mongoDB®

INF 551

Wensheng Wu

Installation on EC2

- Create a new yum repository file for MongoDB
  - cd /etc/yum.repos.d
  - sudo vi mongodb-org-3.4.repo

- Add the following lines to the file:
  - [mongodb-org-3.4]
  - name=MongoDB Repository
  - baseurl=https://repo.mongodb.org/yum/amazon/2013.03/mongodb-org/3.4/x86_64/
  - gpgcheck=1
  - enabled=1
  - gpgkey=https://www.mongodb.org/static/pgp/server-3.4.asc

Installation on EC2

- sudo yum -y install mongodb-org

- sudo service mongod start
  - Start the server

- sudo service mongod stop
  - Stop it

Installation on EC2

- For more details see instructions here:
  - https://docs.mongodb.com/v3.4/tutorial/install-mongodb-on-amazon/

Document store

- MongoDB is a document database
- A document is similar to an JSON object
  - Consists of field-value pairs
  - Value may be another document, array, string, number, etc.
- Document = record/row in RDBMS

Collections

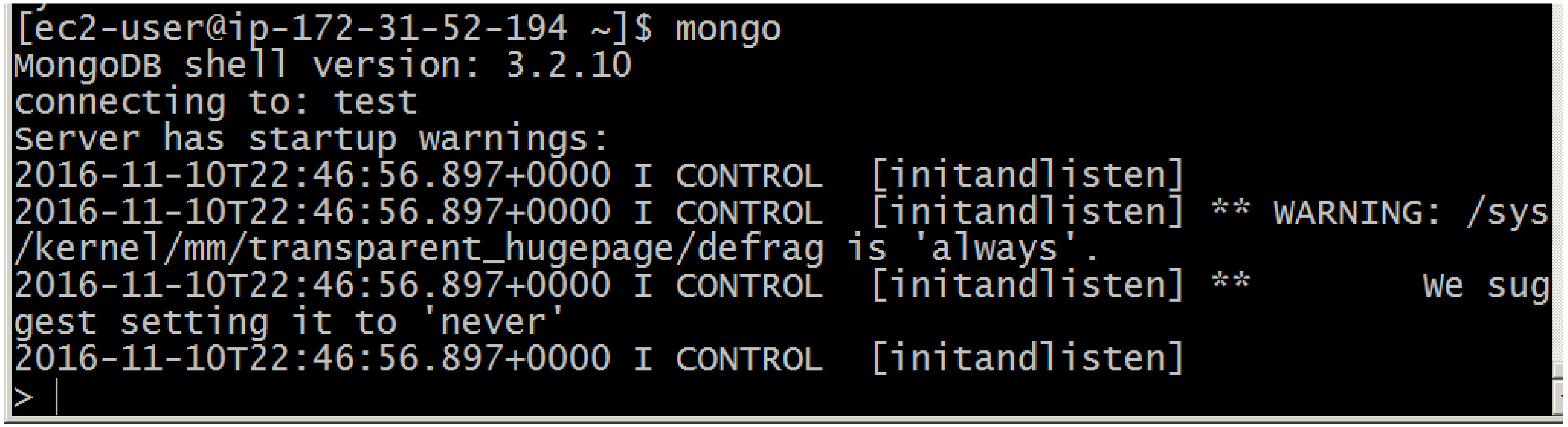
- Documents are stored in a collection
- Collection = table in RDBMS
- But documents may have different structures
  - In contrast, records in RDBMS have the same schema

Primary key

- Every document has a unique _id field
  - That acts as a primary key

MongoDB shell

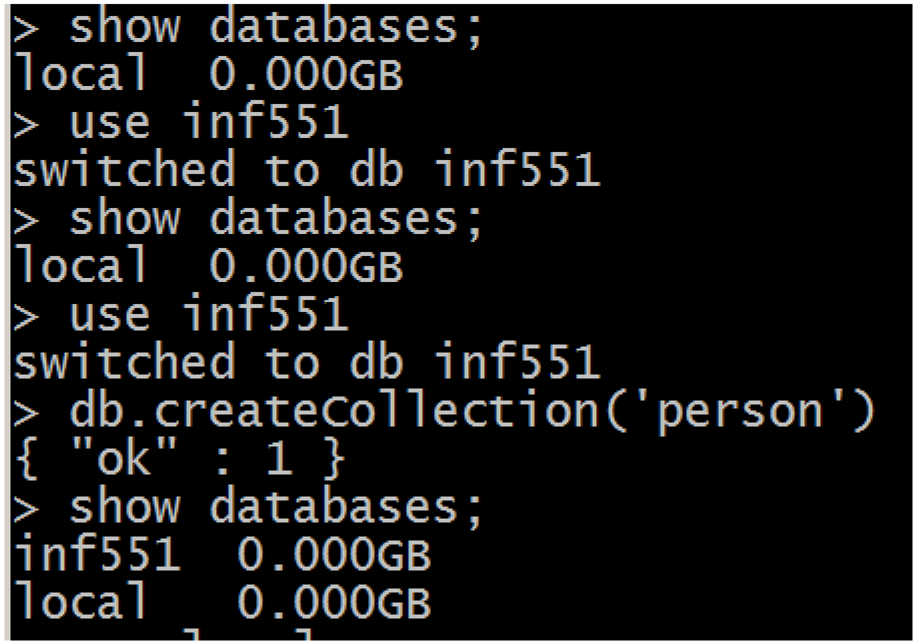
- mongo



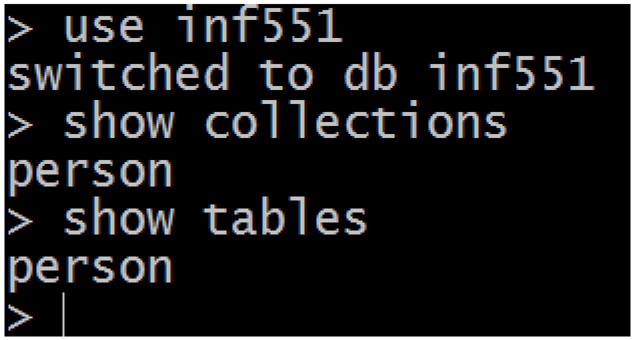
```
[ec2-user@ip-172-31-52-194 ~]$ mongo
MongoDB shell version: 3.2.10
connecting to: test
Server has startup warnings:
2016-11-10T22:46:56.897+0000 I CONTROL  [initandlisten]
2016-11-10T22:46:56.897+0000 I CONTROL  [initandlisten] ** WARNING: /sys
/kernel/mm/transparent_hugepage/defrag is 'always'.
2016-11-10T22:46:56.897+0000 I CONTROL  [initandlisten] **        We sug
gest setting it to 'never'
2016-11-10T22:46:56.897+0000 I CONTROL  [initandlisten]
> 
```

Create a new database

- No need to explicitly create it, just use it
  - It will be automatically created once you add a collection (i.e., table) to it



```
> show databases;
local   0.000GB
> use inf551
switched to db inf551
> show databases;
local   0.000GB
> use inf551
switched to db inf551
> db.createCollection('person')
{ "ok" : 1 }
> show databases;
inf551  0.000GB
local   0.000GB
```



```
> use inf551
switched to db inf551
> show collections
person
> show tables
person
>
```

Databases

- use inf551
  - Switch to database "inf551"

- show databases
  - List all databases

- show tables
  - List all tables/collections in the current db
  - Can also say "show collections"

Create/drop a collection

- db.createCollection('person')
  - db is a shell variable representing the current db

- db.person.drop()
  - Dropping a collection

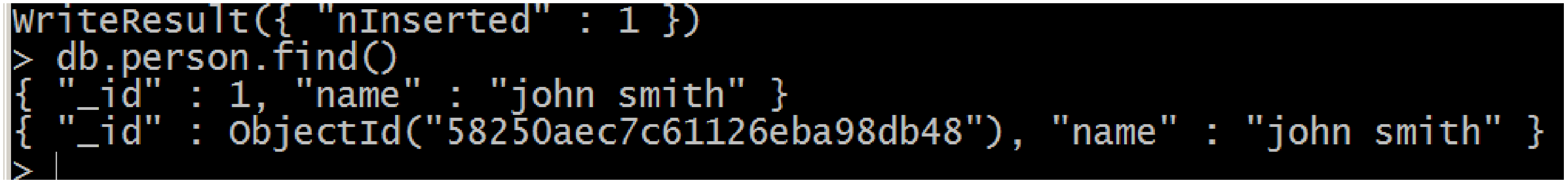
Adding documents

- db.person.insert({"_id": 1, "name": "john smith"})

- db.person.insert({"_id": 1, "name": "david smith"})
  - Error: duplicate key!

ObjectId()

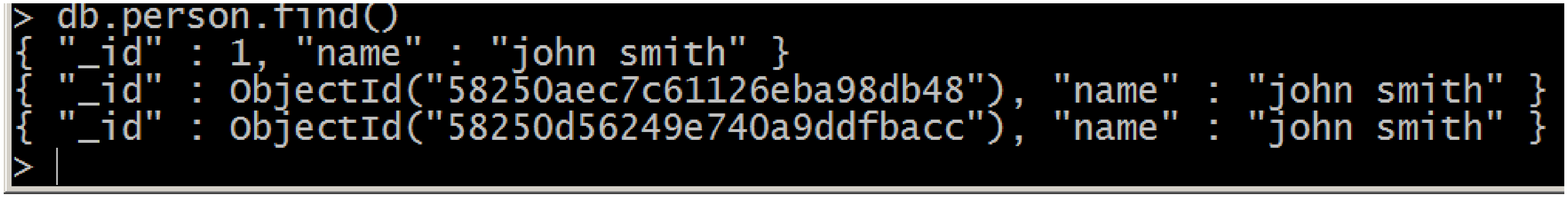
- ObjectId() function creates an ID
- db.person.insert({"_id": ObjectId(), "name": "john smith"})



```
WriteResult({ "nInserted" : 1 })
> db.person.find()
{ "_id" : 1, "name" : "john smith" }
{ "_id" : ObjectId("58250aec7c61126eba98db48"), "name" : "john smith" }
>
```

ObjectId()

- db.person.insert({"name": "john smith"})
  - Here no specification of "_id" field
  - Bu an id will be automatically created



```
> db.person.find()
{ "_id" : 1, "name" : "john smith" }
{ "_id" : ObjectId("58250aec7c61126eba98db48"), "name" : "john smith" }
{ "_id" : ObjectId("58250d56249e740a9ddfbacc"), "name" : "john smith" }
> |
```

ObjectId()

- A 12-byte hexademical value
  - E.g., 58250aec7c61126eba98db48

- Among 12 bytes:
  - 4-byte: the seconds since 1970/1/1
  - 3-byte: machine identifier
  - 2-byte: process id
  - 3-byte: a counter, starting with a random value

Embedded sub-document

- db.person.insert(

```
{
"name": "david johnson",
"address": {"street": "123 maple",
            "city": "LA",
            "zip": 91989},
"phone": ["323-123-0000", "626-124-0999"]
})
```

Array

Insert some more documents

- db.person.insert({"name": "kevin small", "age": 35})

- db.person.insert({"name": "mary lou", "age": 25})

Query

- db.person.find()
  - Return all documents in person

- db.person.find({"name": "kevin small"})
  - Return all documents with specified name

Using query operators

- db.person.find({"age": {$gt: 25}})

- db.person.find({"name": "kevin small", "age": {$gt: 25}})
  - Specify "and" condition

- db.person.find({ $or: [{"name": "kevin small"}, {"age": {$gt: 25}}] })
  - Specify "or" condition

Query operators

- Introduced by $
- $lt, $gt, $lte, $gte, $ne
  - Comparison operators
- $or, $and, $not
  - Logical operators

Projection

- db.person.find(

```
{"age": {$ne: 25} },
{"name":1, "age": 1}
)
```

1: included in result; 0: do not

- This will return name and age (plus _id)
  - i.e., similar to 'select _id, name, age from users where age != 25'

Projection

- db.person.find(

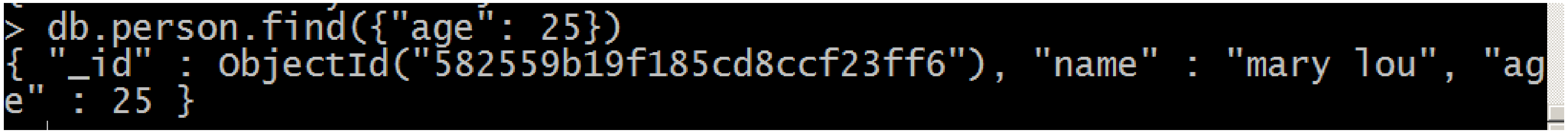
```
        {"age": {$ne: 25} },
        {"name":1, "age": 1, "_id": 0}
 )
```

- This does not return id, e.g.,

```
 { "name" : "john smith" }
 { "name" : "david johnson" }
 { "name" : "kevin small", "age" : 35 }
```

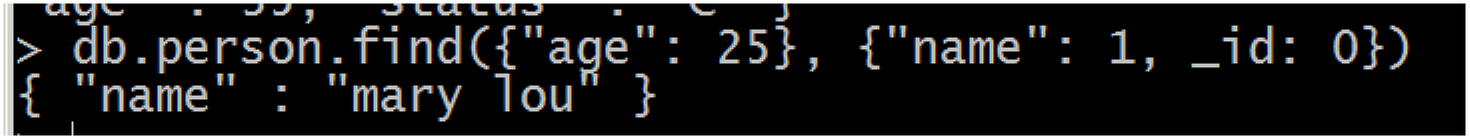
Example

- Without projection



```
> db.person.find({"age": 25})
{ "_id" : ObjectId("582559b19f185cd8ccf23ff6"), "name" : "mary lou", "ag
e" : 25 }
```

- With projection



```
> db.person.find({"age": 25}, {"name": 1, _id: 0})
{ "name" : "mary lou" }
```

Update documents

- db.person.update(
  { "age": { $gt: 25 } },
  { $set: { "status": "C" } },
  { multi: true }
)

Existing documents may not have status field; if not, insert it instead

Update one or all documents

Similar to:

Update users set status = 'C' where age > 25

Another example

- db.person.update({}, {$set: {"status":'C'}}, {multi:true})
  - Note the empty query {}

- Add "status" field to all documents

Remove fields

- db.person.update({}, {$unset: {"status": ""}}, {multi: true})

- Remove the "status" field from all documents

Delete

- db.person.remove({})
  - Remove all documents

- db.person.remove({ "age": {$gt: 30} })
  - Delete with a condition

Count()

- db.person.count()
  - Return # of documents in the person collection

Query on embedded document

- Using dot notation to identify field in embedded document
- db.person.find({"address.city": "LA"})
  - Return all documents whose city sub-field of address field = "LA"

Aggregation

- db.person.aggregate([{"$group": {_id: "$age", total:{$sum:1}}}])

```
{ "_id" : 25, "total" : 1 }
{ "_id" : 35, "total" : 1 }
{ "_id" : null, "total" : 4 }
```

Sharding in MongoDB

- Done at collection-level
  - Distribute records in a collection over multiple machines

- User can specify shard key
  - i.e., a field in a document

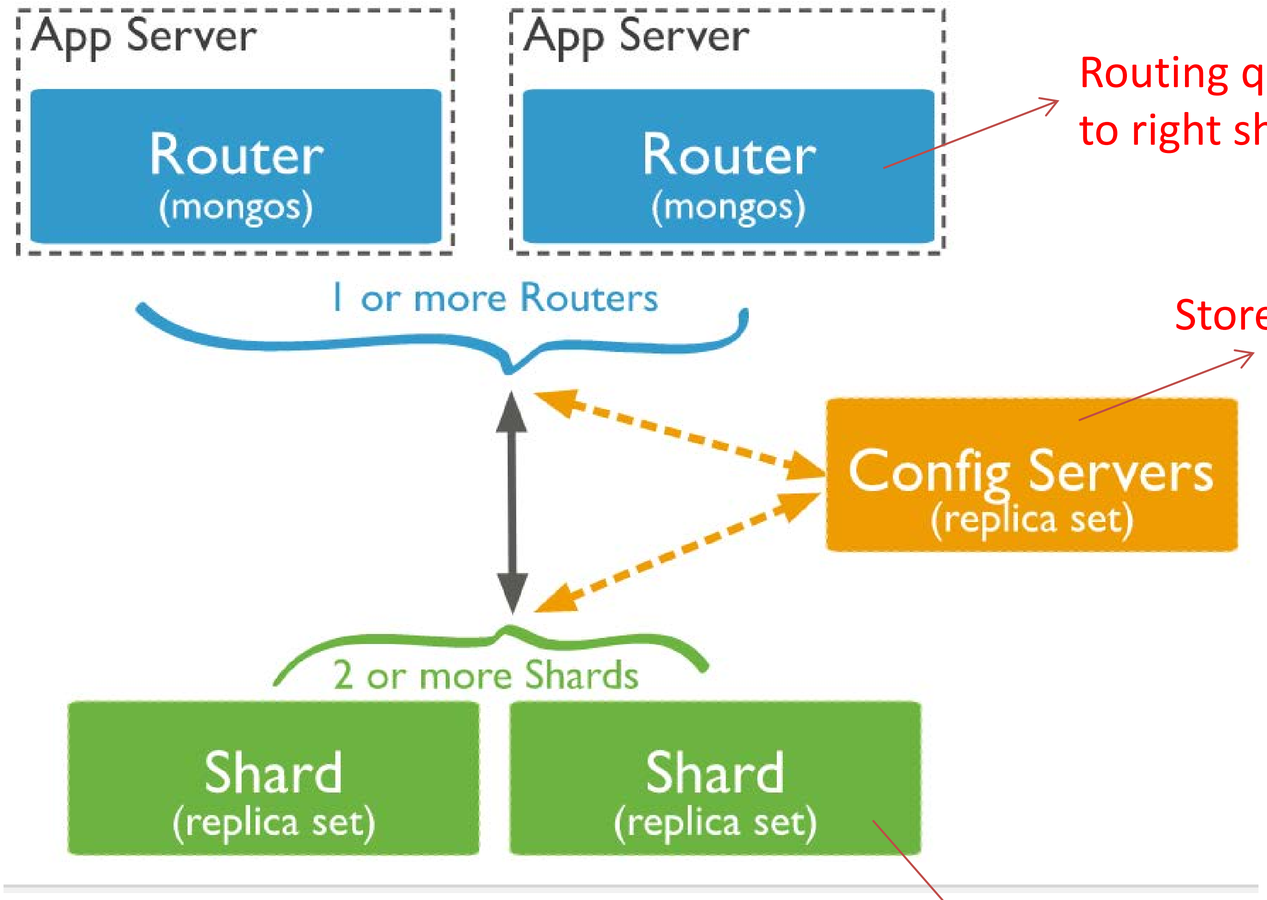
- Support sharding by key range or hashing

Sharding

- Method of distributing data across multiple machines

- Sharding for horizontal scaling

Shared cluster



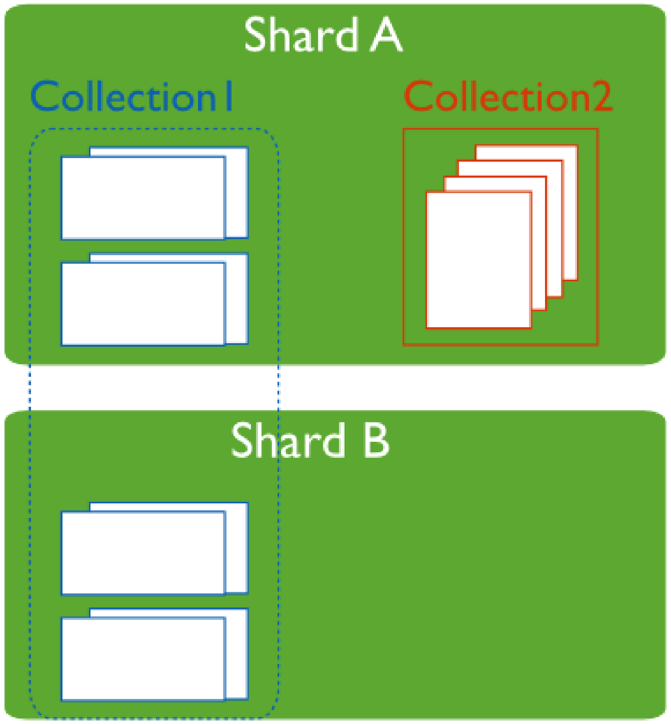
Routing queries to right shards

Store metadata

Each shard may have multiple replicas

Sharded and non-sharded collections

- Collection 1 is sharded; collection 2 is not



Import external dataset

- mongoimport --db inf551 --collection lax --file lax.json
  - No need to pre-create inf551 and lax if they do not exist yet

- More details:
  - https://docs.mongodb.com/getting-started/shell/import-data/

Resources

- CRUB operations in MongoDB
  - https://docs.mongodb.com/v3.4/crud/
  - Create => insert()
  - Read => find()
  - Update => update()
  - Delete => remove()

- Cursor in mongo shell
  - https://docs.mongodb.com/v3.3/tutorial/iterate-a-cursor/
  - https://docs.mongodb.com/v3.4/reference/method/js-cursor/