MongoDB Advanced Administrator Training

Advanced Administrator Instructor Material

Introduction

Advanced Administrator Training is a 2-day consulting engagement at the customer site. It consists of splitting the students into teams and running a set of scenarios on MongoDB that pits each team against each other in a bid to identify and provide a solution to the problem put in front of them. Teams are awarded points on speed, elegance, and effectiveness of their solution. The winning team receives a prize.

Advanced Administrator Training is an entirely practical set of exercises, where teams are hands-on throughout the day, actively working together to identify and implement a solution.

Audience

This session is designed for 4-12 people. Audience should be technical. The day is mostly beneficial for operations/DBAs, but developers may also find content interesting.

The Advanced Administrator Training session should ideally follow on from a MongoDB training course, as candidates will be using prior knowledge to complete exercises. This is not designed for complete beginners.

Prerequisites

You will need to make sure that there are adequate resources in place in order to deliver a successful session. You will require:

Requirement	**Importance**
An instructor machine (your laptop is OK)	CRITICAL
A training room with a projector	CRITICAL
1 machine per team (teams are made up of 3 or 4 people)	CRITICAL
Machines should have outside Internet access (both HTTP and SSH required), as they will be shelling out to EC2 instances.	CRITICAL
A flipchart/whiteboard	non-critical
Credentials • the AWS account `training-aws`, for the instructor • the MMS account `advanced-ops+instructor@mongodb.com`, for the instructor • the MMS account `advanced-ops+student@mongodb.com`, for the students	CRITICAL

If any of these requirements fail, please consult training@mongodb.com and ce@mongodb.com for further information about whether you can deliver the session.

Prizes

As an incentive for teams to get competitive, there should ideally be a prize for the winning team. Please make sure you know how many people are going to be attending the session and ensure you know how many people will be in each team. Each member of the winning team should receive a prize. Prizes should be small – some ideas of prizes include T-shirts, O'Reilly books, boxes of chocolates, bottles of wine, small trophies.

Team Setup

Firstly make sure you explain the plan for the next 2 days:

- Competitors will be divided into teams
- Teams will compete against each other in a series of tasks
- Tasks will revolve around real-life MongoDB scenarios
- Teams will need to identify the problem, propose and execute a solution
- Teams will be awarded points for speed, elegance and accuracy of the solution
- The team with the most points at the end of the second day will be declared winner of the training and will receive a prize

Make sure that the most experienced MongoDB competitors in the room (there are usually 1 or 2 more experienced than the others) are on different teams. It also makes sense to mix in Developers with Operations so there are balanced knowledge pools in each team. Give teams 2 minutes to relocate around a single "team" machine in the classroom, and to come up with a team name.

After people have moved around to their team machines, assign each team a group number and write them up on the whiteboard, scores will be written up against them here.

Each team will have their own assigned EC2 instances. And for the most part, should work entirely out of the `/home/ec2-user/scripts` directory and `mongo` shell on their dedicated virtual machines.

MMS Credentials

Some exercises require you and the students to use MMS. You should log in to MMS as the user `advanced-ops+instructor@mo` and students should log in as `advanced-ops+student@mongodb.com`.

If you're missing the password to either of these accounts, contact `training@mongodb.com` to get access.

Installing and configuring the `aws-cli` tool

Install `aws-cli`

Check whether you have `aws-cli` installed by running:

```
$ aws --version
```

If it isn't installed, you can install it using pip:

```
$ sudo pip install awscli
```

If you don't have pip, you can download it here: https://pip.pypa.io/en/latest/installing.html

Get access to the `training-aws` account

The AWS admin for training needs to give you a username and password to use for accessing the the `training-aws` account. Contact `training@mongodb.com` if you haven't received these.

Sign in to the AWS console at https://mongodb-training-aws.signin.aws.amazon.com/console. You will be required to change your password the first time you log in.

Generate an AWS API key

From the AWS console home (the view that lists all AWS services), under "Administration and Security", click "Identity & Access Management", then click "Users" on the left sidebar. In the list of usernames, click your username, then scroll down and click "Manage Access Keys", then click "Create Access Key". Click "Show User Security Credentials" and leave this window open so you can copy and paste the keys in the next step.

Configure `aws-cli` to use your API key

Run

```
$ aws configure --profile wargaming
```

and when prompted, fill in the following values:

```
AWS Access Key ID [None]: [paste the Access Key ID here]
AWS Secret Access Key [None]: [paste the Secret Access Key here]
Default region name [None]: us-east-1
Default output format [None]: text
```

There should now be a file named `~/.aws/config` containing the following:

```
[profile wargaming]
aws_access_key_id = [your Access Key ID]
aws_secret_access_key = [your Secret Access Key]
region = us-east-1
output = text
```

You can close the window from the previous step, and ignore the warning ("You haven't downloaded the User security credentials. This is the last time these credentials will be available for download."). If you ever lose your AWS API key you can generate a new one.

Verify that you can authenticate

Run

```
$ aws ec2 describe-regions --profile wargaming
```

Verify that you can spin up instances

Open the EC2 console in a browser, then open a shell and cd to the `admin-administrators-training/instance-launch` directory and run

```
$ ./repl.sh 1 [Your Name]
$ ./shutDownRepl.sh
```

You should see one instance called "TrainingReplTeam-1-Your-Name" be created then terminated.

Launching Instances

1. To launch the initial instances for the groups for all sections other than *Performance* go into the Advanced Administrator Training repo located at `training/advanced-administrators-training/` and go into the *instance-launch* directory. To run any of the scripts in this directory you must have the Amazon `aws-cli` tool installed and configured on your machine. If you don't have the Amazon `aws-cli` tool installed on your machine, see the following section for installation instructions.

   To launch a node for each team with the preloaded scripts and data sets run the following command `advanced-administrators-training/instance-launch/repl.sh [Number of teams] [Your Name]` We use your name so that the instances can be associated with you.

   The script should return to you an instance ID. Give it a minute or two for all of the instance to launch and become available. To give the teams the IP addresses to connect to, run the `advanced-administrators-training/instance-launch/getReplIPs.sh` script and the output should give you what you need to get started.

2. To launch the sharded cluster for the performance and sharding exercise, you'll want to run `advanced-administrators-training/instance-launch/shard.sh [Number of teams] [Your Name]`. It should be noted that by default *this script will launch 8 machines per team* - 2 shards with 3 member replica sets each, a mongos, and a config server. This can add up if you have a lot of teams - be mindful of EC2 account limits!

   The script should take about 10 minutes (in total, not per-team). You might want to leave the AWS console open so you can keep an eye on its progress. When it's done it will print the public DNS names of the instances, along with which team each instance belongs to. You can also run `advanced-administrators-training/instance-launch/getShardIPs.sh` to print them out again.

Note: if you ran this training before 2015, you may have needed to run this script hours in advance because it took so long. This should not be necessary any more; it usually takes about 10 minutes.

Scoring

Teams are awarded up to 5 points per exercise based on the following criteria:

- Speed of solution
- Elegance of solution
- Effectiveness of solution

Even if teams do not fully complete the exercise in practice, they can still be awarded points for coming up with a theoretical solution.

Use the following table as a rough guide for awarding points:

Outcome	**Points**
Exercise completed within the time frame, and correct result achieved with most effective method.	5
Exercise completed within the time frame, correct result achieved, but most effective method not used.	4
Exercise not completed on time, but there is a sound theoretical strategy and if there had been more time they would have completed the task.	3
Exercise not completed, no real strategy but some good technical ideas tried.	2
Exercise not completed, no strategy and/or went down the wrong path.	1

N.B. From experience, most teams will not complete the exercises in the time frame. Use your best judgment as to which team is the stronger one on each exercise and award points accordingly.

Exercises

There are a range of different exercise topics to get through during the day. The idea is to cover all key areas of MongoDB – Backup and Restore, Sharding, Replica Set Maintenance, Schema/Aggregation and Security practices.

It is important to get an understanding of the architecture of the customer before you arrive. It may make no sense to spend time working on a sharding exercise if the customer never intends to shard (although it might possibly be useful in future in case they do). Present each team with the scenario, which can also be read by them in their attendee PDF.

Some exercises will take longer than others. For example, the backup and sharded performance exercises are probably half a day each, whereas the rolling replica set upgrades are only 30 minutes or so. Depending how experienced the teams are with the aggregation framework, the time frame can range up to a couple of hours to deliver an aggregation statement. After the time for each exercise has expired, go around each team and ask them to provide the step by step for their solution, and find out whether they completed the task or not. Award points based on the criteria above. Then spend a short amount of time going through the correct solution, detailed in this guide.

After each exercise, allow a 5 minute break to set up the following task.

Virtual Machines

Outside Internet access for the competitors is vital so they can access the virtual machines running in Amazon EC2.

The VMs are `M1.small` instances running Amazon Linux. SSH'ing to the instances will require the *PEM file* contained in the Advanced Administrator Training Git repository. You will need to ensure that the PEM file has been distributed to the competitors in order for them to connect to the instances. You will also need the IP addresses listed from the `/<path-to-wargaming-repo>/ instance-launch/getReplIPs.sh` script. Give each time their virtual machine IP addresses.

From the terminal on a Linux/Mac machine, connect with the following command:

```
ssh ec2-xx-xxx-xx-xx.compute-1.amazonaws.com \
    -l ec2-user -i /path/to/AdvancedOpsTraining.pem
```

When connecting from a Windows machine, ensure there is a suitable SSH client installed that can handle file authentication, such as PuTTY and PuTTYgen. See the following article for information on using PEM files with PuTTY:

http://support.cdh.ucla.edu/help/132-file-transfer-protocol-ftp/583-converting-your-private-key-

Users should log in as `ec2-user`, which has `sudo` access. Each team will have their own instance to connect to in order to complete each exercise.

Backups and Recovery

Premise

"*I thought I was on my development machine and I've accidentally dropped a collection in production.*"

Your data is backed up in MMS, so you can recover all the data that existed immediately before the drop. You'll need to request a point-in-time backup and then restore it.

The collection is `test.users` and the total number of documents prior to the drop was 20,000.

Setup

To set up this scenario, ssh into each instance and run the following script:

```
~/scripts/Backups/premise-1.sh
```

This script will confirm that the automation agent is running and start a replica set on ports 27017, 27018, 27019. Then it will prompt you to import the replica set into MMS.

1. Log in to MMS as `advanced-ops+instructor@mongodb.com`
2. Choose the MMS Group `team01-advanced-ops` from the dropdown in the top-left
3. Install the Monitoring and Backup agents
   - see https://docs.mms.mongodb.com/tutorial/move-agent-to-new-server/#install-additional-agent-as-hot-standby-for-high-availability
4. Import Team 1's replica set for monitoring
   - see https://docs.mms.mongodb.com/tutorial/add-hosts-to-monitoring/#add-mongodb-processes
5. Activate Backup for Team 1's replica set
   - see https://docs.mms.mongodb.com/tutorial/enable-backup-for-replica-set/#procedure
6. Repeat steps 2-5 for each team
7. Re-run the script for on each EC2 instance
   - This will confirm that Backup is enabled on the replica set and finish setting up the scenario.

Cleanup

Once students are done with this exercise, terminate Backup for all the replica sets and remove the replica sets from MMS.

Solution

Obtain the backup

1. Connect to the server using the `mongo` shell and find the bad operation in the oplog:

   ```
   mongo
   use local
   var badOp = db.oplog.rs.find({ op: "c" }).sort({ ts: -1 }).next();
   ```

2. Find the timestamp of the most recent good operation:

   ```
   db.oplog.rs.find({ ts: { $lt: badOp.ts } }).sort({ ts: -1 }).next()
   ```

3. In the MMS console, go to "Backup", your group's replica set, "Restore", "Use Exact Oplog Timestamp", and enter the two numeric values from the `Timestamp` value you found in the oplog.

4. Choose "Pull via Secure HTTP" and `curl` the download URL from the ec2 instance. Keep this tar file; you'll need to unpack it several times while restoring.

Restore the backup

This is a many-step process; see

https://docs.mms.mongodb.com/tutorial/restore-replica-set/#restore-the-primary

Replication

Premise 1

"*MongoDB releases a new version. How do we upgrade our production systems with minimal downtime?*"

Setup

- Use the same VM from the backup/restore exercise. This is preloaded with versions 2.4 and 2.6 of MongoDB.
- Make sure you have done the "Cleanup" step of the previous exercise, to remove everything from MMS.
- Run `~/scripts/Replication/premise-1-2.sh`
- This script will create a 3 node replica set running the previous MongoDB version release with data in the `test` database in the `users` directory. Provide one replica set per team.

Solution 1

1. Take a secondary offline; restart it with the new binary.
2. Take the other secondary (if there is one) offline, and restart it with the new binary as well.
3. Perform an `rs.stepDown()` on the primary. This will force the primary to step down gracefully and attempt to avoid election as primary for 60 seconds (may need to increase this time by specifying it as a parameter to rs.stepDown() if this node is configured with a higher priority, so that it doesn't try to become primary again until it is restarted with the new binary). Restart it with the new binary.

Now your replica set should be fully upgraded with minimal downtime from an application perspective (i.e. just a replica set failover).

Premise 2

"*We have a new set of queries we want to run against the production system. But whenever we try to build the indexes for these queries, we experience a massive slowdown. How can we build indexes on the production system while minimizing the performance penalty?*"

You will build indexes on the `test.users` collection for the fields `emailAddress` and `userId`.

Setup

- Use the same VM from the backup/restore exercise. This is preloaded with versions 2.4 and 2.6 of MongoDB.
- Run `~/scripts/Replication/premise-1-2.sh`
- This script will create a 3 node replica set running the previous MongoDB version release with data in the `test` database in the `users` directory. Provide one replica set per team.

Solution 2

A rolling index build. Similar to the upgrade process, the solution is as follows:

1. **For each secondary in the set, build an index according to the following steps:**
    1. Restart the `mongod` without the `--replSet` option and on a different port.
    2. Create the index using `ensureIndex()`.
    3. Restart the `mongod` as a member of the replica set on its usual port
2. Perform an `rs.stepDown()` on the current primary and make sure to wait for one of the other nodes to be elected primary.
3. Build the index on the former primary using the same procedure as above.

Ensure that the oplog window is large enough to permit the indexing to complete without the node falling too far behind to catch up when it is re-introduced into the replica set.

Premise 1 with MMS Automation

Try Premise 1 again, but using MMS Automation. First the instructor will reset your replica set to its original state. Then you can import the replica set into MMS and use Automation to perform a minimal-downtime version upgrade.

Setup

- Use the same VM from the previous exercises
- Re-run `~/scripts/Replication/premise-1-2.sh`

Solution 1 with MMS Automation

First import the cluster into MMS.

1. Add > Import Existing for Monitoring
    - for "Host Type" choose "Replica Set"
    - for "Internal Hostname" paste the output of `hostname` on the VM
    - for "Port" specify 27017

   MMS should discover the whole replica set once you add the primary.
2. Add > Import Existing for Automation
    - choose the replica set from the "Deployment Item" dropdown

Then use Automation to perform a rolling upgrade.

1. Click the wrench next to the replica set to edit the replica set
2. Change the "version" dropdown to the latest 2.6.x
3. Click "Review and deploy"

Premise 3

"*Your primary has become unavailable because of a power surge. Fortunately, automatic failover kicked in and the system never went down. Unfortunately, you never set alerts to inform you when a node becomes unavailable, so weeks go by until you notice. When you bring it back up, the node does not restart cleanly. Find out why and fix it.*"

Setup

- Use the same VM from the backup/restore exercise. This is preloaded with versions 2.4 and 2.6 of MongoDB.
- Run `~/scripts/Replication/premise-3.sh` which will bring up a replica set, with a member with a small oplog, and insert enough data until it falls off the oplog.

Note that the RS102 error message in the system log indicates the "fall off" issue.

Solution 3

1. Take the node down
2. Delete the data files
3. Restart the node
4. Wait for full resync

Points for elegance if they resize the oplog to something bigger or change it back to the default.

Aggregation Framework + Map-Reduce

Objective

The objective in these exercises is to put in practice as many of the aggregation framework operators as possible. There are extension exercises for savvy developers to be challenged further and as time permits.

Premise

Your cluster is experiencing some performance issues and you would like to determine where the bottlenecks are. You will need to create statistics on slow queries, locking, and operations: use the database profiler and write some aggregation queries to analyze the profiling data.

Prerequisites

- Run the `~/scripts/Aggregation/premise-1-2.sh` script which will set up an empty 3 node replica set.

Setup

Each student should perform the following operations:

1. To prepare the system, first enable the profiler for a new agg database (to record all queries):

```
> use agg;
> db.setProfilingLevel(2);
```

2. Add some sample data:

```
> for (i=0; i<100000; i++) { db.wargame.insert( { count : i } ); }
```

3. Add some queries:

```
> for (i=0; i<100; i++) { db.wargame.find( { count : i } ).toArray(); }
> for (i=0; i<100; i++) { db.wargame.update( { count : i },
                                              { $set : { "another_field" : i } } ); }
```

Exercise 1

Find the maximum response time and average response time for each type of operation in the `system.profile` collection. Hint: group on the "`op`" field.

Your result should have the following form:

```
{
        "result" : [
                {
                        "_id" : "update",
                        "count" : <NUMBER>,
                        "max response time" : <NUMBER>,
                        "avg response time" : <NUMBER>
                },
                {
                        "_id" : "query",
                        "count" : <NUMBER>,
                        "max response time" : <NUMBER>,
                        "avg response time" : <NUMBER>
                },
                {
                        "_id" : "insert",
                        "count" : <NUMBER>,
                        "max response time" : <NUMBER>,
                        "avg response time" : <NUMBER>
                }

                ... for every operation in the system.profile.op field
        ],
        "ok" : 1
}
```

Solution 1

```
// response time by operation type
db.system.profile.aggregate( { $group : {
_id :"$op",
count:{$sum:1},
"max response time":{$max:"$millis"}, "avg response time":{$avg:"$millis"}
}});
```

Result


```
{
  "result" : [
    {
      "_id" : "update",
      "count" : 100,
      "max response time" : 0,
      "avg response time" : 0
    },
    {
      "_id" : "query",
      "count" : 100,
      "max response time" : 32,
      "avg response time" : 26.29
    },
    {
      "_id" : "insert",
      "count" : 3903,
      "max response time" : 3,
      "avg response time" : 0.0007686395080707148
    }
  ],
  "ok" : 1
}
```

Exercise 2

Render detailed statistics using the data from the `system.profile` collection.

Your result should have the following form:

```
{
        "result" : [
                {
                        "_id" : "command",
                        "average response time" : <NUMBER>,
                        "average response time + acquire time" : <NUMBER>,
                        "average acquire time reads" : <NUMBER>,
                        "average acquire time writes" : <NUMBER>,
                        "average lock time reads" : <NUMBER>,
                        "average lock time writes" : <NUMBER>
                },
                {
                        "_id" : "update",
                        "average response time" : <NUMBER>,
                        "average response time + acquire time" : <NUMBER>,
                        "average acquire time reads" : <NUMBER>,
                        "average acquire time writes" : <NUMBER>,
                        "average lock time reads" : <NUMBER>,
                        "average lock time writes" : <NUMBER>
                },
                {
                        "_id" : "query",
                        "average response time" : <NUMBER>,
                        "average response time + acquire time" : <NUMBER>,
                        "average acquire time reads" : <NUMBER>,
                        "average acquire time writes" : <NUMBER>,
                        "average lock time reads" : <NUMBER>,
                        "average lock time writes" : <NUMBER>
                },
                {
                        "_id" : "insert",
                        "average response time" : <NUMBER>,
                        "average response time + acquire time" : <NUMBER>,
                        "average acquire time reads" : <NUMBER>,
                        "average acquire time writes" : <NUMBER>,
                        "average lock time reads" : <NUMBER>,
                        "average lock time writes" : <NUMBER>
                }

                ... for every operation in the system.profile.op field

        ],
        "ok" : 1
}
```

Solution 2

```
// response time analysis
db.system.profile.aggregate(
  [
    { $project :
      {
        "op" : "$op",
        "millis" : "$millis",
        "timeAcquiringMicrosrMS" : { $divide : [ "$lockStats.timeAcquiringMicros.r", 1000 ] },
        "timeAcquiringMicroswMS" : { $divide : [ "$lockStats.timeAcquiringMicros.w", 1000 ] },
        "timeLockedMicrosrMS" : { $divide : [ "$lockStats.timeLockedMicros.r", 1000 ] },
        "timeLockedMicroswMS" : { $divide : [ "$lockStats.timeLockedMicros.w", 1000 ] }
      }
    },
    { $project :
      {
        "op" : "$op",
        "millis" : "$millis",
        "total_time" : { $add : [ "$millis", "$timeAcquiringMicrosrMS", "$timeAcquiringMicroswMS" ]
        "timeAcquiringMicrosrMS" : "$timeAcquiringMicrosrMS",
        "timeAcquiringMicroswMS" : "$timeAcquiringMicroswMS",
        "timeLockedMicrosrMS" : "$timeLockedMicrosrMS",
        "timeLockedMicroswMS" : "$timeLockedMicroswMS"
      }
    },
    { $group :
      {
        _id : "$op",
        "average response time" : { $avg : "$millis" },
        "average response time + acquire time": { $avg: "$total_time"},
        "average acquire time reads" : { $avg : "$timeAcquiringMicrosrMS" },
        "average acquire time writes" : { $avg : "$timeAcquiringMicroswMS" },
        "average lock time reads" : { $avg : "$timeLockedMicrosrMS" },
        "average lock time writes" : { $avg : "$timeLockedMicroswMS" }
      }
    }
  ]
);
```

Result


```
{
"result" : [
  {
    "_id" : "command",
    "average response time" : 20,
    "average response time + acquire time" : 20.006,
    "average acquire time reads" : 0.004,
    "average acquire time writes" : 0.002,
    "average lock time reads" : 20.813,
    "average lock time writes" : 0
  },
  {
    "_id" : "update",
    "average response time" : 0,
    "average response time + acquire time" : 0.0023000000000000017,
    "average acquire time reads" : 0,
    "average acquire time writes" : 0.0023000000000000017,
    "average lock time reads" : 0,
    "average lock time writes" : 0.07200000000000001
  },
  {
    "_id" : "query",
    "average response time" : 26.29,
    "average response time + acquire time" : 27.072920000000003,
    "average acquire time reads" : 0.6860699999999995,
    "average acquire time writes" : 0.09685000000000009,
    "average lock time reads" : 27.512089999999997,
    "average lock time writes" : 0
  },
  {
    "_id" : "insert",
    "average response time" : 0.0007690335811330429,
    "average response time + acquire time" : 0.006153806716226541,
    "average acquire time reads" : 0,
    "average acquire time writes" : 0.005384773135093499,
    "average lock time reads" : 0,
    "average lock time writes" : 0.009285824147654708
  }
],
"ok" : 1
}
```


Sharding & Performance Troubleshooting

Premise 1

"*Whoops! The new intern was in charge of writing a reporting job that would run nightly on your systems during off-peak hours (it takes hours to complete!) and he thought it would be a good idea to test if it works*

in production in the middle of your business day! Now your CPU is spiking and your prod system is sluggish. Deal with it."

Solution 1

Find the op using `db.CurrentOp`. And then end it with `db.killOp()`.

Instances

For the next exercise you will use the clusters launched by *shard.sh*

Each cluster is made up of 2 shards with 3-nodes replica set per shard, 1 config server node and 1 mongos node. Each element of the cluster resides on its own EC2 instance so 8 machines in total.

The cluster is configured badly:

- Data is missing indexes
- They each have low ulimits
- They have a high readahead set
- They are mounted with atime
- EXT3 File System
- EBS volume

Premise 2

"*You have noticed drastic performance issues with your MongoDB deployment. Queries are slow, responsiveness is generally slow, and updates take quite some time as well.*

You have tried a number of things to boost performance. You started reading off secondaries, reworked some of your queries, and even expanded to a sharded cluster.

So far nothing has worked and performance has stayed exactly the same. This is it, your last ditch effort to fix EVERYTHING that is wrong with this deployment. And if you can't fix it, or it will take a long time, identify what the problem is and come up with a plan of action to fix it."

The long-running queries include:

- finding users by screen name:

```
db.live.find( { "user.screen_name" : "____thaaly" } )
```

- finding all users who have a particular name e.g. Beatriz:

```
db.live.find( { "user.name" : /Beatriz/ } )
```

- finding all users in the Brasilia timezone with more than 70 friends, and sorting by most friends:

```
db.live.find( { "user.time_zone" : "Brasilia",
                "user.friends_count" : { $gt : 70 } } )
          .sort( { "user.friends_count" : -1 } )
```

Please investigate all possible causes of slow queries and slow system responsiveness.

Solution 2

Turn down readahead, mount with noatime, raise ulimits, make sure all queries are using appropriate indexes, change the filesystem, and come up with a better alternative for a shard key given the sample queries and the data set.

Query solutions:

- finding users by screen name - missing an index:

```
Q. db.live.find( { "user.screen_name" : "____thaaly" } )
A. db.live.ensureIndex( { "user.screen_name" : 1 } )
```

- finding all users who have a particular name e.g. Beatriz - regex not anchored at the start of the string so won't use an index efficiently (index scan):

```
Q. db.live.find( { "user.name" : /Beatriz/ } )
A. db.live.find( { "user.name" : /^Beatriz/ } )
```

- finding all users in the Brasilia timezone with more than 70 friends, and sorting by most friends - only using one index and not a compound, and sorting by non-indexed field:

```
Q. db.live.find( { "user.time_zone" : "Brasilia",
                   "user.friends_count" : { $gt : 70 } } )
             .sort( { "user.friends_count" : -1 } )
A. db.live.ensureIndex( { "user.time_zone" : 1, "user.friends_count" : 1 } )
```

Security

Premise 1

You've discovered a 3 node replica set running without SSL and not utilizing your internal LDAP service for authentication.

Setup

- Start new instances specifically for this exercise with the securityRepl.sh script, e.g. "./security.sh 2 jason" for two teams for the instructor "jason"

Solution SSL

1. Tutorial here may help: http://docs.mongodb.org/manual/tutorial/upgrade-cluster-to-ssl/
2. Look in /home/ubuntu/ssl_certs:

```
ca.pem
client.pem
crl.pem
server.pem
```

3. Rolling restart of each node with the following parameters:

```
sslMode = allowSSL
sslPEMKeyFile = /home/ubuntu/ssl_certs/server.pem
sslCAFile = /home/ubuntu/ssl_certs/ca.pem
```

4. Either update each config file again, or run the following command on each:

```
db.getSiblingDB('admin').runCommand( { setParameter: 1, sslMode: "preferSSL" } )

// Or config file changes
sslMode = preferSSL
sslPEMKeyFile = /home/ubuntu/ssl_certs/server.pem
sslCAFile = /home/ubuntu/ssl_certs/ca.pem
```

5. Update each config file again, or run the following command on each:

```
db.getSiblingDB('admin').runCommand( { setParameter: 1, sslMode: "requireSSL" } )

sslMode = requireSSL
sslPEMKeyFile = /home/ubuntu/ssl_certs/server.pem
sslCAFile = /home/ubuntu/ssl_certs/ca.pem
```

6. Checkpoint, to verify SSL is running correctly, connect to each node:

```
/usr/bin/mongo --ssl --sslCAFile ~/ssl_certs/ca.pem
    --sslPEMKeyFile ~/ssl_certs/client.pem --port 27017
/usr/bin/mongo --ssl --sslCAFile ~/ssl_certs/ca.pem
    --sslPEMKeyFile ~/ssl_certs/client.pem --port 27018
/usr/bin/mongo --ssl --sslCAFile ~/ssl_certs/ca.pem
    --sslPEMKeyFile ~/ssl_certs/client.pem --port 27019
```

Solution LDAP

1. Tutorial here may help: http://docs.mongodb.org/manual/tutorial/configure-ldap-sasl-openldap/
2. Create admin user (we'll use this user to add our LDAP user):

```
> use admin
> db.createUser(
    {
      user: "superuser",
      pwd: "mongo",
      roles: [ "root" ]
    }
)
```

3. Add a keyfile and enable authentication for the cluster:

```
> openssl rand -base64 741 > mongodb-keyfile

// Modify config files
auth=true
setParameter=saslauthdPath=/var/run/saslauthd/mux
setParameter=authenticationMechanisms=PLAIN,MONGODB-CR
keyFile=/home/ubuntu/mongodb-keyfile
```

```
Note: auth=true is not needed when a keyFile is specified,
also combination of PLAIN/MONGODB-CR can be modified
```

4. Restart all the nodes
5. Authenticate with superuser:

```
/usr/bin/mongo admin --ssl --sslCAFile ~/ssl_certs/ca.pem
    --sslPEMKeyFile ~/ssl_certs/client.pem --port 27017 -u superuser -p mongo
```

6. Add ldap user:

```
db.getSiblingDB("$external").createUser(
    {
      user : "ldapuser",
      roles: [ { role: "read", db: "records" } ]
    }
)
```

7. Log back in with new ldap user ro verify:

```
/usr/bin/mongo admin --ssl --sslCAFile ~/ssl_certs/ca.pem
    --sslPEMKeyFile ~/ssl_certs/client.pem --port 27017

> db.getSiblingDB("$external").auth(
   {
     mechanism: "PLAIN",
     user: "ldapuser",
     pwd:  "ldap",
     digestPassword: false
   }
)
```

Shutting Down & Rebuilding Clusters

To shutdown the nodes launched for each team run *shutDownRepl.sh*: this will terminate all the teams' instances but not the sharded clusters.

To shutdown the sharded clusters run *shutDownCluster.sh*: this will terminate the shards, config server, and mongos.

These will only remove the nodes launched by the previous execution of *repl.sh* and *shard.sh*. If you ran either of the launch commands twice in a row to rebuild a cluster, you will have to log on to the EC2 dashboard, filter on "Advanced Administrator Training" and terminate the instances there. At the end of an Advanced Administrator Training session, *ALWAYS* log on to the EC2 dashboard to make sure all of the instances launched by you have been terminated.