collection.updateOne() method on the inventory collection to update the first document where item equals "paper":

```
> db.inventory.updateOne(
  { item: "paper" },
  {
    $set: { "size.uom": "cm", status: "P" },
    $currentDate: { lastModified: true }
  }
)
```

uses the $set operator to update the value of the size.uom field to "cm" and the value of the status field to "P"; uses the $currentDate operator to update the value of the lastModified field to the current date. If lastModified field does not exist, $currentDate will create the field.

The following example uses the db.collection.updateMany() method on the inventory collection to update all documents where qty is less than 50

```
> db.inventory.updateMany(
   { "qty": { $lt: 50 } },
   {
     $set: { "size.uom": "in", status: "P" },
     $currentDate: { lastModified: true }
   }
)
```

uses the $set operator to update the value of the size.uom field to "in" and the value of the status field to "P"; uses the $currentDate operator to update the value of the lastModified field to the current date. If lastModified field does not exist, $currentDate will create the field

The following example replaces the first document from the inventory collection where item: "paper":

```
> db.inventory.replaceOne(
   { item: "paper" },
   { item: "paper", instock: [ { warehouse: "A", qty: 60 }, { warehouse: "B", qty: 40 } ] }
)
```

Delete -

The following example deletes the *first* document where status is "D"

```
> db.inventory.deleteOne( { status: "D" } )
```

The following example removes all documents from the inventory collection where the status field equals "A":

```
> db.inventory.deleteMany({ status : "A" })
```

The following example deletes all documents from the inventory collection:

```
> db.inventory.deleteMany({})
```

Exit shell

```
> exit
```

Using MongoDB with Go
MongoDB Go driver enables you to programmatically access MongoDB with Go code.

Make a new lab8-mongodb directory.
Install the MongoDB Go driver

$ go get go.mongodb.org/mongo-driver

Download the mongo.go code from Canvas. Note how insertOne and findOne are done from the code.
$ go run mongo.go

If missing dependencies, go get the dependencies listed
Note: Be sure to use docker container inspect <container name> to determine the mongodb container IP address. Update the go code with this IP address.

This code is from the blog and github repo of Victor Tavares. See the repo for examples of other MongoDB operations
A more complete description is available in the GoDocs including client side encryption
https://pkg.go.dev/go.mongodb.org/mongo-driver/mongo

Note: Many examples on the internet use the mgo library which is no longer maintained. Avoid these.

To do -
In Lab 7 you containerized the web server of Lab 4. The application used a map to simulate a database. Replace the map with MongoDB. That is, all data associated with the application will be stored in MongoDB. Feel free to change the type of data that you are storing (for example, blog posts instead of product information).

Run MongoDB and the web server as two separate containers. Test your application using curl.

Note: A typical three tier web application consists of a web server, an application server hosting business logic, and a database server.