PAC_ONE Final Report

1. Team Name
PAC_ONE

2. Team Members
Anuja Tike, Cassidy Tarng, Prathyusha Manusanipalli

3. Project Name
Air Planner

4. Project Description
The project will allow users to view certain statistics and data regarding recorded flights in the U.S. from December 2013 to January 2017. The user will be able to view information to help plan their flights based on what has happened in the past 3 years. With this data, the user can plan ahead to accommodate occurrences such as delays, cancellations, flight time, etc. Furthermore, the use would be able to view travel statistics such as most common destination from a certain state.

5. Project Successes - List and describe the highlights and key success factors of the project

Project Success:
This project allowed to work on and understand NoSQL databases thoroughly. This was one of the most important success factors of the project. Other success factors include developing a python API, using MongoDB concepts studied in class for application purpose, learning how to use virtual machine as well as Amazon AWS for the project. Concepts related to MongoDB such as writing of queries in mongo shell, creating collections and using different techniques such as indexing to achieve efficient results in time, creation and use of mongo instances, configuration servers, sharding, replication for the purpose of data retrieval and fault tolerance were used and successfully understood. Finally, our team is successful in getting the desired results from our chosen dataset using MongoDB.

6. Unexpected Events - List and describe any unexpected events that occurred during the project and the impact that those events may have had on the project and the action(s) taken to address them.

Unexpected Event 1:
There were two major unexpected events occurred during the project. First was, as we had to create replicas and sharding both local VM was crashed due to load many times. One time it worked but as the data is above 1.5 GB, it was taking long time to get results for that data in the first run. After that VM's crashed.

Impact of Unexpected Event 1:
The impact this event may have had on the project is, it would not let us finish our project and it would delay the retrieval of results. Getting result in short time is required because testing is based on that i.e whether the obtained result is correct or not. If it is not correct then necessary corrections should be done on the query to get correct result. Hence, delay in retrieval of results will impact the whole process of development and testing.

Action Taken to resolve Unexpected Event 1:
Action we took to address the first event was, as laptops were not having enough memory and VM's were crashing due to load, we did project on AWS. Also, AWS free tier instances were not sufficient to load the data of around 1.5 GB. So, we used "t2.medium" instances of AWS which has 4 GB memory and are not free.

Unexpected Event 2:
We created instances, replica sets , config server replica sets and shard servers in AWS. We also imported data in created db and collection. It worked fine till these steps. In the last step, when we tried doing sh.status() to check whether the data is sharded properly in shard servers, it could not detect primary in config server replica set.

Impact of Unexpected Event 2:
The impact of this unexpected event was, our imported data was not getting sharded. And because of this we could not run queries in mongos.

Action Taken to resolve Unexpected Event 2:
We tried every possible way to resolve this issue but nothing worked. It was telling that primary in config server is not found though config server replica set had primary. We checked rs.status() and it showed everything was fine, there was one primary and two secondaries.

So, we forced primary to step down using following command:

```
db.runCommand( {
   replSetStepDown: <seconds>,
   secondaryCatchUpPeriodSecs: <seconds>,
   force: <true>
} )
```

This command can only be run in admin database. After running this command primary was forced to step down. Then again election took place and other replica became primary and previous primary became secondary.
After this we again tried to connect mongos and performed sh.status(), it executed perfectly.

7. Lessons Learned - List and describe any lessons learned from this project and provide recommendations that can be used to improve the delivery of future information systems projects.

Lessons Learned:
This project helped to get good understanding of NoSQL database concepts and their application in real life. Also, we got chance to learn python API programming which was new for us.
Recommendation:
Data visualization with any of the visualization tools like Tableau etc. can be added on top of Python API can be used to improve the delivery of future information systems projects.

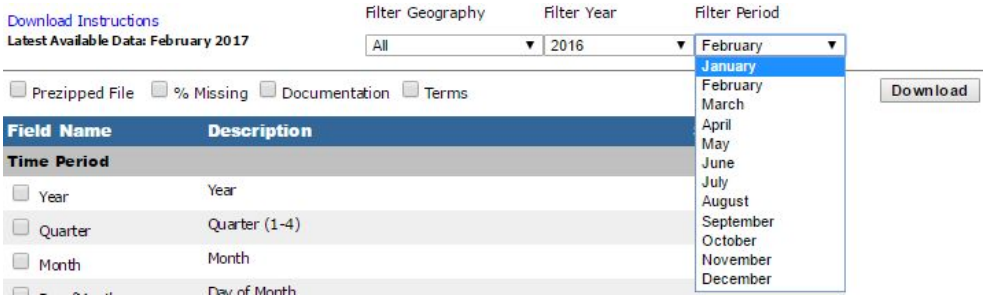
8. DataSet

URL of the data
https://www.transtats.bts.gov/DL_SelectFields.asp?Table_ID=236&DB_Short_Name=On-Time

One line of description of data set including its size
The dataset chosen is from December 2013 to January 2017 and represents the flight records in the United States by the Bureau of Transportation Statistics.
Total size: 1.428GB

Detailed description of data wrangling method
Fortunately, the dataset did not need much cleaning or reformatting. However, this dataset did require a lot of editing in order to create one full CSV file. The following images shows how we compiled all data sets we needed in order to create 1 master CSV file.

1. we had to individually download each month. We chose the dates of December 2013 to January 2017 which resulted in 30 mb files per month.



2. After downloading, we were left with 38 separate files that we needed to merge into one file.

```
 Directory of C:\Users\Cassidy Tarng\CS185C\Project\AirStats\Split

05/04/2017  04:52 PM    <DIR>          .
05/04/2017  04:52 PM    <DIR>          ..
03/14/2017  07:58 PM        38,853,392 2013-12.csv
03/14/2017  07:59 PM        27,499,148 2014-01.csv
03/14/2017  07:33 PM        24,698,512 2014-02.csv
03/14/2017  07:35 PM        28,560,237 2014-03.csv
03/14/2017  07:36 PM        28,045,659 2014-04.csv
03/14/2017  07:37 PM        29,140,325 2014-05.csv
03/14/2017  07:38 PM        29,654,283 2014-06.csv
03/14/2017  07:41 PM        31,040,324 2014-07.csv
03/14/2017  07:44 PM        30,055,311 2014-08.csv
03/14/2017  07:45 PM        27,452,958 2014-09.csv
03/14/2017  07:47 PM        28,836,352 2014-10.csv
03/14/2017  07:48 PM        26,874,971 2014-11.csv
03/14/2017  07:49 PM        27,900,904 2014-12.csv
03/14/2017  07:22 PM        26,455,347 2015-01.csv
03/14/2017  06:51 PM        23,897,972 2015-02.csv
03/14/2017  06:57 PM        27,687,746 2015-03.csv
03/14/2017  06:58 PM        27,176,966 2015-04.csv
03/14/2017  06:59 PM        28,188,545 2015-05.csv
03/14/2017  07:00 PM        28,735,181 2015-06.csv
03/14/2017  07:01 PM        29,989,614 2015-07.csv
03/14/2017  07:02 PM        29,803,558 2015-08.csv
03/14/2017  07:03 PM        27,185,579 2015-09.csv
03/14/2017  07:04 PM        28,387,887 2015-10.csv
03/14/2017  07:05 PM        26,905,060 2015-11.csv
03/14/2017  07:06 PM        27,830,652 2015-12.csv
03/14/2017  06:43 PM        26,413,829 2016-01.csv
```

3. Then we opened each file except for the first file and removed the header using Excel so that the header will not be included into the data when merging.

	A	B	C	D	E	F	G	H	I
1	YEAR	QUARTER	MONTH	DAY_OF_M	DAY_OF_W	FL_DATE	UNIQUE_C	AIRLINE_II	CARRIER
2	2013	4	12	25	3	########	DL	19790	DL
3	2013	4	12	25	3	########	DL	19790	DL
4	2013	4	12	25	3	########	DL	19790	DL

4. Finally using the command line we merged all files into one Master.csv file using the command “copy *.csv Master.csv”

```
C:\Users\Cassidy Tarng\CS185C\Project\AirStats\Split>copy *.csv Master.csv
2013-12.csv
2014-01.csv
2014-02.csv
2014-03.csv
2014-04.csv
2014-05.csv
2014-06.csv
2014-07.csv
2014-08.csv
2014-09.csv
2014-10.csv
2014-11.csv
2014-12.csv
2015-01.csv
2015-02.csv
2015-03.csv
2015-04.csv
2015-05.csv
2015-06.csv
2015-07.csv
2015-08.csv
2015-09.csv
2015-10.csv
2015-11.csv
2015-12.csv
2016-01.csv
2016-02.csv
2016-03.csv
2016-04.csv
2016-05.csv
2016-06.csv
2016-07.csv
2016-08.csv
2016-09.csv
2016-10.csv
2016-11.csv
2016-12.csv
2017-01.csv
        1 file(s) copied.
```

9. NoSQL configuration including virtual machines, replica set, and sharding configuration

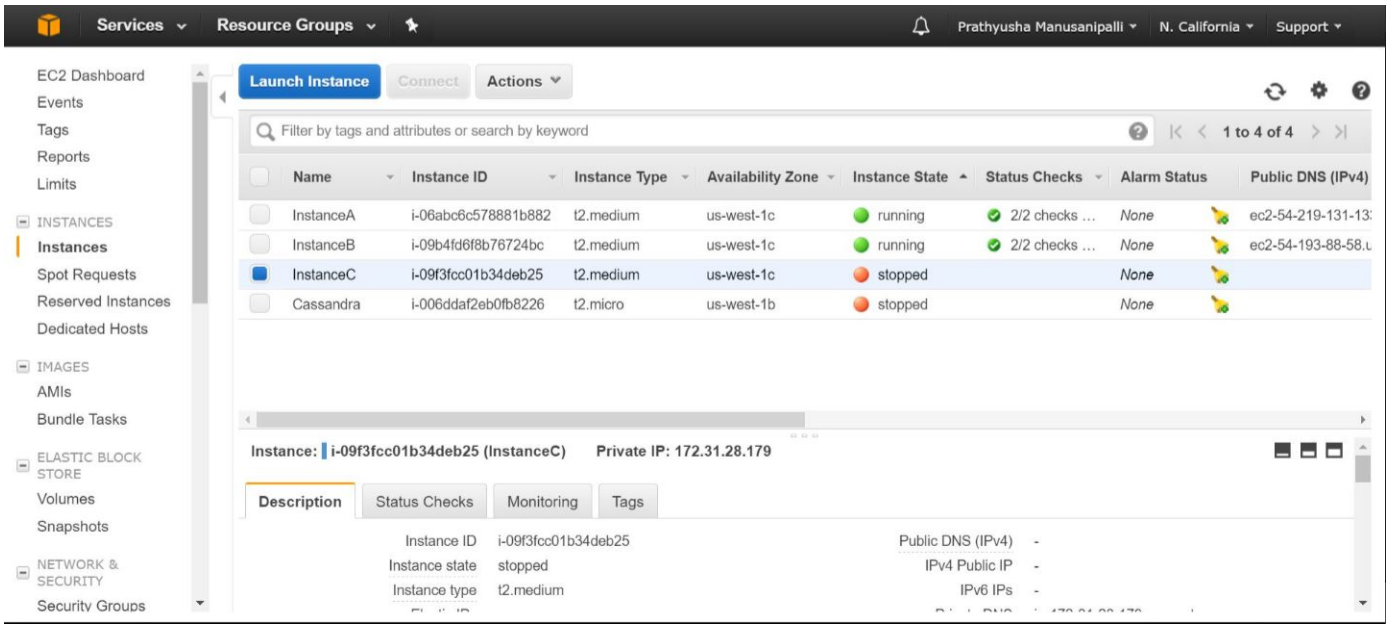
Virtual Machines: Two instances of AWS.
Replica sets: Two Replica sets in each instance.
Sharding: Two shards replica sets in each node.

Instance	**Public DNS**	**Private IP**
Instance A	ec2-54-219-131-133.us-west-1.compute.amazonaws.com	172.31.17.181
Instance B	ec2-54-193-88-58.us-west-1.compute.amazonaws.com	172.31.29.113

Service	**Daemon**	**port**	**dbpath**	**Instance**
ShardController	mongos	29031		
Config Server1	mongod	29024	/db/configs/data1	A(172.31.17.181)

Config server2	mongod	29025	/db/configs/data2	A(172.31.17.181)
Config server3	mongod	29026	/db/configs/data3	A(172.31.17.181)
Shard1 replica1	mongod	29021	/db/shard1copy1/data	A(172.31.17.181)
Shard1 replica2	mongod	29022	/db/shard1copy2/data	A(172.31.17.181)
Shard1 replica3	mongod	29023	/db/shard1copy3/data	A(172.31.17.181)
Shard2 replica1	mongod	29021	/db/shard2copy1/data	B(172.31.29.113)
Shard2 replica2	mongod	29022	/db/shard2copy2/data	B(172.31.29.113)
Shard2 replica3	mongod	29023	/db/shard2copy3/data	B(172.31.29.113)

Step1: Aws Console



Step2 : Making 3 directories on Instance A (172.31.17.181). One for each of the 3 replicas of Shard1.

$ sudo mkdir -p /db/shard1copy1/data /db/shard1copy2/data /db/shard1copy3/data

```
ubuntu@ip-172-31-17-181: ~
Using username "ubuntu".
Authenticating with public key "imported-openssh-key"
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.4.0-1013-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud

42 packages can be updated.
22 updates are security updates.


Last login: Sat May  6 23:10:54 2017 from 130.65.254.8
ubuntu@ip-172-31-17-181:~$ sudo mkdir -p /db/shard1copy1/data /db/shard1copy2/da
ta /db/shard1copy3/data
ubuntu@ip-172-31-17-181:~$
```

Step 3: On InstanceA getting the shard server of shard1 replica1 up on port 29021.
$ sudo mongod --port 29021 -dbpath /db/shard1copy1/data --shardsvr --replSet repl1 --smallfiles --oplogSize 128

```
ubuntu@ip-172-31-17-181: ~
ubuntu@ip-172-31-17-181:~$ sudo mkdir -p /db/shard1copy1/data /db/shard1copy2/da
ta /db/shard1copy3/data
ubuntu@ip-172-31-17-181:~$ sudo mongod --port 29021 -dbpath /db/shard1copy1/data
 --shardsvr --replSet repl1 --smallfiles --oplogSize 128
2017-05-06T23:36:15.008+0000 I CONTROL  [initandlisten] MongoDB starting : pid=1
604 port=29021 dbpath=/db/shard1copy1/data 64-bit host=ip-172-31-17-181
2017-05-06T23:36:15.008+0000 I CONTROL  [initandlisten] db version v3.4.4
2017-05-06T23:36:15.008+0000 I CONTROL  [initandlisten] git version: 88839051587
4a9debd1b6c5d36559ca86b44babd
2017-05-06T23:36:15.009+0000 I CONTROL  [initandlisten] OpenSSL version: OpenSSL
 1.0.2g  1 Mar 2016
2017-05-06T23:36:15.009+0000 I CONTROL  [initandlisten] allocator: tcmalloc
2017-05-06T23:36:15.009+0000 I CONTROL  [initandlisten] modules: none
2017-05-06T23:36:15.009+0000 I CONTROL  [initandlisten] build environment:
2017-05-06T23:36:15.009+0000 I CONTROL  [initandlisten]     distmod: ubuntu1604
2017-05-06T23:36:15.009+0000 I CONTROL  [initandlisten]     distarch: x86_64
2017-05-06T23:36:15.009+0000 I CONTROL  [initandlisten]     target_arch: x86_64
2017-05-06T23:36:15.009+0000 I CONTROL  [initandlisten] options: { net: { port:
29021 }, replication: { oplogSizeMB: 128, replSet: "repl1" }, sharding: { cluste
rRole: "shardsvr" }, storage: { dbPath: "/db/shard1copy1/data", mmapv1: { smallF
iles: true } } }
2017-05-06T23:36:15.033+0000 I STORAGE  [initandlisten]
2017-05-06T23:36:15.033+0000 I STORAGE  [initandlisten] ** WARNING: Using the XF
S filesystem is strongly recommended with the WiredTiger storage engine
```

Step 4: On InstanceA, getting the shard server of shard1 replica2 up on port 29022.

$ sudo mongod --port 29022 -dbpath /db/shard1copy2/data --shardsvr --replSet repl1 --smallfiles --oplogSize 128

```
ubuntu@ip-172-31-17-181: ~
ubuntu@ip-172-31-17-181:~$ sudo mongod --port 29022 -dbpath /db/shard1copy2/data
 --shardsvr --replSet repl1 --smallfiles --oplogSize 128
2017-05-06T23:37:57.751+0000 I CONTROL  [initandlisten] MongoDB starting : pid=1
699 port=29022 dbpath=/db/shard1copy2/data 64-bit host=ip-172-31-17-181
2017-05-06T23:37:57.752+0000 I CONTROL  [initandlisten] db version v3.4.4
2017-05-06T23:37:57.752+0000 I CONTROL  [initandlisten] git version: 88839051587
4a9debd1b6c5d36559ca86b44babd
2017-05-06T23:37:57.752+0000 I CONTROL  [initandlisten] OpenSSL version: OpenSSL
 1.0.2g  1 Mar 2016
2017-05-06T23:37:57.752+0000 I CONTROL  [initandlisten] allocator: tcmalloc
2017-05-06T23:37:57.752+0000 I CONTROL  [initandlisten] modules: none
2017-05-06T23:37:57.752+0000 I CONTROL  [initandlisten] build environment:
2017-05-06T23:37:57.752+0000 I CONTROL  [initandlisten]     distmod: ubuntu1604
2017-05-06T23:37:57.752+0000 I CONTROL  [initandlisten]     distarch: x86_64
2017-05-06T23:37:57.752+0000 I CONTROL  [initandlisten]     target_arch: x86_64
2017-05-06T23:37:57.752+0000 I CONTROL  [initandlisten] options: { net: { port:
29022 }, replication: { oplogSizeMB: 128, replSet: "repl1" }, sharding: { cluste
rRole: "shardsvr" }, storage: { dbPath: "/db/shard1copy2/data", mmapv1: { smallF
iles: true } } }
2017-05-06T23:37:57.777+0000 I STORAGE  [initandlisten]
2017-05-06T23:37:57.777+0000 I STORAGE  [initandlisten] ** WARNING: Using the XF
S filesystem is strongly recommended with the WiredTiger storage engine
2017-05-06T23:37:57.777+0000 I STORAGE  [initandlisten] **          See http://d
ochub.mongodb.org/core/prodnotes-filesystem
```

Step 5: On InstanceA, getting the shard server of shard1 replica3 up on port 29023.
$ sudo mongod --port 29023 -dbpath /db/shard1copy3/data --shardsvr --replSet repl1 --smallfiles --oplogSize 128

```
ubuntu@ip-172-31-17-181: ~
ubuntu@ip-172-31-17-181:~$ sudo mongod --port 29023 -dbpath /db/shard1copy3/data
 --shardsvr --replSet repl1 --smallfiles --oplogSize 128
2017-05-06T23:38:50.300+0000 I CONTROL  [initandlisten] MongoDB starting : pid=1
771 port=29023 dbpath=/db/shard1copy3/data 64-bit host=ip-172-31-17-181
2017-05-06T23:38:50.300+0000 I CONTROL  [initandlisten] db version v3.4.4
2017-05-06T23:38:50.300+0000 I CONTROL  [initandlisten] git version: 88839051587
4a9debd1b6c5d36559ca86b44babd
2017-05-06T23:38:50.300+0000 I CONTROL  [initandlisten] OpenSSL version: OpenSSL
 1.0.2g  1 Mar 2016
2017-05-06T23:38:50.300+0000 I CONTROL  [initandlisten] allocator: tcmalloc
2017-05-06T23:38:50.300+0000 I CONTROL  [initandlisten] modules: none
2017-05-06T23:38:50.300+0000 I CONTROL  [initandlisten] build environment:
2017-05-06T23:38:50.300+0000 I CONTROL  [initandlisten]     distmod: ubuntu1604
2017-05-06T23:38:50.300+0000 I CONTROL  [initandlisten]     distarch: x86_64
2017-05-06T23:38:50.300+0000 I CONTROL  [initandlisten]     target_arch: x86_64
2017-05-06T23:38:50.300+0000 I CONTROL  [initandlisten] options: { net: { port:
29023 }, replication: { oplogSizeMB: 128, replSet: "repl1" }, sharding: { cluste
rRole: "shardsvr" }, storage: { dbPath: "/db/shard1copy3/data", mmapv1: { smallF
iles: true } } }
2017-05-06T23:38:50.325+0000 I STORAGE  [initandlisten]
2017-05-06T23:38:50.325+0000 I STORAGE  [initandlisten] ** WARNING: Using the XF
S filesystem is strongly recommended with the WiredTiger storage engine
2017-05-06T23:38:50.325+0000 I STORAGE  [initandlisten] **          See http://d
ochub.mongodb.org/core/prodnotes-filesystem
```

Step 6: Connecting to one of the shard servers on Instance A

$ mongo 172.31.17.181:29021

```
ubuntu@ip-172-31-17-181: ~
ubuntu@ip-172-31-17-181:~$ mongo 172.31.17.181:29021
MongoDB shell version v3.4.4
connecting to: 172.31.17.181:29021
MongoDB server version: 3.4.4
Server has startup warnings:
2017-05-06T23:36:15.033+0000 I STORAGE  [initandlisten]
2017-05-06T23:36:15.033+0000 I STORAGE  [initandlisten] ** WARNING: Using the XF
S filesystem is strongly recommended with the WiredTiger storage engine
2017-05-06T23:36:15.033+0000 I STORAGE  [initandlisten] **          See http://d
ochub.mongodb.org/core/prodnotes-filesystem
2017-05-06T23:36:15.077+0000 I CONTROL  [initandlisten]
2017-05-06T23:36:15.077+0000 I CONTROL  [initandlisten] ** WARNING: Access contr
ol is not enabled for the database.
2017-05-06T23:36:15.077+0000 I CONTROL  [initandlisten] **          Read and wri
te access to data and configuration is unrestricted.
2017-05-06T23:36:15.077+0000 I CONTROL  [initandlisten] ** WARNING: You are runn
ing this process as the root user, which is not recommended.
2017-05-06T23:36:15.077+0000 I CONTROL  [initandlisten]
2017-05-06T23:36:15.078+0000 I CONTROL  [initandlisten]
2017-05-06T23:36:15.078+0000 I CONTROL  [initandlisten] ** WARNING: /sys/kernel/
mm/transparent_hugepage/enabled is 'always'.
2017-05-06T23:36:15.078+0000 I CONTROL  [initandlisten] **        We suggest set
ting it to 'never'
2017-05-06T23:36:15.078+0000 I CONTROL  [initandlisten]
```

Step 7: Initiating the primary and adding the other two shard servers to the replicaset 1

$ rs.initiate()

$ rs.add("172.31.17.181:29022”)

$ rs.add("172.31.17.181:29023”)

```
ubuntu@ip-172-31-17-181: ~
2017-05-06T23:36:15.078+0000 I CONTROL  [initandlisten] **        We suggest set
ting it to 'never'
2017-05-06T23:36:15.078+0000 I CONTROL  [initandlisten]
2017-05-06T23:36:15.078+0000 I CONTROL  [initandlisten] ** WARNING: /sys/kernel/
mm/transparent_hugepage/defrag is 'always'.
2017-05-06T23:36:15.078+0000 I CONTROL  [initandlisten] **        We suggest set
ting it to 'never'
2017-05-06T23:36:15.078+0000 I CONTROL  [initandlisten]
> rs.initiate
function (c) {
    return db._adminCommand({replSetInitiate: c});
}
> rs.initiate()
{
        "info2" : "no configuration specified. Using a default configuration for
 the set",
        "me" : "ip-172-31-17-181:29021",
        "ok" : 1
}
repl1:SECONDARY> rs.add("172.31.17.181:29022")
{ "ok" : 1 }
repl1:PRIMARY> rs.add("172.31.17.181:29023")
{ "ok" : 1 }
repl1:PRIMARY>
```

Step 8: rs.status() for the replica set1 which is on the Instance A

$ rs.status()

ubuntu@ip-172-31-17-181: ~

```
repl1:PRIMARY> rs.status()
{
        "set" : "repl1",
        "date" : ISODate("2017-05-06T23:44:35.760Z"),
        "myState" : 1,
        "term" : NumberLong(1),
        "heartbeatIntervalMillis" : NumberLong(2000),
        "optimes" : {
                "lastCommittedOpTime" : {
                        "ts" : Timestamp(1494114275, 1),
                        "t" : NumberLong(1)
                },
                "appliedOpTime" : {
                        "ts" : Timestamp(1494114275, 1),
                        "t" : NumberLong(1)
                },
                "durableOpTime" : {
                        "ts" : Timestamp(1494114275, 1),
                        "t" : NumberLong(1)
                }
        },
        "members" : [
                {
                        "_id" : 0,
                        "name" : "ip-172-31-17-181:29021",
                        "health" : 1,
                        "state" : 1,
                        "stateStr" : "PRIMARY",
                        "uptime" : 501,
                        "optime" : {
                                "ts" : Timestamp(1494114275, 1),
                                "t" : NumberLong(1)
                        },
                        "optimeDate" : ISODate("2017-05-06T23:44:35Z"),
                        "electionTime" : Timestamp(1494114144, 2),
                        "electionDate" : ISODate("2017-05-06T23:42:24Z"),
                        "configVersion" : 3,
                        "self" : true
                },
                {
                        "_id" : 1,
                        "name" : "172.31.17.181:29022",
                        "health" : 1,
                        "state" : 2,
                        "stateStr" : "SECONDARY",
```

Step 9: Making 3 directories on Instance B (172.31.29.113). One for each of the 3 replicas of Shard2.

$ sudo mkdir -p /db/shard2copy1/data /db/shard2copy2/data /db/shard2copy3/data

ubuntu@ip-172-31-17-181: ~

```
Using username "ubuntu".
Authenticating with public key "imported-openssh-key"
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.4.0-1013-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud

42 packages can be updated.
22 updates are security updates.


Last login: Sat May  6 23:39:58 2017 from 130.65.254.8
ubuntu@ip-172-31-17-181:~$ sudo mkdir -p /db/configs/data1 /db/configs/data2 /db
/configs/data3
ubuntu@ip-172-31-17-181:~$
```

Step 10: On InstanceB, getting the shard server of shard2 replica1 up on port 29021.
$ sudo mongod --port 29021 -dbpath /db/shard2copy1/data --shardsvr --replSet repl2 --smallfiles --oplogSize 128

```
ubuntu@ip-172-31-29-113: ~
ubuntu@ip-172-31-29-113:~$ sudo mongod --port 29021 -dbpath /db/shard2copy1/data
 --shardsvr --replSet repl2 --smallfiles --oplogSize 128
2017-05-06T23:49:09.497+0000 I CONTROL  [initandlisten] MongoDB starting : pid=9
297 port=29021 dbpath=/db/shard2copy1/data 64-bit host=ip-172-31-29-113
2017-05-06T23:49:09.497+0000 I CONTROL  [initandlisten] db version v3.4.4
2017-05-06T23:49:09.497+0000 I CONTROL  [initandlisten] git version: 88839051587
4a9debd1b6c5d36559ca86b44babd
2017-05-06T23:49:09.497+0000 I CONTROL  [initandlisten] OpenSSL version: OpenSSL
 1.0.2g  1 Mar 2016
2017-05-06T23:49:09.497+0000 I CONTROL  [initandlisten] allocator: tcmalloc
2017-05-06T23:49:09.497+0000 I CONTROL  [initandlisten] modules: none
2017-05-06T23:49:09.497+0000 I CONTROL  [initandlisten] build environment:
2017-05-06T23:49:09.497+0000 I CONTROL  [initandlisten]     distmod: ubuntu1604
2017-05-06T23:49:09.497+0000 I CONTROL  [initandlisten]     distarch: x86_64
2017-05-06T23:49:09.497+0000 I CONTROL  [initandlisten]     target_arch: x86_64
2017-05-06T23:49:09.497+0000 I CONTROL  [initandlisten] options: { net: { port:
29021 }, replication: { oplogSizeMB: 128, replSet: "repl2" }, sharding: { cluste
rRole: "shardsvr" }, storage: { dbPath: "/db/shard2copy1/data", mmapv1: { smallF
iles: true } } }
2017-05-06T23:49:09.521+0000 I STORAGE  [initandlisten]
2017-05-06T23:49:09.521+0000 I STORAGE  [initandlisten] ** WARNING: Using the XF
S filesystem is strongly recommended with the WiredTiger storage engine
2017-05-06T23:49:09.521+0000 I STORAGE  [initandlisten] **          See http://d
ochub.mongodb.org/core/prodnotes-filesystem
```

Step 11: On InstanceB, getting the shard server of shard2 replica2 up on port 29022.
$ sudo mongod --port 29022 -dbpath /db/shard2copy2/data --shardsvr --replSet repl2 --smallfiles --oplogSize 128

```
ubuntu@ip-172-31-29-113: ~
ubuntu@ip-172-31-29-113:~$ sudo mongod --port 29022 -dbpath /db/shard2copy2/data
 --shardsvr --replSet repl2 --smallfiles --oplogSize 128
2017-05-06T23:50:15.247+0000 I CONTROL  [initandlisten] MongoDB starting : pid=9
369 port=29022 dbpath=/db/shard2copy2/data 64-bit host=ip-172-31-29-113
2017-05-06T23:50:15.247+0000 I CONTROL  [initandlisten] db version v3.4.4
2017-05-06T23:50:15.247+0000 I CONTROL  [initandlisten] git version: 88839051587
4a9debd1b6c5d36559ca86b44babd
2017-05-06T23:50:15.247+0000 I CONTROL  [initandlisten] OpenSSL version: OpenSSL
 1.0.2g  1 Mar 2016
2017-05-06T23:50:15.247+0000 I CONTROL  [initandlisten] allocator: tcmalloc
2017-05-06T23:50:15.247+0000 I CONTROL  [initandlisten] modules: none
2017-05-06T23:50:15.247+0000 I CONTROL  [initandlisten] build environment:
2017-05-06T23:50:15.247+0000 I CONTROL  [initandlisten]     distmod: ubuntu1604
2017-05-06T23:50:15.248+0000 I CONTROL  [initandlisten]     distarch: x86_64
2017-05-06T23:50:15.248+0000 I CONTROL  [initandlisten]     target_arch: x86_64
2017-05-06T23:50:15.248+0000 I CONTROL  [initandlisten] options: { net: { port:
29022 }, replication: { oplogSizeMB: 128, replSet: "repl2" }, sharding: { cluste
rRole: "shardsvr" }, storage: { dbPath: "/db/shard2copy2/data", mmapv1: { smallF
iles: true } } }
2017-05-06T23:50:15.272+0000 I STORAGE  [initandlisten]
2017-05-06T23:50:15.272+0000 I STORAGE  [initandlisten] ** WARNING: Using the XF
S filesystem is strongly recommended with the WiredTiger storage engine
2017-05-06T23:50:15.272+0000 I STORAGE  [initandlisten] **          See http://d
ochub.mongodb.org/core/prodnotes-filesystem
```

Step 12: On InstanceB, getting the shard server of shard2 replica3 up on port 29023.
$ sudo mongod --port 29023 -dbpath /db/shard2copy3/data --shardsvr --replSet repl2 --smallfiles --oplogSize 128

```
ubuntu@ip-172-31-29-113: ~
ubuntu@ip-172-31-29-113:~$ sudo mongod --port 29023 -dbpath /db/shard2copy3/data
 --shardsvr --replSet repl2 --smallfiles --oplogSize 128
2017-05-06T23:51:21.324+0000 I CONTROL  [initandlisten] MongoDB starting : pid=9
445 port=29023 dbpath=/db/shard2copy3/data 64-bit host=ip-172-31-29-113
2017-05-06T23:51:21.324+0000 I CONTROL  [initandlisten] db version v3.4.4
2017-05-06T23:51:21.324+0000 I CONTROL  [initandlisten] git version: 88839051587
4a9debd1b6c5d36559ca86b44babd
2017-05-06T23:51:21.324+0000 I CONTROL  [initandlisten] OpenSSL version: OpenSSL
 1.0.2g  1 Mar 2016
2017-05-06T23:51:21.324+0000 I CONTROL  [initandlisten] allocator: tcmalloc
2017-05-06T23:51:21.324+0000 I CONTROL  [initandlisten] modules: none
2017-05-06T23:51:21.324+0000 I CONTROL  [initandlisten] build environment:
2017-05-06T23:51:21.324+0000 I CONTROL  [initandlisten]     distmod: ubuntu1604
2017-05-06T23:51:21.324+0000 I CONTROL  [initandlisten]     distarch: x86_64
2017-05-06T23:51:21.324+0000 I CONTROL  [initandlisten]     target_arch: x86_64
2017-05-06T23:51:21.324+0000 I CONTROL  [initandlisten] options: { net: { port:
29023 }, replication: { oplogSizeMB: 128, replSet: "repl2" }, sharding: { cluste
rRole: "shardsvr" }, storage: { dbPath: "/db/shard2copy3/data", mmapv1: { smallF
iles: true } } }
2017-05-06T23:51:21.349+0000 I STORAGE  [initandlisten]
2017-05-06T23:51:21.349+0000 I STORAGE  [initandlisten] ** WARNING: Using the XF
S filesystem is strongly recommended with the WiredTiger storage engine
2017-05-06T23:51:21.349+0000 I STORAGE  [initandlisten] **          See http://d
ochub.mongodb.org/core/prodnotes-filesystem
```

Step 13: Connecting to one of the shard servers on Instance B.
$ mongo 172.31.29.113:29021

```
ubuntu@ip-172-31-29-113: ~
ubuntu@ip-172-31-29-113:~$ mongo 172.31.29.113:29021
MongoDB shell version v3.4.4
connecting to: 172.31.29.113:29021
MongoDB server version: 3.4.4
Server has startup warnings:
2017-05-06T23:49:09.521+0000 I STORAGE  [initandlisten]
2017-05-06T23:49:09.521+0000 I STORAGE  [initandlisten] ** WARNING: Using the XF
S filesystem is strongly recommended with the WiredTiger storage engine
2017-05-06T23:49:09.521+0000 I STORAGE  [initandlisten] **          See http://d
ochub.mongodb.org/core/prodnotes-filesystem
2017-05-06T23:49:09.569+0000 I CONTROL  [initandlisten]
2017-05-06T23:49:09.570+0000 I CONTROL  [initandlisten] ** WARNING: Access contr
ol is not enabled for the database.
2017-05-06T23:49:09.570+0000 I CONTROL  [initandlisten] **          Read and wri
te access to data and configuration is unrestricted.
2017-05-06T23:49:09.570+0000 I CONTROL  [initandlisten] ** WARNING: You are runn
ing this process as the root user, which is not recommended.
2017-05-06T23:49:09.570+0000 I CONTROL  [initandlisten]
2017-05-06T23:49:09.570+0000 I CONTROL  [initandlisten]
2017-05-06T23:49:09.570+0000 I CONTROL  [initandlisten] ** WARNING: /sys/kernel/
mm/transparent_hugepage/enabled is 'always'.
2017-05-06T23:49:09.570+0000 I CONTROL  [initandlisten] **        We suggest set
ting it to 'never'
2017-05-06T23:49:09.570+0000 I CONTROL  [initandlisten]
```

Step 14: Initiating the primary and adding the other two shard servers to the replicaset 2

$ rs.initiate()

$ rs.add("172.31.29.113:29022")

$ rs.add("172.31.29.113:29023")

```
ubuntu@ip-172-31-29-113: ~
2017-05-06T23:49:09.570+0000 I CONTROL  [initandlisten]
2017-05-06T23:49:09.570+0000 I CONTROL  [initandlisten]
2017-05-06T23:49:09.570+0000 I CONTROL  [initandlisten] ** WARNING: /sys/kernel/
mm/transparent_hugepage/enabled is 'always'.
2017-05-06T23:49:09.570+0000 I CONTROL  [initandlisten] **        We suggest set
ting it to 'never'
2017-05-06T23:49:09.570+0000 I CONTROL  [initandlisten]
2017-05-06T23:49:09.570+0000 I CONTROL  [initandlisten] ** WARNING: /sys/kernel/
mm/transparent_hugepage/defrag is 'always'.
2017-05-06T23:49:09.570+0000 I CONTROL  [initandlisten] **        We suggest set
ting it to 'never'
2017-05-06T23:49:09.570+0000 I CONTROL  [initandlisten]
> rs.initiate()
{
        "info2" : "no configuration specified. Using a default configuration for
 the set",
        "me" : "ip-172-31-29-113:29021",
        "ok" : 1
}
repl2:SECONDARY> rs.add("172.31.29.113:29022")
{ "ok" : 1 }
repl2:PRIMARY> rs.add("172.31.29.113:29023")
{ "ok" : 1 }
repl2:PRIMARY>
```

Step 15: rs.status() for the replica set2 which is on the Instance B

$ rs.status()

```
ubuntu@ip-172-31-29-113: ~
repl2:PRIMARY> rs.status()
{
        "set" : "repl2",
        "date" : ISODate("2017-05-06T23:55:29.673Z"),
        "myState" : 1,
        "term" : NumberLong(1),
        "heartbeatIntervalMillis" : NumberLong(2000),
        "optimes" : {
                "lastCommittedOpTime" : {
                        "ts" : Timestamp(1494114919, 1),
                        "t" : NumberLong(1)
                },
                "appliedOpTime" : {
                        "ts" : Timestamp(1494114919, 1),
                        "t" : NumberLong(1)
                },
                "durableOpTime" : {
                        "ts" : Timestamp(1494114919, 1),
                        "t" : NumberLong(1)
                }
        },
        "members" : [
                {
                        "_id" : 0,
                        "name" : "ip-172-31-29-113:29021",
                        "health" : 1,
                        "state" : 1,
                        "stateStr" : "PRIMARY",
                        "uptime" : 380,
                        "optime" : {
                                "ts" : Timestamp(1494114919, 1),
                                "t" : NumberLong(1)
                        },
                        "optimeDate" : ISODate("2017-05-06T23:55:19Z"),
                        "electionTime" : Timestamp(1494114808, 2),
                        "electionDate" : ISODate("2017-05-06T23:53:28Z"),
                        "configVersion" : 3,
                        "self" : true
                },
                {
                        "_id" : 1,
                        "name" : "172.31.29.113:29022",
                        "health" : 1,
                        "state" : 2,
                        "stateStr" : "SECONDARY",
```

Step 16: Creating 3 directories on Instance A for config servers. One for each of the 3 config servers.

$ sudo mkdir -p /db/configs/data1 /db/configs/data2 /db/configs/data3

```
ubuntu@ip-172-31-17-181: ~
Using username "ubuntu".
Authenticating with public key "imported-openssh-key"
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.4.0-1013-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud

42 packages can be updated.
22 updates are security updates.


Last login: Sat May  6 23:39:58 2017 from 130.65.254.8
ubuntu@ip-172-31-17-181:~$ sudo mkdir -p /db/configs/data1 /db/configs/data2 /db
/configs/data3
ubuntu@ip-172-31-17-181:~$
```

Step 17: On InstanceA, getting the config server up on port 29024.

$ sudo mongod --port 29024 --dbpath /db/configs/data1 --configsvr --replSet configs

```
ubuntu@ip-172-31-17-181: ~
ubuntu@ip-172-31-17-181:~$ sudo mkdir -p /db/configs/data1 /db/configs/data2 /db
/configs/data3
ubuntu@ip-172-31-17-181:~$ sudo mongod --port 29024 --dbpath /db/configs/data1 -
-configsvr --replSet configs
2017-05-07T00:01:27.996+0000 I CONTROL  [initandlisten] MongoDB starting : pid=2
064 port=29024 dbpath=/db/configs/data1 64-bit host=ip-172-31-17-181
2017-05-07T00:01:27.996+0000 I CONTROL  [initandlisten] db version v3.4.4
2017-05-07T00:01:27.996+0000 I CONTROL  [initandlisten] git version: 88839051587
4a9debd1b6c5d36559ca86b44babd
2017-05-07T00:01:27.996+0000 I CONTROL  [initandlisten] OpenSSL version: OpenSSL
 1.0.2g  1 Mar 2016
2017-05-07T00:01:27.996+0000 I CONTROL  [initandlisten] allocator: tcmalloc
2017-05-07T00:01:27.996+0000 I CONTROL  [initandlisten] modules: none
2017-05-07T00:01:27.996+0000 I CONTROL  [initandlisten] build environment:
2017-05-07T00:01:27.996+0000 I CONTROL  [initandlisten]     distmod: ubuntu1604
2017-05-07T00:01:27.996+0000 I CONTROL  [initandlisten]     distarch: x86_64
2017-05-07T00:01:27.996+0000 I CONTROL  [initandlisten]     target_arch: x86_64
2017-05-07T00:01:27.996+0000 I CONTROL  [initandlisten] options: { net: { port:
29024 }, replication: { replSet: "configs" }, sharding: { clusterRole: "configsv
r" }, storage: { dbPath: "/db/configs/data1" } }
2017-05-07T00:01:28.023+0000 I STORAGE  [initandlisten]
2017-05-07T00:01:28.023+0000 I STORAGE  [initandlisten] ** WARNING: Using the XF
S filesystem is strongly recommended with the WiredTiger storage engine
2017-05-07T00:01:28.023+0000 I STORAGE  [initandlisten] **          See http://d
```

Step 18: On InstanceA, getting the config server up on port 29025.
$ sudo mongod --port 29025 --dbpath /db/configs/data2 --configsvr --replSet configs

```
ubuntu@ip-172-31-17-181: ~
ubuntu@ip-172-31-17-181:~$ sudo mongod --port 29025 --dbpath /db/configs/data2 -
-configsvr --replSet configs
2017-05-07T00:02:30.214+0000 I CONTROL  [initandlisten] MongoDB starting : pid=2
144 port=29025 dbpath=/db/configs/data2 64-bit host=ip-172-31-17-181
2017-05-07T00:02:30.214+0000 I CONTROL  [initandlisten] db version v3.4.4
2017-05-07T00:02:30.214+0000 I CONTROL  [initandlisten] git version: 88839051587
4a9debd1b6c5d36559ca86b44babd
2017-05-07T00:02:30.214+0000 I CONTROL  [initandlisten] OpenSSL version: OpenSSL
 1.0.2g  1 Mar 2016
2017-05-07T00:02:30.214+0000 I CONTROL  [initandlisten] allocator: tcmalloc
2017-05-07T00:02:30.214+0000 I CONTROL  [initandlisten] modules: none
2017-05-07T00:02:30.214+0000 I CONTROL  [initandlisten] build environment:
2017-05-07T00:02:30.214+0000 I CONTROL  [initandlisten]     distmod: ubuntu1604
2017-05-07T00:02:30.214+0000 I CONTROL  [initandlisten]     distarch: x86_64
2017-05-07T00:02:30.214+0000 I CONTROL  [initandlisten]     target_arch: x86_64
2017-05-07T00:02:30.214+0000 I CONTROL  [initandlisten] options: { net: { port:
29025 }, replication: { replSet: "configs" }, sharding: { clusterRole: "configsv
r" }, storage: { dbPath: "/db/configs/data2" } }
2017-05-07T00:02:30.239+0000 I STORAGE  [initandlisten]
2017-05-07T00:02:30.239+0000 I STORAGE  [initandlisten] ** WARNING: Using the XF
S filesystem is strongly recommended with the WiredTiger storage engine
2017-05-07T00:02:30.239+0000 I STORAGE  [initandlisten] **          See http://d
ochub.mongodb.org/core/prodnotes-filesystem
2017-05-07T00:02:30.239+0000 I STORAGE  [initandlisten] wiredtiger_open config:
```

Step 19: On InstanceA, getting the config server up on port 29026.
$ sudo mongod --port 29026 --dbpath /db/configs/data3 --configsvr --replSet configs

```
ubuntu@ip-172-31-17-181: ~
ubuntu@ip-172-31-17-181:~$ sudo mongod --port 29026 --dbpath /db/configs/data3 -
-configsvr --replSet configs
2017-05-07T00:03:19.687+0000 I CONTROL  [initandlisten] MongoDB starting : pid=2
224 port=29026 dbpath=/db/configs/data3 64-bit host=ip-172-31-17-181
2017-05-07T00:03:19.687+0000 I CONTROL  [initandlisten] db version v3.4.4
2017-05-07T00:03:19.687+0000 I CONTROL  [initandlisten] git version: 88839051587
4a9debd1b6c5d36559ca86b44babd
2017-05-07T00:03:19.687+0000 I CONTROL  [initandlisten] OpenSSL version: OpenSSL
 1.0.2g  1 Mar 2016
2017-05-07T00:03:19.687+0000 I CONTROL  [initandlisten] allocator: tcmalloc
2017-05-07T00:03:19.687+0000 I CONTROL  [initandlisten] modules: none
2017-05-07T00:03:19.687+0000 I CONTROL  [initandlisten] build environment:
2017-05-07T00:03:19.687+0000 I CONTROL  [initandlisten]     distmod: ubuntu1604
2017-05-07T00:03:19.687+0000 I CONTROL  [initandlisten]     distarch: x86_64
2017-05-07T00:03:19.687+0000 I CONTROL  [initandlisten]     target_arch: x86_64
2017-05-07T00:03:19.687+0000 I CONTROL  [initandlisten] options: { net: { port:
29026 }, replication: { replSet: "configs" }, sharding: { clusterRole: "configsv
r" }, storage: { dbPath: "/db/configs/data3" } }
2017-05-07T00:03:19.712+0000 I STORAGE  [initandlisten]
2017-05-07T00:03:19.712+0000 I STORAGE  [initandlisten] ** WARNING: Using the XF
S filesystem is strongly recommended with the WiredTiger storage engine
2017-05-07T00:03:19.712+0000 I STORAGE  [initandlisten] **          See http://d
ochub.mongodb.org/core/prodnotes-filesystem
2017-05-07T00:03:19.712+0000 I STORAGE  [initandlisten] wiredtiger_open config:
```

Step 20: Connecting to one of the config servers on Instance A
$ mongo 172.31.17.181:29024

```
ubuntu@ip-172-31-17-181: ~
ubuntu@ip-172-31-17-181:~$ mongo 172.31.17.181:29024
MongoDB shell version v3.4.4
connecting to: 172.31.17.181:29024
MongoDB server version: 3.4.4
Server has startup warnings:
2017-05-07T00:01:28.023+0000 I STORAGE  [initandlisten]
2017-05-07T00:01:28.023+0000 I STORAGE  [initandlisten] ** WARNING: Using the XF
S filesystem is strongly recommended with the WiredTiger storage engine
2017-05-07T00:01:28.023+0000 I STORAGE  [initandlisten] **          See http://d
ochub.mongodb.org/core/prodnotes-filesystem
2017-05-07T00:01:28.079+0000 I CONTROL  [initandlisten]
2017-05-07T00:01:28.079+0000 I CONTROL  [initandlisten] ** WARNING: Access contr
ol is not enabled for the database.
2017-05-07T00:01:28.079+0000 I CONTROL  [initandlisten] **          Read and wri
te access to data and configuration is unrestricted.
2017-05-07T00:01:28.079+0000 I CONTROL  [initandlisten] ** WARNING: You are runn
ing this process as the root user, which is not recommended.
2017-05-07T00:01:28.079+0000 I CONTROL  [initandlisten]
2017-05-07T00:01:28.079+0000 I CONTROL  [initandlisten]
2017-05-07T00:01:28.079+0000 I CONTROL  [initandlisten] ** WARNING: /sys/kernel/
mm/transparent_hugepage/enabled is 'always'.
2017-05-07T00:01:28.079+0000 I CONTROL  [initandlisten] **        We suggest set
ting it to 'never'
2017-05-07T00:01:28.079+0000 I CONTROL  [initandlisten]
```

Step 21: Initiating the primary and adding the other two config servers to the 'configs' replicaset
$ rs.initiate()
$ rs.add("172.31.17.181:29025”)
$ rs.add("172.31.17.181:29026”)

```
> rs.initiate()
{
        "info2" : "no configuration specified. Using a default configuration for
 the set",
        "me" : "ip-172-31-17-181:29024",
        "ok" : 1
}
configs:SECONDARY> rs.add("172.31.17.181:29025")
{ "ok" : 1 }
configs:PRIMARY> rs.add("172.31.17.181:29026")
{ "ok" : 1 }
configs:PRIMARY>
```

Step 22: rs.status() for the 'configs' replica set
$ rs.status()

```
ubuntu@ip-172-31-17-181: ~
configs:PRIMARY> rs.status()
{
        "set" : "configs",
        "date" : ISODate("2017-05-07T00:09:31.942Z"),
        "myState" : 1,
        "term" : NumberLong(1),
        "configsvr" : true,
        "heartbeatIntervalMillis" : NumberLong(2000),
        "optimes" : {
                "lastCommittedOpTime" : {
                        "ts" : Timestamp(1494115768, 1),
                        "t" : NumberLong(1)
                },
                "readConcernMajorityOpTime" : {
                        "ts" : Timestamp(1494115768, 1),
                        "t" : NumberLong(1)
                },
                "appliedOpTime" : {
                        "ts" : Timestamp(1494115768, 1),
                        "t" : NumberLong(1)
                },
                "durableOpTime" : {
                        "ts" : Timestamp(1494115768, 1),
                        "t" : NumberLong(1)
                }
        },
        "members" : [
                {
                        "_id" : 0,
                        "name" : "ip-172-31-17-181:29024",
                        "health" : 1,
                        "state" : 1,
                        "stateStr" : "PRIMARY",
                        "uptime" : 484,
                        "optime" : {
                                "ts" : Timestamp(1494115768, 1),
                                "t" : NumberLong(1)
                        },
                        "optimeDate" : ISODate("2017-05-07T00:09:28Z"),
                        "infoMessage" : "could not find member to sync from",
                        "electionTime" : Timestamp(1494115712, 2),
                        "electionDate" : ISODate("2017-05-07T00:08:32Z"),
                        "configVersion" : 3,
                        "self" : true
                },
```

Step 23: Connecting mongos to the config servers

$ mongos --configdb configs/172.31.17.181:29024,172.31.17.181:29025,172.31.17.181:29026 --port 29031

```
ubuntu@ip-172-31-17-181: ~
ubuntu@ip-172-31-17-181:~$ mongos --configdb configs/172.31.17.181:29024,172.31.17.181:29025,172.31.17.181:29026 --port 29031
2017-05-07T00:20:22.952+0000 I CONTROL  [main]
2017-05-07T00:20:22.952+0000 I CONTROL  [main] ** WARNING: Access control is not enabled for the database.
2017-05-07T00:20:22.952+0000 I CONTROL  [main] **          Read and write access to data and configuration is unrestricted.
2017-05-07T00:20:22.952+0000 I CONTROL  [main]
2017-05-07T00:20:22.952+0000 I SHARDING [mongosMain] mongos version v3.4.4
2017-05-07T00:20:22.952+0000 I CONTROL  [mongosMain] git version: 888390515874a9debd1b6c5d36559ca86b44babd
2017-05-07T00:20:22.952+0000 I CONTROL  [mongosMain] OpenSSL version: OpenSSL 1.0.2g  1 Mar 2016
2017-05-07T00:20:22.952+0000 I CONTROL  [mongosMain] allocator: tcmalloc
2017-05-07T00:20:22.952+0000 I CONTROL  [mongosMain] modules: none
2017-05-07T00:20:22.952+0000 I CONTROL  [mongosMain] build environment:
2017-05-07T00:20:22.952+0000 I CONTROL  [mongosMain]     distmod: ubuntu1604
2017-05-07T00:20:22.952+0000 I CONTROL  [mongosMain]     distarch: x86_64
2017-05-07T00:20:22.952+0000 I CONTROL  [mongosMain]     target_arch: x86_64
2017-05-07T00:20:22.952+0000 I CONTROL  [mongosMain] db version v3.4.4
2017-05-07T00:20:22.952+0000 I CONTROL  [mongosMain] git version: 888390515874a9debd1b6c5d36559ca86b44babd
2017-05-07T00:20:22.952+0000 I CONTROL  [mongosMain] OpenSSL version: OpenSSL 1.0.2g  1 Mar 2016
2017-05-07T00:20:22.952+0000 I CONTROL  [mongosMain] allocator: tcmalloc
2017-05-07T00:20:22.952+0000 I CONTROL  [mongosMain] modules: none
2017-05-07T00:20:22.952+0000 I CONTROL  [mongosMain] build environment:
2017-05-07T00:20:22.952+0000 I CONTROL  [mongosMain]     distmod: ubuntu1604
2017-05-07T00:20:22.952+0000 I CONTROL  [mongosMain]     distarch: x86_64
2017-05-07T00:20:22.952+0000 I CONTROL  [mongosMain]     target_arch: x86_64
2017-05-07T00:20:22.952+0000 I CONTROL  [mongosMain] options: { net: { port: 29031 }, sharding: { configDB: "configs/172.31.17.181:29024,172.31.17.181:29025,
172.31.17.181:29026" } }
2017-05-07T00:20:22.979+0000 I NETWORK  [mongosMain] Starting new replica set monitor for configs/172.31.17.181:29024,172.31.17.181:29025,172.31.17.181:29026
2017-05-07T00:20:22.979+0000 I SHARDING [thread1] creating distributed lock ping thread for process ip-172-31-17-181:29031:1494116422:579031327866630134 7 (sl
eeping for 30000ms)
2017-05-07T00:20:22.980+0000 I NETWORK  [mongosMain] changing hosts to configs/172.31.17.181:29025,172.31.17.181:29026,ip-172-31-17-181:29024 from configs/17
2.31.17.181:29024,172.31.17.181:29025,172.31.17.181:29026
2017-05-07T00:20:22.980+0000 I SHARDING [mongosMain] Updating ShardRegistry connection string for shard config from: configs/172.31.17.181:29024,172.31.17.18
1:29025,172.31.17.181:29026 to: configs/172.31.17.181:29025,172.31.17.181:29026,ip-172-31-17-181:29024
2017-05-07T00:20:22.980+0000 I ASIO     [NetworkInterfaceASIO-ShardRegistry-0] Connecting to 172.31.17.181:29026
2017-05-07T00:20:22.980+0000 I ASIO     [NetworkInterfaceASIO-ShardRegistry-0] Connecting to 172.31.17.181:29025
2017-05-07T00:20:22.981+0000 I ASIO     [NetworkInterfaceASIO-ShardRegistry-0] Successfully connected to 172.31.17.181:29026, took 1ms (1 connections now ope
n to 172.31.17.181:29026)
2017-05-07T00:20:22.981+0000 I ASIO     [NetworkInterfaceASIO-ShardRegistry-0] Successfully connected to 172.31.17.181:29025, took 1ms (1 connections now ope
n to 172.31.17.181:29025)
2017-05-07T00:20:22.982+0000 I ASIO     [NetworkInterfaceASIO-ShardRegistry-0] Connecting to ip-172-31-17-181:29024
2017-05-07T00:20:22.982+0000 I ASIO     [NetworkInterfaceASIO-ShardRegistry-0] Connecting to ip-172-31-17-181:29024
2017-05-07T00:20:22.982+0000 I ASIO     [NetworkInterfaceASIO-ShardRegistry-0] Connecting to ip-172-31-17-181:29024
2017-05-07T00:20:22.991+0000 I ASIO     [NetworkInterfaceASIO-ShardRegistry-0] Successfully connected to ip-172-31-17-181:29024, took 9ms (3 connections now
open to ip-172-31-17-181:29024)
2017-05-07T00:20:22.991+0000 I ASIO     [NetworkInterfaceASIO-ShardRegistry-0] Successfully connected to ip-172-31-17-181:29024, took 9ms (3 connections now
open to ip-172-31-17-181:29024)
```

Step 24: Opening mongos shell
$ mongo 172.31.17.181:29031

```
ubuntu@ip-172-31-17-181: ~
 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud

42 packages can be updated.
22 updates are security updates.


Last login: Sun May  7 00:10:26 2017 from 130.65.254.8
ubuntu@ip-172-31-17-181:~$ mongo 172.31.17.181:29031
MongoDB shell version v3.4.4
connecting to: 172.31.17.181:29031
MongoDB server version: 3.4.4
Server has startup warnings:
2017-05-07T00:20:22.952+0000 I CONTROL  [main]
2017-05-07T00:20:22.952+0000 I CONTROL  [main] ** WARNING: Access control is not
 enabled for the database.
2017-05-07T00:20:22.952+0000 I CONTROL  [main] **          Read and write access
 to data and configuration is unrestricted.
2017-05-07T00:20:22.952+0000 I CONTROL  [main]
mongos>
```

Step 25: Adding all 6 shards to mongos (3 on Instance A which are in replica set1 and 3 on Instance B which are on replics set 2)
$ sh.addShard("repl1/172.31.17.181:29021,172.31.17.181:29022,172.31.17.181:29023”)
$ sh.addShard("repl2/ip-172-31-29-113:29021,172.31.29.113:29022,172.31.29.113:29023")

```
mongos> sh.addShard("repl1/ip-172-31-17-181:29021,172.31.17.181:29022,172.31.17.
181:29023")
{ "shardAdded" : "repl1", "ok" : 1 }
mongos> sh.addShard("repl2/ip-172-31-29-113:29021,172.31.29.113:29022,172.31.29.
113:29023")
{ "shardAdded" : "repl2", "ok" : 1 }
mongos>
```

Step 26:sh.status() after adding the shards
> sh.status()

```
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "minCompatibleVersion" : 5,
        "currentVersion" : 6,
        "clusterId" : ObjectId("590e65816ce5a3f3be8d1d0c")
}
  shards:
        {  "_id" : "repl1",  "host" : "repl1/172.31.17.181:29022,172.31.17.181:2
9023,ip-172-31-17-181:29021",  "state" : 1 }
        {  "_id" : "repl2",  "host" : "repl2/172.31.29.113:29022,172.31.29.113:2
9023,ip-172-31-29-113:29021",  "state" : 1 }
  active mongoses:
        "3.4.4" : 1
 autosplit:
        Currently enabled: yes
  balancer:
        Currently enabled:  yes
        Currently running:  no
                Balancer lock taken at Sun May 07 2017 00:08:33 GMT+0000 (UTC) b
y ConfigServer:Balancer
        Failed balancer rounds in last 5 attempts:  0
        Migration Results for the last 24 hours:
                No recent migrations
  databases:

mongos>
```

Step 27: Enabling Sharding

>sh.enableSharding("airstats")

```
mongos> sh.enableSharding("airstats")
{ "ok" : 1 }
mongos> sh.shardStatus( {verbose:1} )
2017-05-07T00:45:27.132+0000 E QUERY    [thread1] TypeError: sh.shardStatus is n
ot a function :
@(shell):1:1
mongos> sh.status( {verbose:1} )
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "minCompatibleVersion" : 5,
        "currentVersion" : 6,
        "clusterId" : ObjectId("590e65816ce5a3f3be8d1d0c")
}
  shards:
        {  "_id" : "repl1",  "host" : "repl1/172.31.17.181:29022,172.31.17.181:2
9023,ip-172-31-17-181:29021",  "state" : 1 }
        {  "_id" : "repl2",  "host" : "repl2/172.31.29.113:29022,172.31.29.113:2
9023,ip-172-31-29-113:29021",  "state" : 1 }
  active mongoses:
        {  "_id" : "ip-172-31-17-181:29031",  "ping" : ISODate("2017-05-07T00:45
:53.554Z"),  "up" : NumberLong(1530),  "waiting" : true,  "mongoVersion" : "3.4.
4" }
 autosplit:
        Currently enabled: yes
  balancer:
        Currently enabled:  yes
        Currently running:  no
                Balancer lock taken at Sun May 07 2017 00:08:33 GMT+0000 (UTC) b
y ConfigServer:Balancer
        Failed balancer rounds in last 5 attempts:  0
        Migration Results for the last 24 hours:
                No recent migrations
  databases:
        {  "_id" : "airstats",  "primary" : "repl2",  "partitioned" : true }

mongos>
```

Step 28: Using simple Index on ORIGIN

>sh.status({verbose:1})

```
mongos> sh.shardCollection("airstats.flight", { ORIGIN: 1 } )
{ "collectionsharded" : "airstats.flight", "ok" : 1 }
mongos>
```

Step 29: Importing the data using mongoimport to the mongo database from csv

>sh.shardCollection("airstats.flight", { ORIGIN: 1 })

```
ubuntu@ip-172-31-17-181: ~
ubuntu@ip-172-31-17-181:~$ mongoimport --host 172.31.17.181 --port 29031 --db ai
rstats --collection flight --type csv --headerline --file /home/ubuntu/Master.cs
v
2017-05-07T00:53:50.282+0000    connected to: 172.31.17.181:29031
2017-05-07T00:53:53.276+0000    [........................] airstats.flight     8
.73MB/1.01GB (0.8%)
2017-05-07T00:53:56.276+0000    [........................] airstats.flight     1
5.9MB/1.01GB (1.5%)
2017-05-07T00:53:59.276+0000    [........................] airstats.flight      20.8MB/1.01GB (2.0%)
2017-05-07T00:54:02.276+0000    [........................] airstats.flight      25.8MB/1.01GB (2.5%)
2017-05-07T00:54:05.276+0000    [........................] airstats.flight      34.5MB/1.01GB (3.3%)
2017-05-07T00:54:08.276+0000    [........................] airstats.flight      42.9MB/1.01GB (4.2%)
2017-05-07T00:54:11.278+0000    [#.......................] airstats.flight      50.1MB/1.01GB (4.9%)
2017-05-07T00:54:14.276+0000    [#.......................] airstats.flight      54.2MB/1.01GB (5.2%)
2017-05-07T00:54:17.285+0000    [#.......................] airstats.flight      58.3MB/1.01GB (5.7%)
2017-05-07T00:54:20.295+0000    [#.......................] airstats.flight      63.2MB/1.01GB (6.1%)
2017-05-07T00:54:23.277+0000    [#.......................] airstats.flight      69.9MB/1.01GB (6.8%)
2017-05-07T00:54:26.276+0000    [#.......................] airstats.flight      77.1MB/1.01GB (7.5%)
2017-05-07T00:54:29.276+0000    [#.......................] airstats.flight      84.3MB/1.01GB (8.2%)
2017-05-07T00:54:32.276+0000    [##......................] airstats.flight      90.7MB/1.01GB (8.8%)
2017-05-07T00:54:35.276+0000    [##......................] airstats.flight      95.8MB/1.01GB (9.3%)
2017-05-07T00:54:38.277+0000    [##......................] airstats.flight      103MB/1.01GB (10.0%)
2017-05-07T00:54:41.277+0000    [##......................] airstats.flight      111MB/1.01GB (10.8%)
2017-05-07T00:54:44.276+0000    [##......................] airstats.flight      119MB/1.01GB (11.5%)
2017-05-07T00:54:47.276+0000    [##......................] airstats.flight      126MB/1.01GB (12.2%)
2017-05-07T00:54:50.276+0000    [###.....................] airstats.flight      133MB/1.01GB (12.9%)
2017-05-07T00:54:53.276+0000    [###.....................] airstats.flight      139MB/1.01GB (13.4%)
2017-05-07T00:54:56.276+0000    [###.....................] airstats.flight      144MB/1.01GB (14.0%)
2017-05-07T00:54:59.276+0000    [###.....................] airstats.flight      152MB/1.01GB (14.7%)
2017-05-07T00:55:02.276+0000    [###.....................] airstats.flight      158MB/1.01GB (15.3%)
2017-05-07T00:55:05.276+0000    [###.....................] airstats.flight      165MB/1.01GB (16.0%)
2017-05-07T00:55:08.287+0000    [####....................] airstats.flight      172MB/1.01GB (16.7%)
2017-05-07T00:55:11.276+0000    [####....................] airstats.flight      179MB/1.01GB (17.4%)
2017-05-07T00:55:14.276+0000    [####....................] airstats.flight      186MB/1.01GB (18.1%)
2017-05-07T00:55:17.276+0000    [####....................] airstats.flight      191MB/1.01GB (18.5%)
2017-05-07T00:55:20.276+0000    [####....................] airstats.flight      198MB/1.01GB (19.2%)
2017-05-07T00:55:23.276+0000    [####....................] airstats.flight      204MB/1.01GB (19.8%)
2017-05-07T00:55:26.278+0000    [####....................] airstats.flight      209MB/1.01GB (20.3%)
2017-05-07T00:55:29.276+0000    [####....................] airstats.flight      214MB/1.01GB (20.7%)
2017-05-07T00:55:32.276+0000    [#####...................] airstats.flight      220MB/1.01GB (21.3%)
2017-05-07T00:55:35.279+0000    [#####...................] airstats.flight      223MB/1.01GB (21.6%)
2017-05-07T00:55:38.276+0000    [#####...................] airstats.flight      226MB/1.01GB (21.9%)
2017-05-07T00:55:41.276+0000    [#####...................] airstats.flight      228MB/1.01GB (22.0%)
2017-05-07T00:55:44.276+0000    [#####...................] airstats.flight      230MB/1.01GB (22.3%)
2017-05-07T00:55:47.286+0000    [#####...................] airstats.flight      233MB/1.01GB (22.6%)
```

Step 30:sh.status() after sharding using the sharding key: ORIGIN

>sh.status()

```
ubuntu@ip-172-31-17-181: ~
mongos> sh.status({verbose:1})
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "minCompatibleVersion" : 5,
        "currentVersion" : 6,
        "clusterId" : ObjectId("590e65816ce5a3f3be8d1d0c")
}
  shards:
        {  "_id" : "repl1",  "host" : "repl1/172.31.17.181:29022,172.31.17.181:29023,ip-172-31-17-181:2
9021",  "state" : 1 }
        {  "_id" : "repl2",  "host" : "repl2/172.31.29.113:29022,172.31.29.113:29023,ip-172-31-29-113:2
9021",  "state" : 1 }
  active mongoses:
        {  "_id" : "ip-172-31-17-181:29031",  "ping" : ISODate("2017-05-07T02:04:40.724Z"),  "up" : Num
berLong(80),  "waiting" : true,  "mongoVersion" : "3.4.4" }
 autosplit:
        Currently enabled: yes
  balancer:
        Currently enabled:  yes
        Currently running:  no
                Balancer lock taken at Sun May 07 2017 02:02:43 GMT+0000 (UTC) by ConfigServer:Balancer
        Failed balancer rounds in last 5 attempts:  0
        Migration Results for the last 24 hours:
                7 : Success
  databases:
        {  "_id" : "airstats",  "primary" : "repl2",  "partitioned" : true }
                airstats.flight
                        shard key: { "ORIGIN" : 1 }
                        unique: false
                        balancing: true
                        chunks:
                                repl1   13
                                repl2   13
                        { "ORIGIN" : { "$minKey" : 1 } } -->> { "ORIGIN" : "ATL" } on : repl2 Timestamp
(7, 0)
                        { "ORIGIN" : "ATL" } -->> { "ORIGIN" : "BOI" } on : repl2 Timestamp(8, 5)
                        { "ORIGIN" : "BOI" } -->> { "ORIGIN" : "BUR" } on : repl2 Timestamp(8, 6)
                        { "ORIGIN" : "BUR" } -->> { "ORIGIN" : "CLT" } on : repl2 Timestamp(8, 7)
                        { "ORIGIN" : "CLT" } -->> { "ORIGIN" : "DFW" } on : repl1 Timestamp(8, 1)
                        { "ORIGIN" : "DFW" } -->> { "ORIGIN" : "EUG" } on : repl1 Timestamp(6, 4)
                        { "ORIGIN" : "EUG" } -->> { "ORIGIN" : "IAD" } on : repl1 Timestamp(6, 8)
                        { "ORIGIN" : "IAD" } -->> { "ORIGIN" : "JFK" } on : repl1 Timestamp(6, 9)
                        { "ORIGIN" : "JFK" } -->> { "ORIGIN" : "LAS" } on : repl1 Timestamp(6, 10)
                        { "ORIGIN" : "LAS" } -->> { "ORIGIN" : "LAX" } on : repl1 Timestamp(3, 4)
```

Step 31: Querying using simple mongo command

```
ubuntu@ip-172-31-17-181: ~
        Migration Results for the last 24 hours:
                7 : Success
  databases:
        {  "_id" : "airstats",  "primary" : "repl2",  "partitioned" : true }
                airstats.flight
                        shard key: { "ORIGIN" : 1 }
                        unique: false
                        balancing: true
                        chunks:
                                repl1   13
                                repl2   13
                        { "ORIGIN" : { "$minKey" : 1 } } -->> { "ORIGIN" : "ATL" } on : repl2 Timestamp
(7, 0)
                        { "ORIGIN" : "ATL" } -->> { "ORIGIN" : "BOI" } on : repl2 Timestamp(8, 5)
                        { "ORIGIN" : "BOI" } -->> { "ORIGIN" : "BUR" } on : repl2 Timestamp(8, 6)
                        { "ORIGIN" : "BUR" } -->> { "ORIGIN" : "CLT" } on : repl2 Timestamp(8, 7)
                        { "ORIGIN" : "CLT" } -->> { "ORIGIN" : "DFW" } on : repl1 Timestamp(8, 1)
                        { "ORIGIN" : "DFW" } -->> { "ORIGIN" : "EUG" } on : repl1 Timestamp(6, 4)
                        { "ORIGIN" : "EUG" } -->> { "ORIGIN" : "IAD" } on : repl1 Timestamp(6, 8)
                        { "ORIGIN" : "IAD" } -->> { "ORIGIN" : "JFK" } on : repl1 Timestamp(6, 9)
                        { "ORIGIN" : "JFK" } -->> { "ORIGIN" : "LAS" } on : repl1 Timestamp(6, 10)
                        { "ORIGIN" : "LAS" } -->> { "ORIGIN" : "LAX" } on : repl1 Timestamp(3, 4)
                        { "ORIGIN" : "LAX" } -->> { "ORIGIN" : "LGB" } on : repl1 Timestamp(4, 0)
                        { "ORIGIN" : "LGB" } -->> { "ORIGIN" : "MSP" } on : repl1 Timestamp(6, 11)
                        { "ORIGIN" : "MSP" } -->> { "ORIGIN" : "OAK" } on : repl1 Timestamp(6, 12)
                        { "ORIGIN" : "OAK" } -->> { "ORIGIN" : "ONT" } on : repl1 Timestamp(6, 13)
                        { "ORIGIN" : "ONT" } -->> { "ORIGIN" : "ORD" } on : repl1 Timestamp(8, 2)
                        { "ORIGIN" : "ORD" } -->> { "ORIGIN" : "PDX" } on : repl1 Timestamp(8, 3)
                        { "ORIGIN" : "PDX" } -->> { "ORIGIN" : "PHX" } on : repl1 Timestamp(8, 4)
                        { "ORIGIN" : "PHX" } -->> { "ORIGIN" : "SAN" } on : repl2 Timestamp(6, 1)
                        { "ORIGIN" : "SAN" } -->> { "ORIGIN" : "SEA" } on : repl2 Timestamp(1, 7)
                        { "ORIGIN" : "SEA" } -->> { "ORIGIN" : "SFO" } on : repl2 Timestamp(2, 2)
                        { "ORIGIN" : "SFO" } -->> { "ORIGIN" : "SJC" } on : repl2 Timestamp(4, 2)
                        { "ORIGIN" : "SJC" } -->> { "ORIGIN" : "SLC" } on : repl2 Timestamp(6, 5)
                        { "ORIGIN" : "SLC" } -->> { "ORIGIN" : "SMX" } on : repl2 Timestamp(6, 6)
                        { "ORIGIN" : "SMX" } -->> { "ORIGIN" : "SNA" } on : repl2 Timestamp(6, 7)
                        { "ORIGIN" : "SNA" } -->> { "ORIGIN" : "YUM" } on : repl2 Timestamp(4, 4)
                        { "ORIGIN" : "YUM" } -->> { "ORIGIN" : { "$maxKey" : 1 } } on : repl2 Timestamp
(2, 4)

mongos> use airstats;
switched to db airstats
mongos> db.flight.count()
1059533
mongos>
```

Step 33: .explain() on a query

```
ubuntu@ip-172-31-17-181: ~
mongos> db.flight.find({'ORIGIN_STATE_NM': 'Nevada'}).explain()
{
        "queryPlanner" : {
                "mongosPlannerVersion" : 1,
                "winningPlan" : {
                        "stage" : "SHARD_MERGE",
                        "shards" : [
                                {
                                        "shardName" : "repl1",
                                        "connectionString" : "repl1/172.31.17.181:29022,172.31.17.181:2
9023,ip-172-31-17-181:29021",
                                        "serverInfo" : {
                                                "host" : "ip-172-31-17-181",
                                                "port" : 29023,
                                                "version" : "3.4.4",
                                                "gitVersion" : "888390515874a9debd1b6c5d36559ca86b44bab
d"
                                        },
                                        "plannerVersion" : 1,
                                        "namespace" : "airstats.flight",
                                        "indexFilterSet" : false,
                                        "parsedQuery" : {
                                                "ORIGIN_STATE_NM" : {
                                                        "$eq" : "Nevada"
                                                }
                                        },
                                        "winningPlan" : {
                                                "stage" : "SHARDING_FILTER",
                                                "inputStage" : {
                                                        "stage" : "COLLSCAN",
                                                        "filter" : {
                                                                "ORIGIN_STATE_NM" : {
                                                                        "$eq" : "Nevada"
                                                                }
                                                        },
                                                        "direction" : "forward"
                                                }
                                        },
                                        "rejectedPlans" : [ ]
                                },
                                {
                                        "shardName" : "repl2",
                                        "connectionString" : "repl2/172.31.29.113:29022,172.31.29.113:2
9023,ip-172-31-29-113:29021",
                                        "serverInfo" : {
```

Step 32: Connecting python to mongo using pymongo

```
ubuntu@ip-172-31-17-181:~$ python
Python 2.7.12 (default, Nov 19 2016, 06:48:10)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from pymongo import MongoClient
>>> client = MongoClient('172.31.17.181', 29031)
>>> db = client.airstats
>>> collection = db['flight']
```

10. Brief description on the NoSQL driver of your choice and detailed end-to-end description from your application to the database. You may describe how to make a connection from the application to the database, how an application submits a command to the database and how it receives a result from the database for this.

Step 1: Install python and mongodb on both the aws instances.

Step 2: Pymongo Driver
NOSQL Driver that we are using is **pymongo**. Connection between python and mongo is established using Pymongo driver.
Installation steps for pymongo using pip:

$ sudo apt install python-pip
$ pip install pymongo==3.4.0
$ pip install --upgrade pymongo
import pymongo (Check if pymongo is installed properly)

Step 3: Connection to MongoClient
Start the python shell using: the command python.
We use MongoClient class in order to establish a connection to the running mongod from python using the pymongo driver.

>>>from pymongo import MongoClient
>>>client = MongoClient() (Connects to the default)
>>> client = MongoClient('172.31.17.181', 29031) (to connect to mongo running on a specific port)
>>> client = MongoClient('mongodb://172.31.17.181:29031/') (MongoDB URI format)

```
from pymongo import MongoClient
client = MongoClient('172.31.17.181', 29031)
db = client.airstats
collection = db['flight']
```

```
ubuntu@ip-172-31-17-181:~$ python
Python 2.7.12 (default, Nov 19 2016, 06:48:10)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from pymongo import MongoClient
>>> client = MongoClient('172.31.17.181', 29031)
>>> db = client.airstats
>>> collection = db['flight']
```

Step 4: Connection to Database

We use the MongoClient instance that we created to access the databases present on mongo. We can use any one of the following formats.

>>> db = client.airstats

>>> db = client[‘airstats’]

Step 5: Getting the Collection

Getting a collection from the database to which we connected is similar to connecting to the database.

>>> collection = db.flight

>>> collection = db[‘flight’]

```
db = client.airstats
collection = db['flight']
```

Step 6: Documents

Mongodb represents and handles the documents in JSON. We use the dictionaries in PyMongo to represent the documents. Once we get the connection to the collection from the step above, we can now implement the queries on this collection.

Query and Code description:

- We wrote a python script that includes all the 15 queries in one program. 15 functions were created and each function supports one query.
- A list of query descriptions is given to the user from which he/she can choose which query to run.

```
def mainPrompt():
    print('Welcome! What would you like to do today? Here are your choices:'
          '\n1  - Find the number of flights coming from a specific state'
          '\n2  - Lists the months and the average departure delay experienced in those months'
          '\n3  - Find the average flight time from flights between certain airports'
          '\n4  - Find all airport codes in a certain state'
          '\n5  - Find the probability that a flight going to an airport will be cancelled'
          '\n6  - Find the average departure delay of a certain airport'
          '\n7  - Find the average arrival delay time for a certain flight carrier'
          '\n8  - Lists the months and the average taxi out time in a given airport'
          '\n9  - Find the average delay due to weather when leaving from a certain airport'
          '\n10 - Find the number of cancelled flights in a given month and year'
          '\n11 - Find the distance in miles between two airports'
          '\n12 - Find the average delay due to late aircraft given a specific carrier and year'
          '\n13 - Find the month and delay with the longest delay given a year'
          '\n14 - Finds the destination state that most flights go to given an origin state'
          '\n15 - List the month and the carrier with the most amount of flights cancelled in a given year')
    print()
```

- We are using the if-else condition to jump to different functions based on the query description that the user passed to execute.
- We give the flexibility to the user to pass the input parameter based on which the query is executed.

```
userValue = raw_input('Enter your choice here: ')
if userValue == '1':
    value = raw_input('What state would you like to see? ')
    numOfFlightsFromState(value)
elif userValue == '2':
    getAverageDepartureDelayPerMonth()
elif userValue == '3':
    origin = raw_input('Origin: ')
    destination = raw_input('Destination: ')
    averageFlightTimeBetweenAirports(origin, destination)
elif userValue == '4':
    state = raw_input('What state would you like to see? ')
    getAirportCodesInState(state)
elif userValue == '5':
    airport = raw_input('What airport would you like to see? ')
    probabilityOfCancellation(airport)
elif userValue == '6':
    airport = raw_input('What airport would you like to see? ')
    averageDepartureDelayAtAirport(airport)
elif userValue == '7':
    carrier = raw_input('What carrier would you like to see? ')
    averageDelayTimePerFlightCarrier(carrier)
elif userValue == '8':
    airport = raw_input('What airport would you like to see? ')
    averageTaxiOutTimePerMonth(airport)
elif userValue == '9':
    airport = raw_input('What airport would you like to see? ')
    averageWeatherDelayWhenLeaving(airport)
elif userValue == '10':
    month = raw_input('Which month? ')
    year = raw_input('Which year? ')
    findNumberOfCancelledFlights(month, year)
```

```
elif userValue == '10':
    month = raw_input('Which month? ')
    year = raw_input('Which year? ')
    findNumberOfCancelledFlights(month, year)
elif userValue == '11':
    airport1 = raw_input('From airport: ')
    airport2 = raw_input('to airport: ')
    findDistanceBetweenAirports(airport1, airport2)
elif userValue == '12':
    carrier = raw_input('What carrier would you like to see? ')
    year = raw_input('Which year? ')
    averageLateAircraftDelayFromCarrier(carrier, year)
elif userValue == '13':
    year = raw_input('Which year? ')
    findTheMonthWithTheLongestDelayGivenAYear(year)
elif userValue == '14':
    state = raw_input('What state would you like to see? ')
    findMostTraveledStateFromGivenState(state)
elif userValue == '15':
    year = raw_input('What year would you like to see? ')
    listMonthAndCarrierOfMostCancelledFlights(year)
```

- Query 1:
- print('The number of flights coming from {} is {}'.format(state, collection.find({'ORIGIN_STATE_NM': state}).count()))
- We are querying on the connected collection using dictionaries to represent the document. The query is ran on the database, returns the result to pymongo and is printed on the shell.

```
# Find the number of flights coming from a specific state
def numOfFlightsFromState(state):
    print('The number of flights coming from {} is {}'.format(state, collection.find({'ORIGIN_STATE_NM': state}).count()))
```

```
ubuntu@ip-172-31-17-181:~$ python Main.py
Welcome! What would you like to do today? Here are your choices:
1  - Find the number of flights coming from a specific state
2  - Lists the months and the average departure delay experienced in those months
3  - Find the average flight time from flights between certain airports
4  - Find all airport codes in a certain state
5  - Find the probability that a flight going to an airport will be cancelled
6  - Find the average departure delay of a certain airport
7  - Find the average arrival delay time for a certain flight carrier
8  - Lists the months and the average taxi out time in a given airport
9  - Find the average delay due to weather when leaving from a certain airport
10 - Find the number of cancelled flights in a given month and year
11 - Find the distance in miles between two airports
12 - Find the average delay due to late aircraft given a specific carrier and year
13 - Find the month and delay with the longest delay given a year
14 - Finds the destination state that most flights go to given an origin state
15 - List the month and the carrier with the most amount of flights cancelled in a given year
()
Enter your choice here: 1
What state would you like to see? Nevada
The number of flights coming from Nevada is 42213
```

11. For each 15 function, include the following items

- **One line of description for that function**
- **DDL and/or DML commands of your NoSQL choice to fulfill the function**
- **Screenshot(s) to CLEARLY show the function works.**

Function 1:

a. This function gives the count of all flights from a specific state.

b. For this we used find and count functions.
   Find: finds the state.
   Count: counts the total number of flights

c.

```
# Find the number of flights coming from a specific state
def numOfFlightsFromState(state):
    print('The number of flights coming from {} is {}'.format(state, collection.find({'ORIGIN_STATE_NM': state}).count()))
```

Output:

```
ubuntu@ip-172-31-17-181:~$ python Main.py
Welcome! What would you like to do today? Here are your choices:
1  - Find the number of flights coming from a specific state
2  - Lists the months and the average departure delay experienced in those months
3  - Find the average flight time from flights between certain airports
4  - Find all airport codes in a certain state
5  - Find the probability that a flight going to an airport will be cancelled
6  - Find the average departure delay of a certain airport
7  - Find the average arrival delay time for a certain flight carrier
8  - Lists the months and the average taxi out time in a given airport
9  - Find the average delay due to weather when leaving from a certain airport
10 - Find the number of cancelled flights in a given month and year
11 - Find the distance in miles between two airports
12 - Find the average delay due to late aircraft given a specific carrier and year
13 - Find the month and delay with the longest delay given a year
14 - Finds the destination state that most flights go to given an origin state
15 - List the month and the carrier with the most amount of flights cancelled in a given year
()
Enter your choice here: 1
What state would you like to see? Nevada
The number of flights coming from Nevada is 42213
```

Function 2:

a. This function lists the months and the average departure delay experienced in those months.

b. For this we used aggregate, group, and avg function.
   Aggregate: Calculates aggregate values for data in collection.
   Group: Groups by month

c.

```
# Lists the months and the average departure delay experienced in those months
def getAverageDepartureDelayPerMonth():
    results = collection.aggregate([{'$group': {'_id': "$MONTH", 'avgDelay': {'$avg': "$DEP_DELAY"}}}, {'$sort': {'_id' : 1}}])
    print('Month |  Delay in Minutes')
    print('---------------------------')
    for result in results:
        if result['_id'] is not None:
            print("%-6d|%9.2f" % (result['_id'], result['avgDelay']))
```

Output:

```
ubuntu@ip-172-31-17-181:~$ python Main.py
Welcome! What would you like to do today? Here are your choices:
1  - Find the number of flights coming from a specific state
2  - Lists the months and the average departure delay experienced in those months
3  - Find the average flight time from flights between certain airports
4  - Find all airport codes in a certain state
5  - Find the probability that a flight going to an airport will be cancelled
6  - Find the average departure delay of a certain airport
7  - Find the average arrival delay time for a certain flight carrier
8  - Lists the months and the average taxi out time in a given airport
9  - Find the average delay due to weather when leaving from a certain airport
10 - Find the number of cancelled flights in a given month and year
11 - Find the distance in miles between two airports
12 - Find the average delay due to late aircraft given a specific carrier and year
13 - Find the month and delay with the longest delay given a year
14 - Finds the destination state that most flights go to given an origin state
15 - List the month and the carrier with the most amount of flights cancelled in a given year
()
Enter your choice here: 2
Month |  Delay in Minutes
----------------------------
1     |    12.58
2     |    14.07
3     |     8.84
4     |     8.51
5     |     9.78
6     |    12.72
7     |    11.88
8     |    10.00
9     |     6.88
10    |     7.18
12    |    12.58
```

Function 3:

a. This function finds the average flight time from flights between certain airports.
b. For this we used aggregate and group functions which groups origin and average flight times.
c.

```python
# Find the average flight time from flights between certain airports
def averageFlightTimeBetweenAirports(origin, destination):
    results = collection.aggregate([{'$match': {"ORIGIN": origin, "DEST": destination}}, {'$group': {'_id': "$ORIGIN", 'avgFlightTime': {'$avg': "$AIR_TIME"}}}])
    for result in results:
        print("The average time of flight from %s to %s is %.2f minutes" % (origin, destination, result['avgFlightTime']))
```

Output:

```
ubuntu@ip-172-31-17-181:~$ python Main.py
Welcome! What would you like to do today? Here are your choices:
1  - Find the number of flights coming from a specific state
2  - Lists the months and the average departure delay experienced in those months
3  - Find the average flight time from flights between certain airports
4  - Find all airport codes in a certain state
5  - Find the probability that a flight going to an airport will be cancelled
6  - Find the average departure delay of a certain airport
7  - Find the average arrival delay time for a certain flight carrier
8  - Lists the months and the average taxi out time in a given airport
9  - Find the average delay due to weather when leaving from a certain airport
10 - Find the number of cancelled flights in a given month and year
11 - Find the distance in miles between two airports
12 - Find the average delay due to late aircraft given a specific carrier and year
13 - Find the month and delay with the longest delay given a year
14 - Finds the destination state that most flights go to given an origin state
15 - List the month and the carrier with the most amount of flights cancelled in a given year
()
Enter your choice here: 3
Origin: SFO
Destination: LAX
The average time of flight from SFO to LAX is 54.63 minutes
```

Function 4:

a. This function finds all airport codes in a certain state.
b. For this we used aggregate and group functions which serves the query.
c.

```python
# Find all airport codes in a certain state
def getAirportCodesInState(state):
    results = collection.aggregate([{'$match': {"ORIGIN_STATE_NM": state}}, {'$group': {'_id': { 'origin' : "$ORIGIN", 'city': "$ORIGIN_CITY_NAME"}} }])
    print('Code | City, State')
    print('------------------------')
    for result in results:
        print("%-5s| %s" % (result['_id']['origin'], result['_id']['city']))
```

Output:

```
ubuntu@ip-172-31-17-181:~$ python Main.py
Welcome! What would you like to do today? Here are your choices:
1  - Find the number of flights coming from a specific state
2  - Lists the months and the average departure delay experienced in those months
3  - Find the average flight time from flights between certain airports
4  - Find all airport codes in a certain state
5  - Find the probability that a flight going to an airport will be cancelled
6  - Find the average departure delay of a certain airport
7  - Find the average arrival delay time for a certain flight carrier
8  - Lists the months and the average taxi out time in a given airport
9  - Find the average delay due to weather when leaving from a certain airport
10 - Find the number of cancelled flights in a given month and year
11 - Find the distance in miles between two airports
12 - Find the average delay due to late aircraft given a specific carrier and year
13 - Find the month and delay with the longest delay given a year
14 - Finds the destination state that most flights go to given an origin state
15 - List the month and the carrier with the most amount of flights cancelled in a given year
()
Enter your choice here: 4
What state would you like to see? Oregon
Code | City, State
-------------------------
LMT  | Klamath Falls, OR
MFR  | Medford, OR
OTH  | North Bend/Coos Bay, OR
EUG  | Eugene, OR
RDM  | Bend/Redmond, OR
PDX  | Portland, OR
```

Function 5:

a. This function finds the probability that a flight is going to a certain airport will be cancelled.
b. For this we used aggregate and group functions.
c.

```
# Find the probability that a flight going to a certain airport will be cancelled
def probabilityOfCancellation(airport):
    results = collection.aggregate([{'$match': {"DEST": airport}},{'$group':{'_id': "$DEST", 'chance_of_cancellation': {'$avg': "$CANCELLED"}}}])
    for result in results:
        print("The probability that a flight going to %s will be cancelled is %.4f or %2.2f percent" % (airport, result['chance_of_cancellation'], result['chance_of_cancellation'] *
            100.0))
```

Output:

```
ubuntu@ip-172-31-17-181:~$ python Main.py
Welcome! What would you like to do today? Here are your choices:
1  - Find the number of flights coming from a specific state
2  - Lists the months and the average departure delay experienced in those months
3  - Find the average flight time from flights between certain airports
4  - Find all airport codes in a certain state
5  - Find the probability that a flight going to an airport will be cancelled
6  - Find the average departure delay of a certain airport
7  - Find the average arrival delay time for a certain flight carrier
8  - Lists the months and the average taxi out time in a given airport
9  - Find the average delay due to weather when leaving from a certain airport
10 - Find the number of cancelled flights in a given month and year
11 - Find the distance in miles between two airports
12 - Find the average delay due to late aircraft given a specific carrier and year
13 - Find the month and delay with the longest delay given a year
14 - Finds the destination state that most flights go to given an origin state
15 - List the month and the carrier with the most amount of flights cancelled in a given year
()
Enter your choice here: 5
What airport would you like to see? SFO
The probability that a flight going to SFO will be cancelled is 0.0246 or 2.46 percent
```

Function 6:

a. This function finds the average departure delay of a certain airport.
b. For this we used aggregate and group functions.

```
# Find the average departure delay of a certain airport
def averageDepartureDelayAtAirport(airport):
    results = collection.aggregate([{'$match': {"ORIGIN": airport}}, {'$group': {'_id': "$ORIGIN",'average_departure_delay': { '$avg': "$DEP_DELAY"}}}])
    for result in results:
        print("The average departure delay from %s is %.2f minutes" % (airport, result['average_departure_delay']))
```

Output:

```
ubuntu@ip-172-31-17-181:~$ python Main.py
Welcome! What would you like to do today? Here are your choices:
1  - Find the number of flights coming from a specific state
2  - Lists the months and the average departure delay experienced in those months
3  - Find the average flight time from flights between certain airports
4  - Find all airport codes in a certain state
5  - Find the probability that a flight going to an airport will be cancelled
6  - Find the average departure delay of a certain airport
7  - Find the average arrival delay time for a certain flight carrier
8  - Lists the months and the average taxi out time in a given airport
9  - Find the average delay due to weather when leaving from a certain airport
10 - Find the number of cancelled flights in a given month and year
11 - Find the distance in miles between two airports
12 - Find the average delay due to late aircraft given a specific carrier and year
13 - Find the month and delay with the longest delay given a year
14 - Finds the destination state that most flights go to given an origin state
15 - List the month and the carrier with the most amount of flights cancelled in a given year
()
Enter your choice here: 6
What airport would you like to see? LAX
The average departure delay from LAX is 9.95 minutes
```

Function 7:

a. This function finds the average delay time for a certain flight carrier.
b. For this we used aggregate and group functions which serve the query.
c.

```
# Find the average arrival delay time for a certain flight carrier
def averageDelayTimePerFlightCarrier(carrier):
    results = collection.aggregate([{'$match': {"CARRIER": carrier}},{'$group':{'_id': "$CARRIER",'average_arrival_delay': { '$avg': "$ARR_DELAY" }}}])
    for result in results:
        print("The average delay time of %s is %2.2f minutes" % (carrier, result['average_arrival_delay']))
```

Output:

```
ubuntu@ip-172-31-17-181:~$ python Main.py
Welcome! What would you like to do today? Here are your choices:
1  - Find the number of flights coming from a specific state
2  - Lists the months and the average departure delay experienced in those months
3  - Find the average flight time from flights between certain airports
4  - Find all airport codes in a certain state
5  - Find the probability that a flight going to an airport will be cancelled
6  - Find the average departure delay of a certain airport
7  - Find the average arrival delay time for a certain flight carrier
8  - Lists the months and the average taxi out time in a given airport
9  - Find the average delay due to weather when leaving from a certain airport
10 - Find the number of cancelled flights in a given month and year
11 - Find the distance in miles between two airports
12 - Find the average delay due to late aircraft given a specific carrier and year
13 - Find the month and delay with the longest delay given a year
14 - Finds the destination state that most flights go to given an origin state
15 - List the month and the carrier with the most amount of flights cancelled in a given year
()
Enter your choice here: 7
What carrier would you like to see? DL
The average delay time of DL is 0.57 minutes
```

Function 8 :

a. This function lists the months and the average taxi out time in a given airport.
b. For this we used aggregate and group functions which serve the query.
c.

```
# Lists the months and the average taxi out time in a given airport
def averageTaxiOutTimePerMonth(airport):
    results = collection.aggregate([{'$match': {"ORIGIN": airport}}, {'$group': {'_id': "$MONTH", 'avgTaxiOutTime': {'$avg': "$TAXI_OUT"}}}, {'$sort': {'_id': 1}}])
    print('Month |  Taxi Out Time in Minutes')
    print('--------------------------------')
    for result in results:
        if result['_id'] is not None:
            print("%-6d|%9.2f" % (result['_id'], result['avgTaxiOutTime']))
```

Output:

```
ubuntu@ip-172-31-17-181:~$ python Main.py
Welcome! What would you like to do today? Here are your choices:
1  - Find the number of flights coming from a specific state
2  - Lists the months and the average departure delay experienced in those months
3  - Find the average flight time from flights between certain airports
4  - Find all airport codes in a certain state
5  - Find the probability that a flight going to an airport will be cancelled
6  - Find the average departure delay of a certain airport
7  - Find the average arrival delay time for a certain flight carrier
8  - Lists the months and the average taxi out time in a given airport
9  - Find the average delay due to weather when leaving from a certain airport
10 - Find the number of cancelled flights in a given month and year
11 - Find the distance in miles between two airports
12 - Find the average delay due to late aircraft given a specific carrier and year
13 - Find the month and delay with the longest delay given a year
14 - Finds the destination state that most flights go to given an origin state
15 - List the month and the carrier with the most amount of flights cancelled in a given year
()
Enter your choice here: 8
What airport would you like to see? SFO
Month |  Taxi Out Time in Minutes
-----------------------------------
1     |    15.41
2     |    16.83
3     |    16.53
4     |    16.97
5     |    18.83
6     |    19.79
7     |    20.70
8     |    18.97
9     |    17.95
10    |    20.11
12    |    15.76
```

Function 9:

a. This function finds the average delay due to weather when leaving from a certain airport.
b. For this we used aggregate and group functions which serve the query.
c.

```
# Find the average delay due to weather when leaving from a certain airpor
def averageWeatherDelayWhenLeaving(airport):
    results = collection.aggregate([{'$match': {"ORIGIN": airport}}, {'$group': {'_id': "$MONTH", 'average_weather_delay': {'$avg': "$WEATHER_DELAY"}}}, {'$sort': {'_id': 1}}])
    print('Month |  Weather Delay in Minutes')
    print('--------------------------------')
    for result in results:
        if result['_id'] is not None:
            print("%-6d|%9.2f" % (result['_id'], result['average_weather_delay']))
```

Output:

```
ubuntu@ip-172-31-17-181:~$ python Main.py
Welcome! What would you like to do today? Here are your choices:
1  - Find the number of flights coming from a specific state
2  - Lists the months and the average departure delay experienced in those months
3  - Find the average flight time from flights between certain airports
4  - Find all airport codes in a certain state
5  - Find the probability that a flight going to an airport will be cancelled
6  - Find the average departure delay of a certain airport
7  - Find the average arrival delay time for a certain flight carrier
8  - Lists the months and the average taxi out time in a given airport
9  - Find the average delay due to weather when leaving from a certain airport
10 - Find the number of cancelled flights in a given month and year
11 - Find the distance in miles between two airports
12 - Find the average delay due to late aircraft given a specific carrier and year
13 - Find the month and delay with the longest delay given a year
14 - Finds the destination state that most flights go to given an origin state
15 - List the month and the carrier with the most amount of flights cancelled in a given year
()
Enter your choice here: 9
What airport would you like to see? LAX
Month |  Weather Delay in Minutes
-----------------------------------
1     |     1.66
2     |     1.13
3     |     0.77
4     |     0.57
5     |     0.96
6     |     0.55
7     |     0.56
8     |     1.28
9     |     1.10
10    |     0.61
12    |     1.10
```

Function 10:

a. This function lists the months and number of cancelled flights in a given year.
b. For this we used find function.
c.

```
# Lists the months and number of cancelled flights in a given year
def findNumberOfCancelledFlights(month, year):
    print('The number of flights cancelled in {}-{} is {} from a total of {} flights'.format(month, year, collection.find({'MONTH': int(month), 'YEAR': int(year), 'CANCELLED': 1}).
        count(), collection.find({'MONTH': int(month), 'YEAR': int(year)}).count()))
```

Output:

```
ubuntu@ip-172-31-17-181:~$ python Main.py
Welcome! What would you like to do today? Here are your choices:
1  - Find the number of flights coming from a specific state
2  - Lists the months and the average departure delay experienced in those months
3  - Find the average flight time from flights between certain airports
4  - Find all airport codes in a certain state
5  - Find the probability that a flight going to an airport will be cancelled
6  - Find the average departure delay of a certain airport
7  - Find the average arrival delay time for a certain flight carrier
8  - Lists the months and the average taxi out time in a given airport
9  - Find the average delay due to weather when leaving from a certain airport
10 - Find the number of cancelled flights in a given month and year
11 - Find the distance in miles between two airports
12 - Find the average delay due to late aircraft given a specific carrier and year
13 - Find the month and delay with the longest delay given a year
14 - Finds the destination state that most flights go to given an origin state
15 - List the month and the carrier with the most amount of flights cancelled in a given year
()
Enter your choice here: 10
Which month? 6
Which year? 2014
The number of flights cancelled in 6-2014 is 1741 from a total of 106156 flights
```

Function 11:

a. This function finds the distance between two airports.
b. For this we used aggregate and group function.
c.

```
# Find the distance in miles between two airports
def findDistanceBetweenAirports(airport1, airport2):
    results = collection.aggregate([{'$match': {"ORIGIN": airport1, "DEST": airport2}}, {'$group': {'_id': "$ORIGIN",'distance': { '$avg': "$DISTANCE"}}}])
    for result in results:
        print("The distance between %s and %s is %.0f miles" % (airport1, airport2, result['distance']))
```

Output:

```
ubuntu@ip-172-31-17-181:~$ python Main.py
Welcome! What would you like to do today? Here are your choices:
1  - Find the number of flights coming from a specific state
2  - Lists the months and the average departure delay experienced in those months
3  - Find the average flight time from flights between certain airports
4  - Find all airport codes in a certain state
5  - Find the probability that a flight going to an airport will be cancelled
6  - Find the average departure delay of a certain airport
7  - Find the average arrival delay time for a certain flight carrier
8  - Lists the months and the average taxi out time in a given airport
9  - Find the average delay due to weather when leaving from a certain airport
10 - Find the number of cancelled flights in a given month and year
11 - Find the distance in miles between two airports
12 - Find the average delay due to late aircraft given a specific carrier and year
13 - Find the month and delay with the longest delay given a year
14 - Finds the destination state that most flights go to given an origin state
15 - List the month and the carrier with the most amount of flights cancelled in a given year
()
Enter your choice here: 11
From airport: SFO
to airport: LAX
The distance between SFO and LAX is 337 miles
```

Function 12:

a. This function finds the average delay due to late aircraft given a specific carrier and year.
b. We used aggregate and group function which serves the query.
c.

```
# Find the average delay due to late aircraft given a specific carrier and year
def averageLateAircraftDelayFromCarrier(carrier, year):
    results = collection.aggregate([{'$match': {"CARRIER": carrier, "YEAR": int(year)}}, {'$group': {'_id': "$CARRIER",'late_delay': { '$avg': "$NAS_DELAY"}}}])
    for result in results:
        print("The average delay due to late aircraft from %s in %s is %.2f minutes" % (carrier, year, result['late_delay']))
```

Output:

```
ubuntu@ip-172-31-17-181:~$ python Main.py
Welcome! What would you like to do today? Here are your choices:
1  - Find the number of flights coming from a specific state
2  - Lists the months and the average departure delay experienced in those months
3  - Find the average flight time from flights between certain airports
4  - Find all airport codes in a certain state
5  - Find the probability that a flight going to an airport will be cancelled
6  - Find the average departure delay of a certain airport
7  - Find the average arrival delay time for a certain flight carrier
8  - Lists the months and the average taxi out time in a given airport
9  - Find the average delay due to weather when leaving from a certain airport
10 - Find the number of cancelled flights in a given month and year
11 - Find the distance in miles between two airports
12 - Find the average delay due to late aircraft given a specific carrier and year
13 - Find the month and delay with the longest delay given a year
14 - Finds the destination state that most flights go to given an origin state
15 - List the month and the carrier with the most amount of flights cancelled in a given year
()
Enter your choice here: 12
What carrier would you like to see? DL
Which year? 2014
The average delay due to late aircraft from DL in 2014 is 13.89 minutes
```

Function 13:

a. This function finds the month and delay in a given year.
b. We used aggregate and group function which serves the query.
c.

```
# Find the month and delay with the longest delay given a year
def findTheMonthWithTheLongestDelayGivenAYear(year):
    results = collection.aggregate([{'$match': {"YEAR": int(year)}}, {'$group':{'_id': "$MONTH",'delay': { '$avg': "$ARR_DELAY" }}}, {'$sort': {'delay': -1}}, {'$limit': 1}])
    for result in results:
        print("The month with the longest delays in %s was month %s at %.2f minutes of delay" % (year, result['_id'], result['delay']))
```

Output:

```
ubuntu@ip-172-31-17-181:~$ python Main.py
Welcome! What would you like to do today? Here are your choices:
1  - Find the number of flights coming from a specific state
2  - Lists the months and the average departure delay experienced in those months
3  - Find the average flight time from flights between certain airports
4  - Find all airport codes in a certain state
5  - Find the probability that a flight going to an airport will be cancelled
6  - Find the average departure delay of a certain airport
7  - Find the average arrival delay time for a certain flight carrier
8  - Lists the months and the average taxi out time in a given airport
9  - Find the average delay due to weather when leaving from a certain airport
10 - Find the number of cancelled flights in a given month and year
11 - Find the distance in miles between two airports
12 - Find the average delay due to late aircraft given a specific carrier and year
13 - Find the month and delay with the longest delay given a year
14 - Finds the destination state that most flights go to given an origin state
15 - List the month and the carrier with the most amount of flights cancelled in a given year
()
Enter your choice here: 13
Which year? 2013
The month with the longest delays in 2013 was month 12 at 7.14 minutes of delay
```

Function 14:

a. This function finds the destination state that most flights go from origin state.
b. We used aggregate and group function which serves the query.
c.

```
# Finds the destination state that most flights go to given an origin state
def findMostTraveledStateFromGivenState(state):
    results = collection.aggregate([{'$match': {"ORIGIN_STATE_NM": state}}, {'$group':{'_id': {'origin': "$ORIGIN_STATE_NM", 'dest': "$DEST_STATE_NM"}, 'count': { '$sum': 1}}}, {'$
        sort': {'count': -1}}, {'$limit': 1}])
    for result in results:
        print("The state with the most flights from %s is %s with a number of %d flights" % (state, result['_id']['dest'], result['count']))
```

Output:

```
ubuntu@ip-172-31-17-181:~$ python Main.py
Welcome! What would you like to do today? Here are your choices:
1  - Find the number of flights coming from a specific state
2  - Lists the months and the average departure delay experienced in those months
3  - Find the average flight time from flights between certain airports
4  - Find all airport codes in a certain state
5  - Find the probability that a flight going to an airport will be cancelled
6  - Find the average departure delay of a certain airport
7  - Find the average arrival delay time for a certain flight carrier
8  - Lists the months and the average taxi out time in a given airport
9  - Find the average delay due to weather when leaving from a certain airport
10 - Find the number of cancelled flights in a given month and year
11 - Find the distance in miles between two airports
12 - Find the average delay due to late aircraft given a specific carrier and year
13 - Find the month and delay with the longest delay given a year
14 - Finds the destination state that most flights go to given an origin state
15 - List the month and the carrier with the most amount of flights cancelled in a given year
()
Enter your choice here: 14
What state would you like to see? Florida
The state with the most flights from Florida is California with a number of 9964 flights
```

Function 15:

a. This function lists the month and the carrier with the most amount of cancelled flights in a given year.
b. We used aggregate and group function which serves the query.
c.

```
# List the month and the carrier with the most amount of flights cancelled in a given year
def listMonthAndCarrierOfMostCancelledFlights(year):
    results = collection.aggregate([{'$match': {"CANCELLED": 1, "YEAR": int(year)}}, {'$group':{'_id': {'month': "$MONTH", 'carrier': "$CARRIER"}, 'count': { '$sum': 1}}}, {'$sort':
        {'count': -1}}, {'$group': {'_id':'$_id.month', 'carrier':{'$first': '$_id.carrier'},'count':{'$first': '$count'}}}, {'$sort': {'_id': 1}}])
    print('Month |  Carrier  | Number of Cancellations')
    print('-----------------------------------------')
    for result in results:
        if result['_id'] is not None:
            print("%-6d|%6s   |%13d" % (result['_id'], result['carrier'], result['count'] ))
```

Output:

```
ubuntu@ip-172-31-17-181:~$ python Main.py
Welcome! What would you like to do today? Here are your choices:
1  - Find the number of flights coming from a specific state
2  - Lists the months and the average departure delay experienced in those months
3  - Find the average flight time from flights between certain airports
4  - Find all airport codes in a certain state
5  - Find the probability that a flight going to an airport will be cancelled
6  - Find the average departure delay of a certain airport
7  - Find the average arrival delay time for a certain flight carrier
8  - Lists the months and the average taxi out time in a given airport
9  - Find the average delay due to weather when leaving from a certain airport
10 - Find the number of cancelled flights in a given month and year
11 - Find the distance in miles between two airports
12 - Find the average delay due to late aircraft given a specific carrier and year
13 - Find the month and delay with the longest delay given a year
14 - Finds the destination state that most flights go to given an origin state
15 - List the month and the carrier with the most amount of flights cancelled in a given year
()
Enter your choice here: 15
What year would you like to see? 2014
Month |  Carrier  | Number of Cancellations
-----------------------------------------
1     |    WN   |          643
2     |    OO   |         1193
3     |    OO   |          509
4     |    OO   |          414
5     |    OO   |          391
6     |    OO   |          925
7     |    OO   |          688
8     |    OO   |          476
9     |    WN   |          342
10    |    OO   |          321
```