MongoDB Advanced Administrator Training

Release 3.4

MongoDB, Inc.

Mar 24, 2017

Contents

1 Advanced Administrator

1.1 Advanced Administrator

Learning Objectives

Upon completing this training students should understand:

- How to install and configure a highly available Ops Manager deployment
- Understand all the necessary components and architecture choices for an Ops Manager deployment
- How to effectively manage clusters using Ops Manager
- How to deploy and operate secured MongoDB deployments

Hands-on Approach

This training is a full hands-on experience.

- You will be given access to a set of AWS instances.
- All the necessary software will be available within those same instances
- However, all architecture decisions and configuration steps will be made by students
- You are expected to work in teams. So you will be sharing a set of machines with your colleagues.
- Use the instructor for guidance and advice but he should mostly be there for observation and time boxing the labs.

Note: At this point you should have created the infrastructure for the students and be retrieving its configuration by running the following command that will create the /tmp/advadm* files with all the info about IPs, LBs, VPCs:

```
./describe.py --run <TRAINING-RUN> --out /tmp/advadmin
```

- show the students how many teams have been created
- show how to install the machines key file in the ssh agent

```
ssh-add -K AdvancedAdministrator.pem
```

- exemplify how they have access to the machines by making an ssh connection with key forwarding

```
ssh -A -i AdvancedAdministrator.pem centos@<AWS-INSTANCE-PUBLIC-IP>
```

- Access key file should be available through training portal files

---

Expected Takeaways

There are a few important objectives that we want to accomplish by the end of this course:

- Understand the necessary infrastructure needed to run Ops Manager
- Understand the different architecture choices and their tradeoffs in different deployments
- Understand the different options that Ops Manager allows for backup
- Clear understanding of the benefits of using Ops Manager to manage different clusters
- Deploy secured, monitored, and fully managed infrastructure for your application

---

Note: Make sure the students understand that, for this completion of this course, they are expected to have a significant experience with MongoDB, and that we assume that terms like:

- Replica Set
- Shard Cluster
- mongos
- mongod
- config servers
- indexes
- shard keys
- ...

are very familiar to the them.

Take your time, ask questions

It's important to foster the discussion of different options and review options so:

- Use the whiteboard (if available)
- Talk to your team members to defined clear tasks and responsibilities
- Use the instructor for guidance and ask for advice
- Take chances, break stuff!

Note: Reinforce that this is the perfect opportunity for students to make experiments and test configuration options.

You are there to guide them but they should make use of this time to actually put in practice some ideas and check if they would work, in the context of the exercises in hand

At this point you should assemble the teams and share the teams machine details files.

Let's review what we have

Once we have our teams assembled it is time to do our first checklist.

Within your team configuration file you should have:

- Load balancer address.
- Public and private ip address for each AWS instance.
- An internal VPC set of ip addresses for each instance.
- 3 <*opsmgr*> instances.
- Up to 12 <*node*> instances.
- *AdvancedAdministrator.pem* key file to allow access to the instances.

Note: At this point you should make sure the students have all the necessary files needed to access the infrastructure.

- everyone should be assigned to one team, max of 3 person teams
- check if everyone has their team configuration file
- check that everyone has their key file and successfully added the key to their ssh-agents
- check that everyone is able to reach the AWS instances
  - ask students to connect to one of their designated team instances.
- check that everyone is able to run command `ls /share`

You may want to explain to the students that:

- each team works in under their private vpc
- to navigate and configure the infrastructure, students should be using the `/etc/hosts` defined names

---

Within our instances ...

In our machines we will have access to all nodes in the deployment:

```
cat /etc/hosts
```

A `/share` folder with all necessary software downloaded

```
ls /share
```

And you should have MongoDB installed in your instances:

```
mongod --version
```

Note: Connect to one of the instances and run all the commands yourself.

```
ssh -A -i AdvancedAdministrator.pem centos@<AWS-INSTANCE-PUBLIC-IP>

ls /share
mongod --version
```

This is also a good opportunity to show that if you forward the ssh key you will be able to jump to all machines of a given team.

```
ssh -A -i AdvancedAdministrator.pem centos@<AWS-INSTANCE-PUBLIC-IP>

ssh -A centos@10.0.0.101

ssh -A centos@10.0.0.22
```

1.2 Lab: Ops Manager Installation

Premise

Ops Manager is a solution for On-Prem MongoDB cluster operational management.

Enables features like:

- Automation
- Backup and Recovery
- Monitoring

Over the course of this lab we will be installing Ops Manager with high availability and scalability in mind.

Note: For details on how to setup clusters for this class, see:

- https://docs.google.com/document/d/1vhA6NvlTsPe1rw_fb7N5NrYzJ78odiHWBd5yf9vPd64

For this lab we will group students into teams.

- There will be a maximum of 9 teams!
- Each team will have at their disposal 15 virtual machines
  - 3 Ops Manager hosts
  - 12 Node hosts
- Verify that all students have successfully pinged their designated hosts
- These hosts have role names and public IPs/DNS names

Ops Manager HA

Ops Manager requires a number of servers for high availability.

- Monitoring and backup/recovery are essential for production operations.
- Therefore, it's important to assure high availability for Ops Manager.
- For this we need to follow a specific deployment topology.

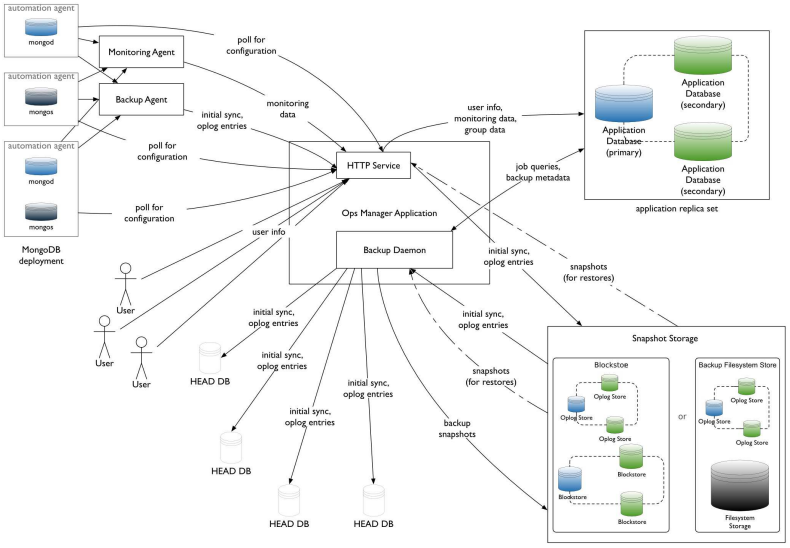
Ops Manager Scalability

Why do we need our operations tool to be scalable?

- The main reason is backup and recovery requirements
- The amount of data individual applications generate will grow
- The number of applications your Ops Manager deployment supports will grow
- Plan to accommodate both forms of growth

Ops Manager Architecture Review

Let's review the Ops Manager architecture[1] :



Note: For this diagram raise students attention to the following:

- Ops Manager is composed out of two main services
  - HTTP Service: Ops Manager Application
  - Backup daemon
- Ops Manager will collect data from the different agents

[1] https://docs.opsmanager.mongodb.com/current/core/system-overview/

  - backup
  - monitoring
  - automation
- Ops Manager has three main datastores
  - Application DB (MongoDB Cluster)
    * user info
    * monitoring data
    * group data
    * backup job queries
    * backup metadata
  - Backup DB (MongoDB Cluster)
    * OpLog store
    * Sync store
    * Blockstore: data snapshots, divided in small chunks being compressed and de-deduplicated
    * or Filestore: data snapshots as files
  - Head Database
    * Keeps track of oplog and initial syncs
    * There's one head database for each backed-up replica set
    * Similar load as a Secondary

Launch the Ops Manager MongoDB Instances

It's time to set up the our hosts that will support our Ops Manager Deployment.

We will need the following instances:

- Two replica sets of 3 data nodes
  - Application Database replica set as **APPDB**
  - Backup Database replica set as **BACKUPDB**
- Three instances to run redundant services of Ops Manager Application
  - The hosts that will be supporting this deployment:
    * `opsmgr1`, `opsmgr2` and `opsmgr3`
- Load Balancer
  - The load balancer is already set up and its name should be given to you by the trainer.

Note: Ask students to make a deployment and configuration plan. Use the following questions as guide:

- Where do we start ?
  - Boot up Ops Manager Application replica set
- Do we need a replica set to support the Backup Daemon?

  - Yes, you always need one, and we will be using the 'Blockstore' part of the Backup DB
    * we can use a Filestore as an alternative but for the exercise let's use the Blockstore
- 2 possible solutions:
  - each `opsmgr` node gets App, BackupDaemon, AppDB and BackupDB
  - `opsmgr` gets App, BackupDaemon and BackupDB, `nodes` get AppDB

---

1. Configure Ops Manager Application Database

Ops Manager needs to store data:

- Configuration of nodes, groups, users
- Metrics for monitoring
- Backup metdata and job queries

Also consider relevant security settings[2] for this database.

From the available machines go ahead and set up a replica set to support the *Application Database*.

Note:

- Students should be assembling a 3 node replica set to allow the HA.
- To validate their deployment you should ask students to run validation script
  - `validate_appdb.py`

Instruction set: - Connect to each `opsmgr` instance and launch a mongod

```
mongod -f /share/etc/appdb.conf
```

- The */share/etc/appdb.conf* file should have the following settings:

```
cat /share/etc/appdb.conf
replication:
  replSetName: "APPDB"
  oplogSizeMB: 100
storage:
  wiredTiger:
    engineConfig:
      cacheSizeGB: 2
      journalCompressor: zlib
net:
  port: 27001
```

- Once all instances `mongod` instances are up initiate the replica set

```
# replace opsmgr{1,2,3} with corresponding internal ip addresses
mongo --host opsmgr1
> rs.initiate(
  _id: 'APPDB',
  members: [
  {_id: 1, host:'opsmgr1'},
  {_id: 2, host:'opsmgr2'},
```

[2] https://docs.mongodb.com/manual/administration/security-checklist/

```
  {_id: 3, host:'opsmgr3'},
  ]
)
```

---

2. Configure and Launch the Ops Manager Service

Habemus Replica Set!

Now it's time to launch the **Ops Manager** service. For this you will need to:

- Edit Ops Manager configuration `conf-mms.properties`:
  - Follow instructions to install from rpm[3]
  - The file can be found at */share/downloads/mongodb-mms-XXX-X.x86_64.rpm*
  - Point these to the previously configured replica set: **APPDB**
  - Allow offline binary access[4]
- Launch the Ops Manager service

Note: We are striving for an Highly Available Ops Manager deployment therefore:

- Before mentioning *again* the HA requirement you may ask students why we need 2 `opsmgr` nodes?
- Make sure students use machines that are tagged with Ops Manager

You may want to point to the following documentation page:

configure HA app[5]

- Generate `gen.key`

```
ssh -A centos@opsmgr1
#install ops manager server
yum install -y /share/downloads/mongodb-mms-2.0.6.363-1.x86_64.rpm
#edit the configuration options
vi /opt/mongodb/mms/conf/conf-mms.properties
#generate a gen.key file
openssl rand 24 > /share/gen.key
cp /share/gen.key /etc/mongodb-mms/

#copy this generated file to all opsmgr hosts
scp /share/gen.key centos@opsmgr2:/etc/mongodb-mms/opsmgr.key
scp /share/gen.key centos@opsmgr3:/etc/mongodb-mms/opsmgr.key
#make sure you use replace opsmgr for the host ip
```

- Edit Ops Manager configuration file `/etc/mongodb-mms/conf-mms.properties`

```
#replace mongo.mongoUri with APPDB replica set connection string
# mongodb://opsmgr1:27001,opsmgr2:27001,opsmgr3:27001/?replicaSet=APPDB
sed -i.bak 's/^\(mongo.mongoUri=\).*/\1mongodb:\/\/opsmgr1\:27001,opsmgr2\:27001,
↪opsmgr3\:27001\/\?replicaSet=APPDB/' /etc/mongodb-mms/conf-mms.properties
#or just use vi
```

[3] https://docs.opsmanager.mongodb.com/current/tutorial/install-on-prem-with-rpm-packages/#install-the-onprem-package-on-each-server-being-used-for-onprem
[4] https://docs.opsmanager.mongodb.com/current/tutorial/configure-local-mode/
[5] https://docs.opsmanager.mongodb.com/current/tutorial/configure-application-high-availability/

```
#repeat the operation on all opsmgr hosts
```

- Launch Ops Manager on all `opsmgr` hosts

```
ssh -A centos@opsmgr1 'service mongodb-mms start'
ssh -A centos@opsmgr2 'service mongodb-mms start'
ssh -A centos@opsmgr3 'service mongodb-mms start'
```

To validate this exercise the students should be able to:

- Register admin user credentials and Ops Manager group

---

3. Install Ops Manager Automation Agents

At this point **Ops Manager** should be up and running. Now it's time to install our Automation Agents[6]:

- In the remaining VMs, install the automation agent
- Make sure that all nodes are discoverable on the servers dashboard
- Validate that all agents are reporting pings correctly

Note: You should ask the students to install the agents on the remaining VMs

- Validate that all nodes are discoverable
- All agents are reporting data without errors

1.3 Lab: Enable the Ops Manager Public API

Learning Objectives

Upon completing this lab, students will be able to:

- Understand the requirements for enabling Ops Manager Public API
- Configure Ops Manager groups to allow Public API requests

[6] https://docs.opsmanager.mongodb.com/current/tutorial/nav/install-automation-agent/

Exercise: Selective Node Rest Client

Ops Manager enables administrators to determine from where Public API calls are allowed. From which client nodes it accepts such interface requests.

Enable your deployment of Ops Manager to allow only one specific client to perform API calls.

- Generate an API Key called “generic”
- Add CIDR block for ip whitelisting enabling

Note: Enabling the Public API Ops Manager will require an ip whitelist CIDR block.

- The most permissive CIDR Block is 0.0.0.0/0
- for this exercise students should choose
  - a particular machine
  - or set of machines to accept API requests only from such nodes.

ex:

```
// accept only from 120.10.10.132
120.10.10.132/32

// from 123.10.10.132 to 123.10.10.135
123.10.10.132/30
```

Exercise: Enable Group Public API

Groups are more than just sets of machines and MongoDB instances. Groups also enclose security considerations.

Administrators have the ability to enable the Public API interface on a per group basis.

For this exercise go ahead and:

- Create a group called “LOVE_API_CALLS”
- Enable Public API for this group

Note: Students should

- create the group
- enable the API calls

To validate this you should ask students to submit the following request

```
// set APIKEY with `generic` key value
curl -u "admin@localhost.com:$APIKEY" --digest
-i "http://opsmgr.training/api/public/v1.0/groups"
```

1.4 Lab: Ops Manager User Administration

Learning Objectives

Upon completing this lab, students will be able to:

- Administer Ops Manager groups
- Identify the differences between Ops Manager user roles
- Create and define Ops Manager users

Exercise: Create Group

Connect to your Ops Manager instance and create the following `group`:

- **CIRCUS_MAXIMUS**

Note: This is a very simple exercise.

Take the time to explore the `admin` menu with students.

Exercise: Create Users

Using the Ops Manager API[7] , create the following users:

- **aediles@localhost.com** :
  - password: "123ABCabc!"
  - role: Owner[8]
- **patrician@localhost.com** :
  - password: "123ABCabc!"
  - role: Monitoring Admin[9]
- **consus@localhost.com** :
  - password: "&o7chac0v3r3d"
  - role: Backup Admin[10]

Note: To accomplish this task, the users will have to do the following steps:

- enable the api[11]
- enable the public api on the group
- create different users using the HTTP Rest API

e.g.: **Owner**

[7] https://docs.opsmanager.mongodb.com/current/api/
[8] https://docs.opsmanager.mongodb.com/current/reference/user-roles/#owner
[9] https://docs.opsmanager.mongodb.com/current/reference/user-roles/#monitoring-admin
[10] https://docs.opsmanager.mongodb.com/current/reference/user-roles/#backup-admin
[11] https://docs.opsmanager.mongodb.com/current/tutorial/enable-public-api/

```
curl -u "admin@localhost.com:$APIKEY" -H "Content-Type: application/json"
--digest -i -X POST "http://opsmgr.training/api/public/v1.0/users" --data '
{
"username": "aediles@localhost.com",
"emailAddress": "aediles@localhost.com",
"firstName": "This",
"lastName": "Mine",
"password": "aLLm1n3$3cr3t",
"roles": [{
  "groupId": "$GROUPID",
  "roleName": "GROUP_OWNER"
  }]
}'
```

Exercise: Create Global Users

In various different situations, we will need users with global roles. Please create, either through the API or web console, the following users:

- First user with Global Automation Admin[12] role : *automater@localhost.com*
- Second user granted Global User Admin[13] user : *masterchef@localhost.com*

After creating these users, connect with the most appropriate user to change the password of the **CIRCUS_MAXIMUS** *Owner* user. The new password should be *"$uperC00l"*

This last operation should be accomplished using the HTTP Rest API interface.

Note: Make sure that students understand which user to log in with and how to change the user password of *thisismine@localhost.com* user.

For the operation students will have to do the following steps:

- enable Public API for user *masterchef@localhost.com*
- change user *aediles@localhost.com* password through the API

```
// get the user
curl -u "masterchef@localhost.com:$APIKEY"
--digest -i "http://opsmgr.training/api/public/v1.0/users/byName/aediles@localhost.com
↪"
curl -u "masterchef@localhost.com:$APIKEY" -H "Content-Type: application/json"
--digest -i -X PATCH "http://opsmgr.training/api/public/v1.0/users/$USERID" --data '
{
"password": "$uperC00lpa22"
}'
```

[12] https://docs.opsmanager.mongodb.com/current/reference/user-roles/#global-automation-admin
[13] https://docs.opsmanager.mongodb.com/current/reference/user-roles/#global-user-admin

1.5 Lab: Secure Replica Set

Premise

- Setting up a MongoDB Replica set is quite easy and fast.
- Setting up a Secured MongoDB replica set requires a few extra steps.
- In this lab we will be exploring how to setup a secured Replica Set through Ops Manager.

Note: Security is an important topic of production deployments and we want the students to be fully acquainted with the different options of MongoDB Security.

X.509 Authentication Mechanism

We will be using X.509 certificates[14] for authentication and TLS/SSL network encryption.

Note: The actual X.509 details are out-of-scope for this training. Our purpose is **not**:

- to educate people on the authentication mechanism itself
- detailed explanation on how TLS/SSL works

Our purpose is to:

- Review the different authentication mechanisms
- How students can use such mechanism if they choose too
- The tradeoffs of X.509 when compared with other auth mechanisms

---

Ops Manager Group SSL and Auth

To build secured MongoDB deployments you first need to enable Auth and SSL[15] on your group.

All VMs have a set of certificates that you will be using to configure your secured deployment.

In folder */shared/etc/ssl* you will find:

- `ca.pem`: SSL CA certificate
- `automation.pem`: Automation agent certificate
- `backup.pem`: Backup agent certificate
- `monitor.pem`: Monitoring agent certificate
- `nodeX.pem`: Replica set member certificates (X)
- `dbadmin.pem`: MongoDB DB Admin certificate

Note: Ask students to list the files under */shared/etc/ssl* on instances to validate that their installation process is correct.

Make sure to highlight that:

[14] https://docs.mongodb.com/manual/core/security-x.509/
[15] https://docs.opsmanager.mongodb.com/current/tutorial/enable-ssl-for-a-deployment/

- The enabling of auth and ssl on a group level is to ensure correct communicate between all instances
- Ensuring the same Certificate Authority (CA) certificate
- Enabling agents to perform their normal operations
- Create the required agent users

`VERYSAFE` Group

Let's start by creating a group called `VERYSAFE` that has SSL enabled.

- Using the existing certificates, configure the agents accordingly.
- You need to specify certificates for
  - Certificate Authority
  - Monitoring Agent
  - Backup Agent
  - Automation Agent
- **The existing certificates do not have any decryption password!**

Note: This might be a bit hard for students that are not experienced with Ops / Cloud Manager

- Take the time to navigate users through the UI to configure these settings
- Make sure that students do not provide a decypher password for the certificates

Once the group is created you will need to reconfigure your existing agents to use the new group

- Either you install new agents or just stop the service and reconfigure the agents configuration file

```
sudo service mongodb-mms-automation-agent stop
#edit /etc/mongodb-mms/automation-agent.config and add new APIKey and GroupId
sudo service mongodb-mms-automation-agent start
```

- students will find the corresponding instructions on *Settings -> Agents -> Host Version Agent*

Secure Replica Set Deploy

Once the automation agent has been reconfigured and servers are detected on your deployment, it's then time to deploy our secure replica set.

Create a replica set named **SECURE** with the following configuration:

- 3 Nodes:
  - **node1**, **node2** and **node3**
  - Port *27000*
- **clusterAuthMechanism**: x509
- **sslMode**: requiredSSL
- **sslPEMKeyFile**: */shared/etc/ssl/nodeX.pem*

Note: Students should create a replica set from Ops Manager UI that will reflect the wanted configuration:

- Once the `VERYSAFE` group has been create we need to **Add -> New Replica Set**
  - Name: SECURE
  - Port Range: **27000**
  - DB Path Prefix: `/data`
  - clusterAuthMechanism: **x509**
  - sslMode: `requiredSSL`
  - Apply
- Once we have applied we now need to **Modify** the individual node members with the corresponding server **sslPEMKeyFile**
  - node1: */shared/etc/ssl/node1.pem*
- `nodeX.pem`: there should be a certificate file per each node.
- This setting needs to be configured on a per instance level.

X509 Users

Time to create users that will authenticate using an X.509 certificate.

- Go ahead and create a dbAdminAnyDatabase[16] user that authenticates using `dbadmin.pem` certificate.
- To create users that authenticate using X509 certificates you should check Certificate Subject as user[17] docs
- After the user has been created, connect to the *Primary* node of the replica set and create database "`allgood`".

Note:

- Students might not be familiar with this mechanism so point them to Certificate Subject as user[18].
- Students will have to extract the certificate `subject` info

```
openssl x509 -in /share/etc/ssl/dbadmin.pem -inform PEM -subject -nameopt RFC2253
subject= C=US,ST=New York,L=New York City,O=MongoDB,OU=EDUCATION,CN=dbadmin
-----BEGIN CERTIFICATE-----
MIIDezCCAmOgAwIBAgIDBZc1MA0GCSqGSIb3DQEBBQUAMGUxCzAJBgNVBAMTAkNB
```

  and use that subject info to create the required user.
- We can create this user using the Ops Manager UI or we can connect to the replica set and perform the following operations:
- Authenticate to primary node using automation agent certificate

```
mongo --host PRIMARY_NODE --ssl --sslPEMKeyFile /share/etc/ssl/automation.pem --
↪sslCAFile /share/etc/ssl/ca.pem
> db.getSiblingDB("$external").auth({
  mechanism: "MONGODB-X509",
```

[16] https://docs.mongodb.com/manual/reference/built-in-roles/#dbAdminAnyDatabase
[17] https://docs.mongodb.com/manual/tutorial/configure-x509-client-authentication/#add-x-509-certificate-subject-as-a-user
[18] https://docs.mongodb.com/manual/tutorial/configure-x509-client-authentication/#add-x-509-certificate-subject-as-a-user

```
  user:"C=US,ST=New York,L=New York City,O=MongoDB,OU=EDUCATION,CN=automation"
  })
```

- Create the new `dbAdminAnyDatabase` user with the `dbadmin.pem` certificate

```
db.getSiblingDB("$external").runCommand({
  createUser: "C=US,ST=New York,L=New York City,O=MongoDB,OU=EDUCATION,CN=dbadmin
↪",
  roles: [
    { role: 'dbAdminAnyDatabase', db: 'admin' }
  ],
  writeConcern: { w: "majority" , wtimeout: 5000 }
})
```

- Connect to primary using `dbadmin.pem`:

```
mongo --host PRIMARY_NODE  --ssl --sslPEMKeyFile /share/etc/ssl/dbadmin.pem
--sslCAFile /share/etc/ssl/ca.pem
```

- Authenticate and create new database `allgood`

```
db.getSiblingDB("$external").auth({
  mechanism: "MONGODB-X509",
  user:"C=US,ST=New York,L=New York City,O=MongoDB,OU=EDUCATION,CN=dbadmin"
})

db.createDatabase("allgood")
```

1.6 Lab: Reconfig Replica Set

Learning Objectives

Upon completing this lab, students should be able to:

- Reconfigure a replica set
- Outline the different stages of deployments

Dependencies

- In order to complete all purposed exercises we need to first enable the Public API.
- Go to your group settings and enable the Public API.
- Do not forget to set an appropriate CIDR block for the IP whitelist and Generate the API Key.

Note: Make sure students are aware of this dependency.

Exercise: Initial Replica Set

- Using the Ops Manager UI go ahead and create a 3 node replica set:
  - Replica set name `META`
  - Two data bearing nodes
    - use hosts: `node3` and `node4`
  - One arbiter
    - use host: `node5`
  - All nodes should be set to use port **27000**

Note: All instances should be installed using MongoDB 3.2.1 enterprise

---

Exercise: Add Replica Set Members

- Let's assume that we require highler level of High Availability (HA).
- Add 2 new data bearing nodes
  - First node should have priority 0
    - use `node6`
  - Second node should be an hidden replica.
    - use `node8`

Exercise: Decomission Replica Member

- One of your nodes is not making the cut. - Not pointing fingers but ... `node3` is *acting up*
- Change your replica set by *"decommissioning"* one of the instances
- Make sure that your replica set keeps up majority and previous level of node failure resilience

---

Note: Make sure students are aware that removing one of nodes might require adding an extra one.

- Make sure that the they will need to chang the Replica Set configuration priroities if they don't want to just remove the problematic node
  - In this case `node3`
- Changing the priorty of `node6` to 1 will keep the same availabilty
- We can also remove the `arbiter` to keep an odd number of nodes

Questions to ask:

- Q: What happens if we just remove the problematic node and the arbiter?
  - A: This is not enough to guarantee the same availability. The remaining nodes are not capable of becoming **primary** nodes

---

Exercise: Upgrade MongoDB Version

- Our CTO, for compliance reasons, demands that all of our nodes should be on the lattest version of MongoDB.
- Upgrade all nodes in your replica set without downtime.

Note:

- Please instruct students to change the version of MongoDB to the most recent version of MongoDB
- Make sure they know how to make the required version available in their Ops Manager deployment.
- Update the Version Manifest: Deployment -> Version Manager -> Update MongoDB Version Manifest

---

Exercise: Update Node Priority

Our initial setup is not in line with the expectations of the CTO in terms of hierarchy *(talking about micromanagement!)*.

- Update the priorities of nodes to the following configuration:
  - **node4**: 10
  - **node6**: 7
  - **node5**: Arbiter
  - **node8**: 0 and slave delayed by 10hours
- All of these changes should be done using the Ops Manager API!

Note: This is the most challenging exercise of this lab. Walk the students through the exercise. Students will have to dig into the API Documentation[19] to retrive the necessary instructions.

Students will also have to enabled the public api and generate a public API key.

Here's the set of instructions they will need to accomplish:

- set a variable containing ops manager admin username

```
export username=admin@localhost.com
```

- Get the group Id and add it to an environment variable
- Settings -> Group Settings (group id)

```
export GROUPID=YOURGROUPID
```

- Create a new API key
- Settings -> Public API Key -> Generate

```
export apiKey=YOURAPIKEY
```

- Get current automation configuration

```
curl -u "$username:$apiKey" -H "Content-Type: application/json" --digest -i
"/api/public/v1.0/groups/$GROUPID/automationConfig"
```

[19] https://docs.opsmanager.mongodb.com/current/reference/api/automation-config/#update-automation-configuration-via-api

- Save the json content payload into a file `automation.json`
- Edit that file to reflect the requested priorities changes


```
{
  "auth": {
      "disabled": true,
      "usersDeleted": [],
      "usersWanted": []
  },
  "backupVersions": [],
  "indexConfigs": [],
  "mongoDbVersions": [
      {
          "builds": [
              {
                  "bits": 64,
                  "gitVersion": "ed70e33130c977bda0024c125b56d159573dbaf0",
                  "platform": "linux",
                  "url": "http://opsmgr.vagrant.dev:8080/automation/mongodb-releases/
↪linux/mongodb-linux-x86_64-3.2.1.tgz"
              }
          ],
          "name": "3.2.1"
      }
  ],
  "monitoringVersions": [
      {
          "baseUrl": null,
          "directoryUrl": null,
          "hostname": "N",
          "name": "3.9.1.238-1"
      }
  ],
  "options": {
      "downloadBase": "/var/lib/mongodb-mms-automation",
      "downloadBaseWindows": "%SystemDrive%\\MMSAutomation\\versions"
  },
  "processes": [
      {
          "args2_6": {
              "net": {
                  "port": 27000
              },
              "operationProfiling": {},
              "replication": {
                  "replSetName": "META"
              },
              "storage": {
                  "dbPath": "/data/META_0"
              },
              "systemLog": {
                  "destination": "file",
                  "path": "/data/META_0/mongodb.log"
              }
          },
          "authSchemaVersion": 5,
          "hostname": "node4",
          "logRotate": {
```

```
            "sizeThresholdMB": 1000.0,
            "timeThresholdHrs": 24
        },
        "name": "META_0",
        "processType": "mongod",
        "version": "3.2.1"
    },
    {
        "args2_6": {
            "net": {
                "port": 27000
            },
            "operationProfiling": {},
            "replication": {
                "replSetName": "META"
            },
            "storage": {
                "dbPath": "/data/META_1"
            },
            "systemLog": {
                "destination": "file",
                "path": "/data/META_1/mongodb.log"
            }
        },
        "authSchemaVersion": 5,
        "hostname": "node6",
        "logRotate": {
            "sizeThresholdMB": 1000.0,
            "timeThresholdHrs": 24
        },
        "name": "META_1",
        "processType": "mongod",
        "version": "3.2.1"
    },
    {
        "args2_6": {
            "net": {
                "port": 27000
            },
            "operationProfiling": {},
            "replication": {
                "replSetName": "META"
            },
            "storage": {
                "dbPath": "/data/META_2"
            },
            "systemLog": {
                "destination": "file",
                "path": "/data/META_2/mongodb.log"
            }
        },
        "authSchemaVersion": 5,
        "hostname": "node5",
        "logRotate": {
            "sizeThresholdMB": 1000.0,
            "timeThresholdHrs": 24
        },
        "name": "META_2",
```

```
            "processType": "mongod",
            "version": "3.2.1"
        },
        {
            "args2_6": {
                "net": {
                    "port": 27000
                },
                "operationProfiling": {},
                "replication": {
                    "replSetName": "META"
                },
                "storage": {
                    "dbPath": "/data/META_3"
                },
                "systemLog": {
                    "destination": "file",
                    "path": "/data/META_3/mongodb.log"
                }
            },
            "authSchemaVersion": 5,
            "hostname": "node3",
            "logRotate": {
                "sizeThresholdMB": 1000.0,
                "timeThresholdHrs": 24
            },
            "name": "META_3",
            "processType": "mongod",
            "version": "3.2.1"
        },
        {
            "args2_6": {
                "net": {
                    "port": 27000
                },
                "operationProfiling": {},
                "replication": {
                    "replSetName": "META"
                },
                "storage": {
                    "dbPath": "/data/META_4"
                },
                "systemLog": {
                    "destination": "file",
                    "path": "/data/META_4/mongodb.log"
                }
            },
            "authSchemaVersion": 5,
            "hostname": "node8",
            "logRotate": {
                "sizeThresholdMB": 1000.0,
                "timeThresholdHrs": 24
            },
            "name": "META_4",
            "processType": "mongod",
            "version": "3.2.1"
        }
    ],
```

```
  "replicaSets": [
      {
          "_id": "META",
          "members": [
              {
                  "_id": 0,
                  "arbiterOnly": false,
                  "hidden": false,
                  "host": "META_0",
                  "priority": 10.0,
                  "slaveDelay": 0,
                  "votes": 1
              },
              {
                  "_id": 1,
                  "arbiterOnly": false,
                  "hidden": false,
                  "host": "META_1",
                  "priority": 7.0,
                  "slaveDelay": 0,
                  "votes": 1
              },
              {
                  "_id": 2,
                  "arbiterOnly": true,
                  "hidden": false,
                  "host": "META_2",
                  "priority": 0,
                  "slaveDelay": 0,
                  "votes": 1
              },
              {
                  "_id": 3,
                  "arbiterOnly": true,
                  "hidden": false,
                  "host": "META_3",
                  "priority": 0,
                  "slaveDelay": 0,
                  "votes": 1
              },
              {
                  "_id": 4,
                  "arbiterOnly": false,
                  "hidden": true,
                  "host": "META_4",
                  "priority": 0,
                  "slaveDelay": 36000,
                  "votes": 1
              }
          ]
      }
  ],
  "roles": [],
  "sharding": [],
  "uiBaseUrl": null,
  "version": 2
}
```

- Update the automation configuration

```
export username=yourusername
curl -u "$username:$apiKey" -H "Content-Type: application/json" --digest -i -X PUT
"https://opsmgr.training/api/public/v1.0/groups/$GROUPID/automationConfig" --data␣
↪@automation.json
```

1.7 Lab: Shard Cluster

Learning Objectives

Upon completing this lab, students will be able to:

- Create a sharded cluster using Ops Manager
- Identify the necessary steps to configure the cluster
- Create the correct shard key for a given dataset
- Understand Zone sharding
- Detect balancing issues

Exercise: Create Shard Cluster

Using the Ops Manager UI, let's create a MongoDB sharded cluster with the following configuration:

- Two shards cluster, with three nodes per shard, distributing the process like that:
  - **shard001**: `node1`, `node2` and `node3`
  - **shard002**: `node4`, `node5` and `node6`
  - **config servers**: 3 processes on `node9`
  - **mongos**: `opsmgr1`, `node10` and `node11`
  - Each `mongod` should be running on different hosts
  - All config servers[20] should be running on a single host
    - These should be placed on host `node10`

Note: The idea behind this exercise is to have students setup the cluster using the regex rules, on server name selection, that ops manager allows.

We also want the students to change the configuration in following labs:

- point out that this configuration is there to be changed
- should **not** be followed in production

[20] https://docs.mongodb.com/manual/core/sharded-cluster-config-servers/

To validate the students implementation check the following:

- after running `describe.py` on this training run get all the private and public ips to validate that the defined clusters follow the expected recipe

---

Exercise: Correct Config Servers Distribution

Like Britney Spears used to say *"Oops, I did it again"*, we made a mistake on our previous setup and installed all our config servers[21] on a single host.

So now we need to fix our deployment by doing a configuration change:

- Edit the cluster configuration by setting the config servers[22] into separate instances
  - They should be placed on `node9`, `node8` and `node7`

Note: In this exercise we want the students to understand that these operations will happen with no down time.

To validate this exercise, after the reconfiguration is complete, ask the students to connect to the defined hosts and verify that the running instance is infact a config server node.

Exercise: Detect Node Down

Our shard cluster is composed by several different nodes (mongods) running on several different hosts.

It's critical that we keep an eye on our cluster. Using the tools available to you, create the necessary mechanisms to be notified in the event of a node failure.

Note: In this exercise we want students to create a set of alerts:

- for host failure
- for process failure

The instruct will be responsible of injecting failures in the teams environment

To *inject process failure* we need to connect to a designated host (instructor should pick one) and kill the process(es) running on that host.

For *host failure* to be triggered, the instructor should connect to aws console[23] and stop the execution of a given host.

These can also be `opsmgr` instances where students should be monitoring the running Application DB replica set.

---

[21] https://docs.mongodb.com/manual/core/sharded-cluster-config-servers/
[22] https://docs.mongodb.com/manual/core/sharded-cluster-config-servers/
[23] https://console.aws.amazon.com/console/home

Exercise: Configure Shard

Time to use our distributed database in it's full power.

We will be using a dataset of US consumer complaints. These are records of complaints, on several sectors/states/companies, filed by US consumers.

The dataset should be imported as collection *"complaints"* in the database *"consumer"*, which can also be referred as the **"complaints.consumer"** namespace.

We also want you to configure a few settings:

- set the chunksize[24] to 1MB (the smallest allowed)
- set the primary shard to max shard size[25] of 500MB (512)

Note: The idea behind these configuration options are: - a) to get user administrators acquainted with them - b) create the conditions to simulate jumbo chunks[26] situation

Make sure to note that using the smallest size allowed or setting such a low shard max size will have implications. Mostlikely, other values should be considered for production environments

For the exercise students should do the following: - set chunk size and primary shard[27] max size

```
use config
db.settings.save( { _id:"chunksize", value: 1 } )
db.runCommand({
  addShard: "<replica_set>/<hostname><:port>",
  maxSize: 512 })
```

Exercise: Configure Shard (continued)

Let's go ahead and import the **consumer** dataset that is available in the `opsmgr` instances in the folder `/dataset/consumer`:

- import/restore this dataset into the *consumer.complaints* namespace
- Once data is imported let's shard the `complaints` collection using the following shard key:

```
sh.enableSharding('consumer')
sh.shardCollection('consumer.complaints', {company:1, state: 1})
```

Note: The above set of instructions are incomplete. We need a prior step, before running `db.shardCollection` command!

Which command is it ?

Note: To select a correct shard key this should have a few properties:

- query isolation
- write distribution

[24] https://docs.mongodb.com/manual/tutorial/modify-chunk-size-in-sharded-cluster/
[25] https://docs.mongodb.com/manual/tutorial/manage-sharded-cluster-balancer/#sharded-cluster-config-max-shard-size
[26] https://docs.mongodb.com/manual/core/sharding-data-partitioning/#jumbo-chunk
[27] https://docs.mongodb.com/manual/core/sharded-cluster-shards/#primary-shard

- high cardinality
- non-monotonically growing
- key value frequency
- key immutable
- key values are also immutable

Make sure you higlight these with the students

For this exercise we will not take all of them in consideration, and just use **company** and **state** to attend the expected distribution.

The missing command is the creation of an index on the selected shard key fields:

```
use consumer
db.complaints.createIndex({company:1, state: 1})
```

There are several aspects of the selected shard key that are not optimal:

- Frequency is inappropriate. If we run the following aggregation query we can see that standard deviation is quite of

```
db.complaints.aggregate( [
  {$group: { _id: { c: '$company',  st: '$state'}, count: {$sum:1}}},
  {$group: { _id: '', std: {$stdDevPop: '$count'}, avg: {$avg: "$count"}}}
])
```

- There is no indication that **query isolation** will be assured
  - can only be guaranteed on cases where **company** field is used.

Apart from all of this points we will be using this shard key for several other operations.

Exercise: Zone Sharding

We want to isolate subsets of data into a particular shard.

Let's create a zone shard[28] that assigns all data from the company **Bank of America** to one particular shard, **shard002**, and all other data on the remaining shard.

Note: We want students to understand the benefits of zone sharding and also it's limitations.

- Ask the students to create the necessary shard ranges and tag the shards accordingly
- Ask the students if it's possible to isolate all data of a particuar company, in this case **"Bank Of America"** into a single shard.
  - Ask if we could do the same by **zip_code**: Answer should be *no*
    - **zip_code** does not make part of the shardkey!
- Ask students if the order of the shard key would affect their ability to isolate subsets of data
  - It will, because the order of the shard key definition will inevitabily affect the shard ranges and therefore the chunk distribution
- Ranges to be created

[28] https://docs.mongodb.com/manual/release-notes/3.3-dev-series/#sharded-cluster

```
sh.addTagRange("consumer.complaints",
  {company: MinKey, state: MinKey },
  {company: 'Bank of America', state: MinKey },
  "BeforeBankOfAmerica")

sh.addTagRange("consumer.complaints",
  {company: 'Bank of America', state: MinKey },
  {company: 'Bank of America_', state: MinKey },
  "BankOfAmerica")

sh.addTagRange("consumer.complaints",
  {company: "Bank of America_", state: MinKey },
  {company: MaxKey, state: MaxKey },
  "AfterBankOfAmerica")
```

- Then apply these to the appropriate shards

```
use config
sh.addShardTag("shard001", "BeforeBankOfAmerica")
sh.addShardTag("shard002", "BankOfAmerica")
sh.addShardTag("shard003", "AfterBankOfAmerica")
```

- If students insert the following documents

```
db.complaints.insert({"company":"Bank", "state":"CA"})
db.complaints.insert({"company":"Bank of America", "state":"CA"})
db.complaints.insert({"company":"Bank of Canada", "state":"CA"})
```

- And then run the following queries

```
db.complaints.find({"company":"Bank"}).explain()
db.complaints.find({"company":"Bank of America"}).explain()
db.complaints.find({"company":"Bank of Canada"}).explain()
```

- These commands should respectively hit only one shard: shard001, shard002 and shard003

Exercise: Detect Balancing Issues

To avoid having unbalanced shards we should look for some metrics on the sharded collection:

- Which command should we use to detect possible inbalances?
- What's the procedure to solve unbalanced distribution of data accross shards?

Note: The first reaction of students will be to look for number of chunks per shard. You should also mention that inbalances might be cause by existence of *hot shards* - despite having a balanced distribution of chunks they are not balanced in terms of traffic on the different shards.

- Students should use one of 2 things:
  - Query the **config.chunks** collection

```
use config
db.chunks.aggregate([
  {$match: {"ns" : "consumer.complaints"},
```

```
  {$group: {_id: "$shard", nchunks: {$sum:1}}}
])
```

  - Run `sh.status()` command and look for the shard number of chunks per shard

---

Exercise: Move Primary Shard

All sharding enabled[29] databases will have a primary shard. The primary shard will host/hold all non-sharded collections.

We can check each database primary shard using the `sh.status()` command.

For this exercise we are going to do the following:

- add two more shard nodes
  - three data bearing *mongod* each
  - each mongod a separate host
  - these should be named **shard003** and **shard004**
- move primary[30] shard of **consumer** database to **shard003**

Note: To perform this task students should run the following:

- add new shards - use the Ops Manager UI to add the new shards
- move primary[31]

```
db.adminCommand( { movePrimary : "consumer", to : "shard003" } )
```

- While approaching this topic, make sure you clarify with students the considerations around primary move

---

Exercise: Drain Shard

So aparently our application can survive with only two shards.

Given the elastic nature of MongoDB we can change the sharding configuration and consequent server footprint.

Go ahead and remove one of the shards from your sharded cluster.

The procedure should be: - make sure we have ready backup - remove it from the cluster

---

Note: In this exercise we want students to be acquainted with MongoDB elastic nature where we can remove and add shards with some agility.

To accomplish this task students should do the following:

- generate a backup dump from Ops Manager
- change the shard ranges and tags so we can maintain the pre-defined dataset isolation

[29] https://docs.mongodb.com/manual/reference/method/sh.enableSharding/#sh.enableSharding
[30] https://docs.mongodb.com/manual/reference/command/movePrimary/
[31] https://docs.mongodb.com/manual/reference/command/movePrimary/

```
use config
sh.addShardTag("shard0004", "BeforeBankOfAmerica")
sh.addShardTag("shard0004", "AfterBankOfAmerica")
sh.addShardTag("shard0003", "BankOfAmerica")
```

- remove shard - run this operation through Ops Manager UI

1.8 Lab: Analyzing Profiler Data

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.*"

Note: Ask students to students to set up a replica set with the following config: - **Replica Set Name** : AFK - **node10** : priority 10 - **node9** : priority 9 - **node11** : hidden

---

Setup

1. 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.aggcol.insert( { count : i } ); }
```

3. Add some queries.

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

Exercise

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.

Results

Your aggregation query should return documents of the following form:

```
{
  "_id" : "update",
  "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
```

Note: Solution:

```
// 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"}
}});
```

1.9 Lab: Cloud Manager Point-in-Time Backup

Exercise: Point-in-Time Backup

Premise: "*Suppose someone introduced an incorrect code path that randomly drops the database from our production environment.*"

Your data is backed up in Ops Manager, 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 `injector.data` and the total number of documents, regardless of the drop, should be 20,000.

Note: In the `/shared/` folder students can find a script called `inject20k`. This is an executable that will insert 20K documents and will drop the datbase at a random point in time.

Students will have to: - Determine the which data was lost - Request a PITR from Ops Manager - Reinsert any data post drop

Ask the students to set up a 3 node replica set with the following config:

- **Replica Set Name** : DROP_OUT
- **node1** : Priority 10
- **node2** : Priority 3
- **node3** : Priority 0

The replica set should be set up using the Ops Manager UI.