MongoDB Advanced Administrator Training

Introduction

Advanced Administrator Training is a 2-day on-site course designed for DBAs. The instructor leads teams through a series of scenarios that represent potential issues encountered during the normal operation of MongoDB. Teams are hands-on throughout the course, actively working together to identify and implement solutions. The instructor evaluates teams based on the speed, elegance, and effectiveness of their solutions and provides feedback on strategy and best practices.

Exercises

Each scenario exercises and develops skills in critical areas such as diagnostics, troubleshooting, maintenance, performance tuning, and disaster recovery.

Each team will be provided with their own Amazon EC2 machines to work with. The instructor will supply 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*. Refer to the linked instructions for using PEM files with PuTTY[1].

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.

The instructor will also have SSH access to all the instances in order to run scripts that set up each scenario.

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.

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

Replication

Premise 1

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

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`.

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.

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.*"

Aggregation Framework + Map-Reduce

Objective

The objective of these exercises is practice as many of the aggregation framework operators as possible.

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.

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
}
```

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
}
```

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.*"

Instances

For this section, teams will use different EC2 machines. The instructor will prove the IP address for a `mongos` instance connected to a sharded cluster.

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.

Security

Objective

The objective of this exercise is to work through several security concepts in MongoDB.

Premise

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

The goals of this exercise are to convert a running 3 node MongoDB replica set to use SSL for encrypted traffic and LDAP for user authentication.

1. Connect to your team's instance with the "ubuntu" user for this exercise:

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

2. Verify the 3 node replica set is running (locally, on ports 27017, 27018, 27019):

   ```
   /usr/bin/mongo
   > rs.status()
   ```

3. Verify the local OpenLDAP server is running:

   ```
   sudo testsaslauthd -u ldapuser -p ldap
   ```

All certificates needed are in /home/ubuntu/ssl_certs

Now you must determine the steps needed to convert this 3 node replica set to use SSL and LDAP for authentication. Upon completing the exercise, the following command will return success:

```
/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
  })
```