MongoDB Advanced Administrator Training

MongoDB Advanced Administrator Training

Release 3.2

MongoDB, Inc.

September 20, 2016

Contents

1 Advanced Administrator

1.1 Lab: Ops Manager Installation

Permise

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.

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.

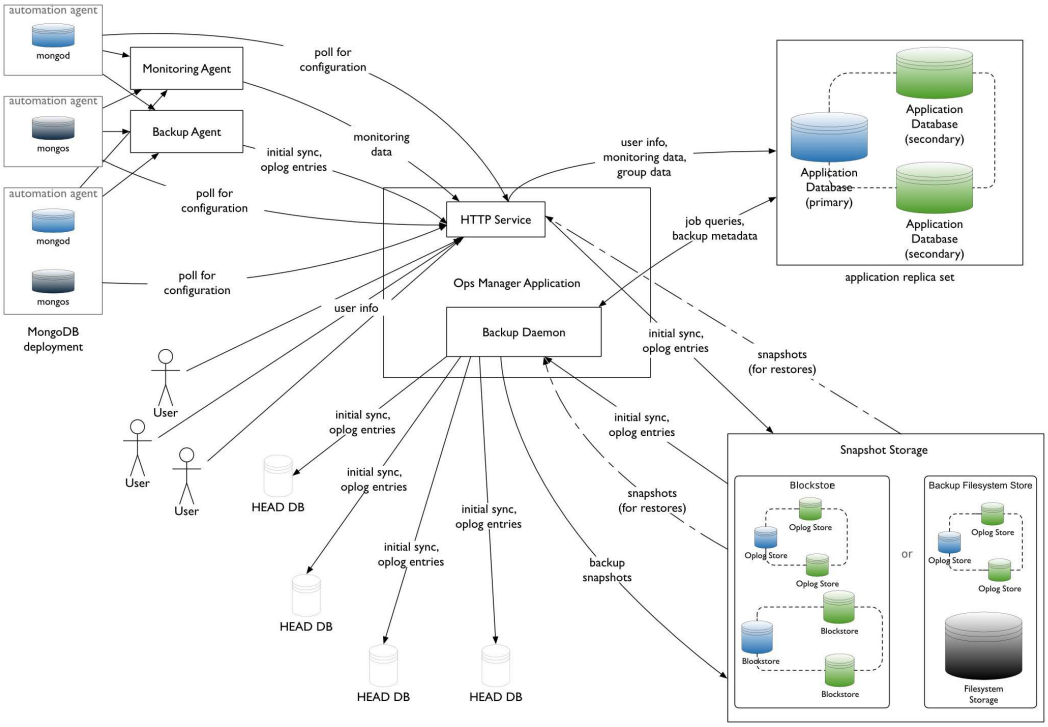
Ops Manager Scalability

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] :



Launch Ops Manager MongoDB Instances

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

We will need the following instances:

- Two Replica Set Clusters
  - Application Replica Set
  - BlockStore Replica Set
- 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 should be included in your team definition file.

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

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*.

2. Configure and Launch OpsManager Service

Habemus Replica Set!

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

- Edit Ops Manager configuration `conf-mms.properties`:
  - Point these to the previously configured Replica Set: **APPDB**
  - Allow offline binary access[3]
- Launch Ops Manager services

3. Install OpsManager Automation Agents

At this point **Ops Manager** should be up and running. Now it's time to install our Automation Agents[4]:

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

1.2 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

[2] https://docs.mongodb.com/manual/administration/security-checklist/
[3] https://docs.opsmanager.mongodb.com/current/tutorial/configure-local-mode/
[4] 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

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

1.3 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**

Exercise: Create Users

Using the Ops Manager API[5] , create the following users:

- **aediles@localhost.com** :
  - password: "123ABCabc!"
  - role: Owner[6]
- **patrician@localhost.com** :
  - password: "123ABCabc!"
  - role: Monitoring Admin[7]
- **consus@localhost.com** :
  - password: "&o7chac0v3r3d"
  - role: Backup Admin[8]

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[9] role : *automater@localhost.com*
- Second user granted Global User Admin[10] 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.

1.4 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.

[5] https://docs.opsmanager.mongodb.com/current/api/
[6] https://docs.opsmanager.mongodb.com/current/reference/user-roles/#owner
[7] https://docs.opsmanager.mongodb.com/current/reference/user-roles/#monitoring-admin
[8] https://docs.opsmanager.mongodb.com/current/reference/user-roles/#backup-admin
[9] https://docs.opsmanager.mongodb.com/current/reference/user-roles/#global-automation-admin
[10] https://docs.opsmanager.mongodb.com/current/reference/user-roles/#global-user-admin

X.509 Authentication Mechanism

We will be using X.509 certificates[11] for authentication and TLS/SSL network encryption.

Ops Manager Group SSL and Auth

To build secured MongoDB deployments you first need to enable Auth and SSL[12] 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

`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!**

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

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*

X509 Users

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

- Go ahead and create a dbAdminAnyDatabase[13] user that authenticates using `dbadmin.pem` certificate.
- To create users that authenticate using X509 certificates you should check Certificate Subject as user[14] docs
- After the user has been created, connect to the *Primary* node of the replica set and create database "`allgood`".

1.5 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.

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

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 poiting 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

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.

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!

Lab: Shard Cluster **(page 10)** Deploy sharded cluster

Lab: Analyzing Profiler Data **(page 13)** Cluster performance analysis using profiler and monitoring dashboards

Lab: Cloud Manager Point-in-Time Backup **(page 14)** Perform point-in-time backup restores using Ops Manager backup

1.6 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[15] should be running on a single host
    - These should be placed on host *Node10*

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

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[16] 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[17] into separate instances
  - They should be placed on *Node9*, *Node8* and *Node7*

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.

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[18] to 1MB (the smallest allowed)
- set the primary shard to max shard size[19] of 500MB (512)

Exercise: Configure Shard (continued)

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

- import/restore this dataset into *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 ?

---

[16] https://docs.mongodb.com/manual/core/sharded-cluster-config-servers/
[17] https://docs.mongodb.com/manual/core/sharded-cluster-config-servers/
[18] https://docs.mongodb.com/manual/tutorial/modify-chunk-size-in-sharded-cluster/
[19] https://docs.mongodb.com/manual/tutorial/manage-sharded-cluster-balancer/#sharded-cluster-config-max-shard-size

Exercise: Zone Sharding

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

Let's create a zone shard[20] that assigns all data from the company **Bank of America** to one particular shard, **shard002**, and all other data on the remaining shard.

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?

Exercise: Move Primary Shard

All sharding enabled[21] 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[22] shard of `consumer` database to **shard003**

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

[20] https://docs.mongodb.com/manual/release-notes/3.3-dev-series/#sharded-cluster
[21] https://docs.mongodb.com/manual/reference/method/sh.enableSharding/#sh.enableSharding
[22] https://docs.mongodb.com/manual/reference/command/movePrimary/

1.7 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.*"

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
```

1.8 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.

Find out more
mongodb.com | mongodb.org
university.mongodb.com

Having trouble?
File a JIRA ticket:
jira.mongodb.org

Follow us on twitter
@MongoDBInc
@MongoDB