Guide to MongoDB - Indexing, Clustering, Aggregation Pipeline

Status	STARTED
Date	26 May 2020
Responsible	Mohammed Rasheedulla Shariff
Sign off/ Approved By	
Comment	
Team	Settlement Core

Category	Description
JIRA Issue	**SGC-17826** - Getting issue details... STATUS
Document Referred	

Discovery Documentation

Index-ing

- Indexes are special data structures, that store a small portion of the data set in an easy-to-traverse form.
- They store the value of a specific field or set of fields, ordered by the value of the field as specified in the index.
- MongoDB can return sorted results by using the ordering in the index.
- IMPROVES SEARCH OPERATIONS.

TYPE OF SCAN WITH INDEX : Indexes support the efficient execution of queries in MongoDB. Without indexes, MongoDB must perform a *collection scan*, i.e. scan every document in a collection, to select those documents that match the query statement. If an appropriate index exists for a query, MongoDB can use the index to limit the number of documents it must inspect.

Creating Index :

DATA FORMAT MongoScript vs Spring boot

Mongo shell data

```
{
        "_id":1,
        "index":1,
        "deptName":"dept1",
        "description":"this is a description for the department second time",
}
```

Create and define INDEX

Symtax

```
db.collection.createIndex( { <keyname>: [-1 | 1] }, <options> )

EG:
db.department.createIndex( {index:1}, {unique:true, name:"indexed_id"} )
db.department.createindex( {deptName: -1}, {unique:false, name:"indexed_deptname"} )
```

Definition

1. **Keyname** : Key to be INDEXED.
2. **1 | -1** : Order of the index will be stored(1 - Ascending, 2 - Descending).
3. **Unique :** The index value can be unique or repeated.
4. **Name** : User defined name for the INDEX created.

Spring boot input class

```
@Document(collection = "department")
public class Department {
        @Id
        private Integer id;

        @Indexed(unique = true, direction = IndexDirection.ASCENDING, name = "indexed_id")
        private Integer index;

            @Indexed(unique = false, name = "indexed_deptName", direction = IndexDirection.ASCENDING)
        private String deptName;

        private String description;
}
```

spring boot

```
    @Indexed(unique = true, direction = IndexDirection.ASCENDING, name = "indexed_id")
    private Integer index;

    @Indexed(unique = false, name = "indexed_deptName", direction = IndexDirection.ASCENDING)
    private String deptName;
```

1. **@Indexed** : annotate on the key to be INDEXED.
2. **direction** : Order of the index will be stored(Ascending, Descending).
3. **Unique** : The index value can be unique or repeated.
4. **name** : User defined name for the INDEX created.

Types of Index :

sl. no	Types	Description	syntax	
1	_id	All the collections of MongoDB have an index on the _id field that exists by default.	Default	
2	Single field	MongoDB supports user-defined indexes on a single field of a document	`@Indexed(name = "first_name_index", direction = IndexDirection.DESCENDING)` `private String firstName;`	

3	Compound Index	MongoDB also supports user-defined indexes on multiple fields.	`@Document`
			`@CompoundIndexes({`
			`@CompoundIndex(def = "{'firstName':1, 'salary':-1}", name = "compound_index_1"),`
			`@CompoundIndex(def = "{'secondName':1, 'profession':1}", name = "compound_index_2")})`
			`public class Person {`
			`/…/`
			`}`	

4	Multikey Index	MongoDB uses multikey indexes to index the content stored in arrays	`@Indexed(name = "multi_index", direction = IndexDirection.DESCENDING)`
			`private Address address;`
			
			`//Usuage example`
			`db.userdetails.find().sort({"address.locality.street": -1})`	

5	Text Index	Index on a STRING field with large text in it. Can be used to search/ sort a sub string.	`@TextIndexed(name = "desc_index", direction = IndexDirection.DESCENDING)` `private String description;`	
6	Hashed Indexes	MongoDB allows us to create hashed Indexes to reduce the size of the Indexes. It is because only the hash is stored in the Index instead of the entire key. Hashed keys don't support range queries. In sharding, if you are looking to use hash-based partitioning, then we need to have hashed Indexes on the shard key.	nill	

Clustering (Atlas mongoDB cloud)

- Creating a mongodB cluster with replica set on ATLAS.
- Connecting to Spring boot and mongodb compass
- Creating a database and document on the cluster.

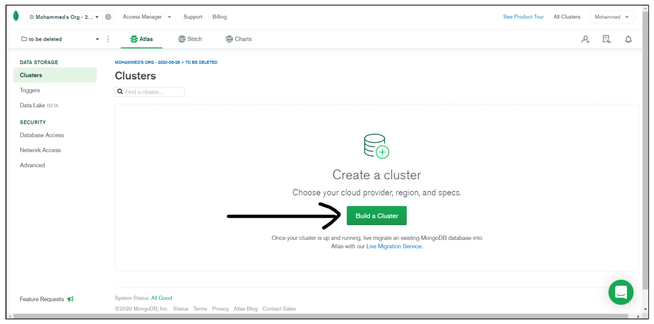
STEPS HOW TO :

Step 1 :- Go to the website and login/Register to start https://www.mongodb.com/cloud/atlas

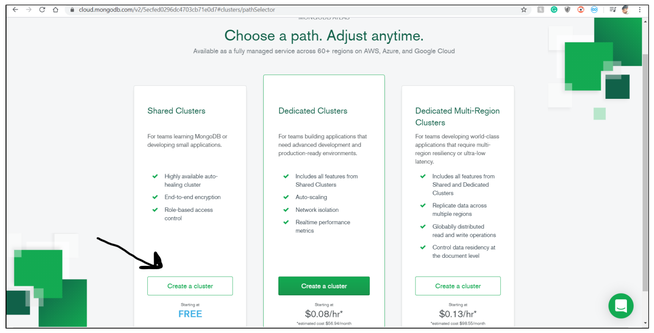
It automatically creates a **Project** to begin.

Step 2 :- Create a cluster

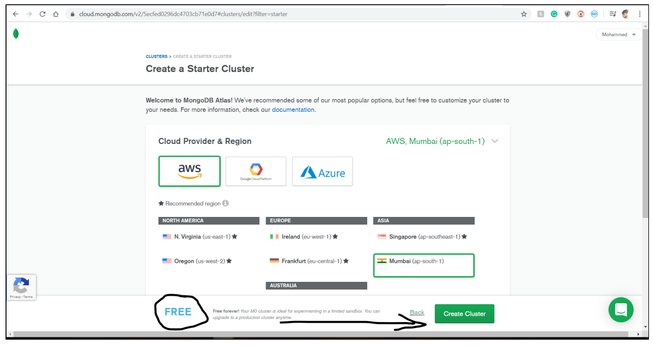
Click on BUILD A CLUSTER



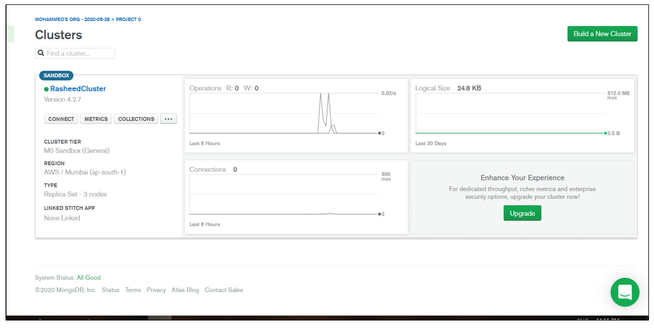
Select the FREE service for DEMO



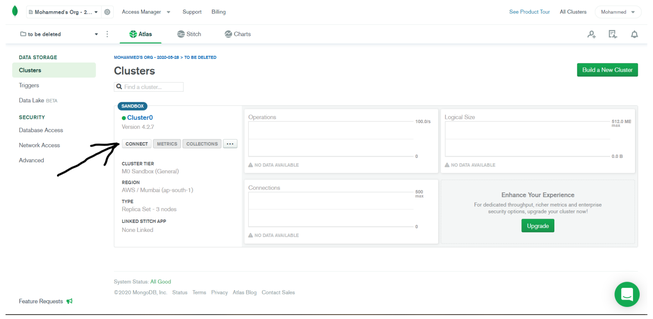
Select any **Cloud Provider & Region** and click create cluster



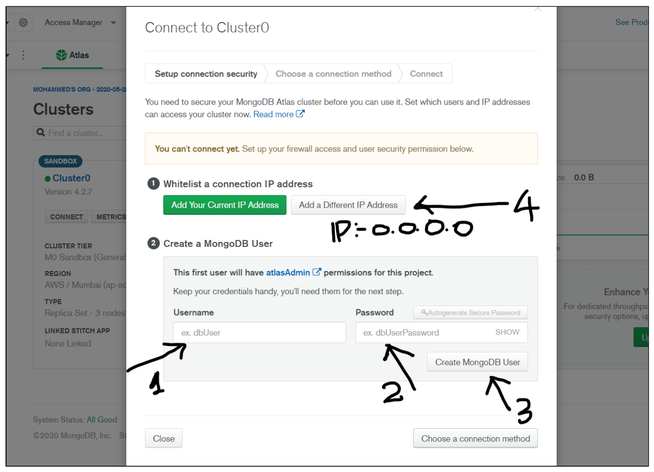
A CLUSTER IS CREATED with 3 replica sets nodes (1 Primary, 2 secondary)



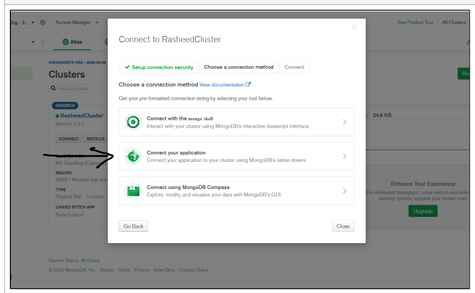
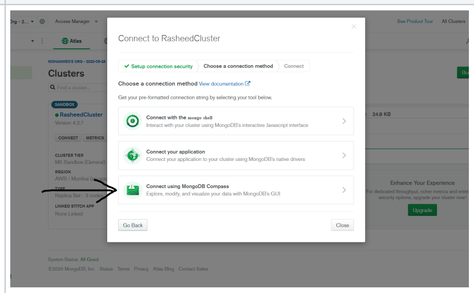
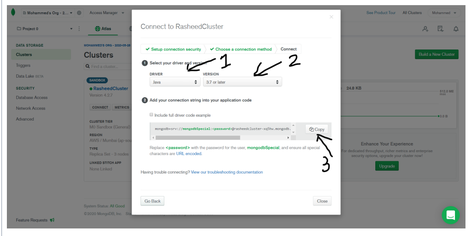
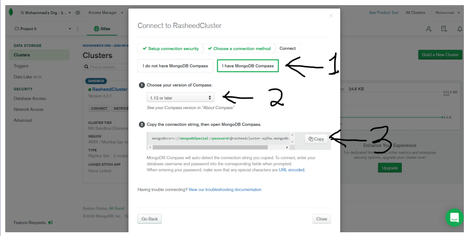
Step 3 :- Making a connection



But first we should Create a user and password for DB and mention IP port of local device



Choose a connection method

Spring boot connection method	Compass connection method
	
Copy the mondoDB URI	Copy the mondoDB URI
	

Add the mongoDB URI in APPLICATION.PROPERTIES

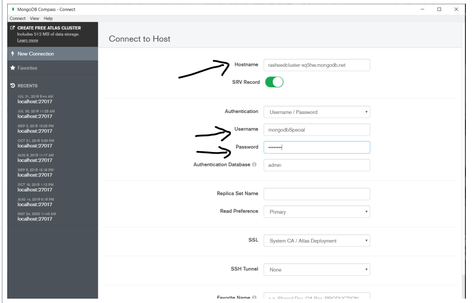
.

.

.

Replace <password> with UserPassword created in beginning of step 3

The properties will **AUTO FILL** in compass connection login



Step 4 :- CURD OPERATIONS

Spring boot CRUD method	Compass CRUD method
CRUD operations can be made using spring data operations	CRUD can be done using mondoDB commands

CRUD

```
@Autowired
private MongoTemplate mongoTemplate;

@Autowired
private MongoTemplateRepo mongoTemplateRepo;

@GetMapping("/get")
public List getMap(){
    return mongoTemplate.findAll(Department.class);
}
```

CRUD

```
db.department.findAll()
```

Filter Operations

sl. no	Operator	Description	Spring Boot	Shell

1	$gt, $lt, $gte, $lte
	
	$and, $or
		- Find with logical operators ($gt, $lt)
		- Find with combining queries ($and, $or)
		
		-Sort
			`@Autowired`
			`private MongoTemplate mongoTemplate;`
			
			`//Greater than $gt`
			`Query query = new Query();`
			`query.addCriteria(where("index").gt(2));`
			`mongoTemplate.find(query, Employee.class);`
			
			`//lesser than`
			`$gtQuery query = new Query();`
			`query.addCriteria(where("index").lt(2));`
			`mongoTemplate.find(query, Employee.class);`
			
			`//And operator`
			`Query query = new Query();`
			`query.addCriteria(where("index").gt(2).lt(4));`
			`mongoTemplate.find(query, Employee.class);`
			
			`Query query = new Query();`
			`query.addCriteria(where("index").gt(2).and("age").lt(25).and("name").is("Raj"));`
			`mongoTemplate.find(query, Employee.class);`
			
			`//Or operator`
			`Query query = new Query();`
			`Criteria criteria = new Criteria();`
			`criteria.orOperator(where("index").lt(10), where("name").is("Raj"));`
			`query.addCriteria(criteria);`
			`mongoTemplate.find(query, Employee.class);`
			
			`//SORT`
			`Query query = new Query();`
			`Criteria criteria = new Criteria();`
			`criteria.orOperator(where("index").gt(2), where("post").is("MID"));`
			`query.addCriteria(criteria).with(Sort.by("index").descending());`
			`mongoTemplate.find(query, Employee.class);`
				`//Greater than $gt`
				`db.collection.find( {index:{$gt:2}} )`
				`db.collection.find( {name:{$gt:"Raj"}} )`
				
				`//Lesser than $lt`
				`db.collection.find( {index:{$lt:2}} )`
				
				`//And operator`
				`db.collection.find( {index:{$gt:2, $lt:4}} )`
				`db.collection.find( {index:{$gt:2}}, {age:{$lt:25}}, {name:"Raj"} )`
				
				`//Or operator`
				`db.collection.find`
				`(`
				`$or:[`
				`{age:45},`
				`{index: {$lt:10}}`
				`]`
				`)`
				
				`//SORT`
				`db.collection.find`
				`(`
				`$or:[`
				`{age:45},`
				`{index: {$lt:10}}`
				`]`
				`).sort({index: -1})`

2	$in	- Matches any of the values specified in an array.	`Query query = new Query();`
			`query.addCriteria(where("name").in(Arrays.asList("Raj", "Zain")));`
			`mongoTemplate.find(query, Employee.class);`
				`db.collection.find(`
				`{name:`
				`$in: [`
				`"Raj",`
				`"Zain"`
				`]`
				`}`
				`)`

3	$project	Extract only the required fields	`Query query = new Query();`
			`query.addCriteria(where("index").gt(2)).fields().include("name");`
			`mongoTemplate.find(query, Employee.class);`
				`db.collection.find( {index:{$gt:2}}, [name: 1] )`

4	$regex	find by regex	`Query query = new Query();`
			`query.addCriteria(where("name").regex("IG"));`
			`mongoTemplate.find(query, Employee.class);`
				`db.collection.find( {name:{$regex: /Ra/}} )`

5	$exists, $type	-The key is present in the data($exists)
		
		-The value is of a specific type($type)
			`//$exist`
			`Query query = new Query();`
			`query.addCriteria(where("name").exists(false));`
			`mongoTemplate.find(query, Employee.class);`
			
			`//$type`
			`Query query = new Query();`
			`query.addCriteria(where("index").type(JsonSchemaObject.Type.INT_32));`
			`mongoTemplate.find(query, Employee.class);`
				`//exist`
				`db.collection.find( {name: {$exist:false}} )`
				
				`//type`
				`db.collection.find( {name: {$type: "int"}} )`

6	$mod	modulus	Query query = new Query();
			query.addCriteria(where("index").mod(4, 0));
			mongoTemplate.find(query, Employee.class);
				db.collection.find({name: {$mod: [4, 0] }})

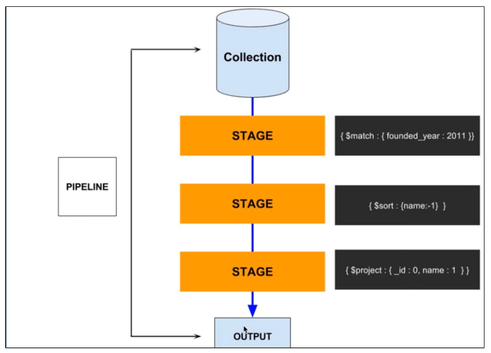
7	$text	**text indexes** to support **text search** queries on string content.
		
		Text indexes can include any field whose value is a string or an array of string elements.
			//Declaring text index in POJO
			@TextIndexed
			private String summary;
			
			//Usage
			Query query = TextQuery.queryText(TextCriteria.forDefaultLanguage().matchingAny("searchterm"));
			mongoTemplate.find(query, Employee.class);
				db.collection.find(
				{ $text:
				  {
				  $search: <string>,
				  $language: <string>,
				  $caseSensitive: <boolean>,
				  $diacriticSensitive: <boolean>
				  }
				})

8	$where	nil	nil	$where: "index = id"

9	$all
	
	$elemMatch
		Array query operator
		
		$all - Matches arrays that contain all elements specified in the query(For Array of String)
		
		$elemMatch - Selects documents if element in the array field matches all the specified $elemMatch conditions.(For array of object)
			//Array of String
			Query query = new Query();
			query.addCriteria(where("list").all(Arrays.asList("list1")));
			mongoTemplate.find(query, Employee.class);
			
			//Array of object
			Query query = new Query();
			query.addCriteria(where("idid").elemMatch(where("obID").is(1)));
			mongoTemplate.find(query, Employee.class);
				interests: $all: ["basketball", "video games"]
				
				//elemMatch
				recipe: $elemMatch: {"type": "veg", rating: {$gt: 3}}

Aggregate Pipeline

- Take Input from a single collection
- Pass the documents of the collection through one or more stages
- Each stage perform different operations in the Pipeline
- Each stage take as Input whatever the stage before produced as Output.
- The Input and Output for all stages are documents (stream of documents)
- At the end of Pipeline we get access to the output of the transformed and aggregated Output



Stage Operators

sl. no	Stage Operators	Description	Spring boot	Shell

1	$match	it filters the documents to pass only the documents that match the specified condition(s) to the next pipeline stage.	MatchOperation matchOperation = Aggregation.match(where("index").gt(2));
			Aggregation aggregation = Aggregation.newAggregation(matchOperation);
			AggregationResults<Object> result = mongoTemplate.aggregate(aggregation, Employee.class, Object.class);
			result.getMappedResults();
				//Syntax
				{ $match: { <query> } }
				
				Eg:
				db.collection.aggregate(
				[
				  //Stage 1
				  {$match: {age: {$gt:20}}
				])

2	$group	Groups input documents by the specified _id expression and for each distinct grouping.	MatchOperation matchOperation = Aggregation.match(where("index").gte(0));
			GroupOperation groupOperation = Aggregation.group("index", "name");
			Aggregation aggregation = Aggregation.newAggregation(matchOperation, groupOperation);
			AggregationResults<Object> result = mongoTemplate.aggregate(aggregation, Employee.class, Object.class);
			result.getMappedResults();
			
			//Other operations
			GroupOperation groupOperation = Aggregation.group("post").addToSet("index").as("ids").count().as("idCount");
				//Syntax
				{ $group:
				  {
				  _id: <expression>, // Group By Expression
				  <field1>: { <accumulator1> : <expression1> },
				  ...
				  }
				}
				
				Eg:
				db.collection.aggregate(
				[
				  //Stage 1
				  {$match: {age: {$gt:20}}
				
				  //Stage 2
				  {$group: {_id: {index: "$index", name: "$name"}}
				])

3	$count	Passes a document to the next stage that contains a count of the number of documents input to the stage.	MatchOperation matchOperation = Aggregation.match(where("index").gte(0));
			GroupOperation groupOperation = Aggregation.group("index", "name");
			CountOperation countOperation = Aggregation.count().as("index");
			Aggregation aggregation = Aggregation.newAggregation(matchOperation, groupOperation, countOperation);
			AggregationResults<Object> result = mongoTemplate.aggregate(aggregation, Employee.class, Object.class);
			result.getMappedResults();
				//Syntax
				{ $count: <alias string> }
				
				Eg:
				db.collection.aggregate(
				[
				  //Stage 1
				  {$match: {age: {$gt:20}}
				  //Stage 2
				  {$group: {_id: {index: "$index", name: "$name"}}
				  //Stage 3
				  {$count: "countedIndex"}
				])

4	$sort	Sorts all input documents and returns them to the pipeline in sorted order	MatchOperation matchOperation = Aggregation.match(where("index").gte(0));
			GroupOperation groupOperation = Aggregation.group("index", "post");
			SortOperation sortOperation = Aggregation.sort(Sort.Direction.DESC, "index");
			Aggregation aggregation = Aggregation.newAggregation(matchOperation, groupOperation, sortOperation);
			mongoTemplate.aggregate(aggregation, Employee.class, Object.class).getMappedResults()
				//Syntax
				{ $sort: { <field1>: <sort order>, <field2>: <sort order> ... } }
				
				db.collection.aggregate(
				[
				//Stage 1
				{$match: {age: {$gt:20}}
				//Stage 2
				{$group: {_id: {index: "$index", name: "$name"}}
				//Stage 3
				{$sort: {index: -1}
				])

5	$project	Passes along the documents with the requested fields to the next stage in the pipeline. The specified fields can be existing fields from the input documents or newly computed fields	MatchOperation matchOperation = Aggregation.match(where("index").gte(0));
			GroupOperation groupOperation = Aggregation.group("index", "post", "name");
			ProjectionOperation projectionOperation = Aggregation.project().andExpression("index * index").as("Square");
			ProjectionOperation projectionOperation2 = Aggregation.project().andExclude("name");
			Aggregation aggregation = Aggregation.newAggregation(matchOperation, groupOperation, projectionOperation, projectionOperation2);
			mongoTemplate.aggregate(aggregation, Employee.class, Object.class).getMappedResults();
				//Syntax
				{ $project: { "<field1>": 0, "<field2>": 0, ... } }
				
				db.collection.aggregate(
				[
				//Stage 1
				{$match: {age: {$gt:20}}
				//Stage 2
				{$group: {_id: {index: "$index", name: "$name"}}
				//Stage 3
				{$project: {index: 0, name:1}
				])

Q&A