Technical Report

MongoDB on the NetApp Data Fabric

Reference Architecture with AFF, ONTAP Cloud, and VMware vSphere

Karthikeyan Nagalingam and Rodrigo Nascimento, NetApp
February 2017 | TR-4492

In partnership with

mongoDB®

Abstract

This reference architecture showcases an end-to-end solution that efficiently deploys and protects virtualized MongoDB on the NetApp® Data Fabric. The solution uses NetApp storage technology and VMware vSphere. The scale-out NetApp AFF array hosts the MongoDB virtual machines and database. This architecture provides very high performance with consistent low latency and excellent inline storage efficiency. The solution leverages NetApp Snap Creator® backup software to achieve instant, space-efficient copies of the MongoDB environment for use in testing, development, QA, backup, and recovery. This reference architecture also showcases remote replication to the NetApp ONTAP® Cloud storage operating system and NetApp Private Storage in Amazon Web Services for remote backups and disaster recovery.

NetApp®

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 Introduction

MongoDB is a very popular open-source scale-out NoSQL database. It powers modern big data analytics applications that require low latency for reads and writes, high availability, and advanced data management. Key use cases for MongoDB include real-time analytics, product catalogs, content management, and mobile applications.

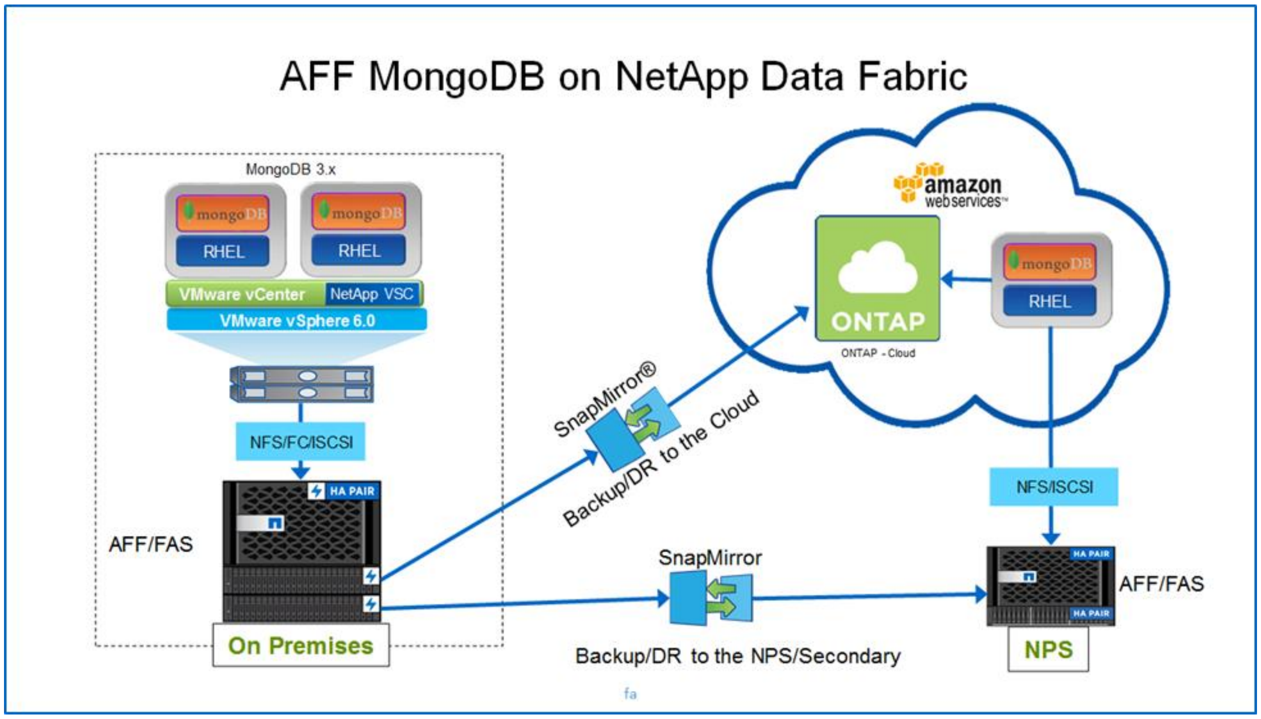
This reference architecture showcases a validated end-to-end solution design for efficiently deploying a virtualized scale-out MongoDB NoSQL database on the NetApp Data Fabric and VMware vSphere. In this solution, the scale-out NetApp All Flash FAS (AFF) array hosts MongoDB virtual machines (VMs) and databases. Backup and disaster recovery services are provided by a NetApp ONTAP Cloud software-defined storage solution in Amazon AWS as well as NetApp Private Storage. The NetApp solution offers the following key benefits that enable deployment of MongoDB mission-critical applications:

- Predictable high performance with consistent low latency as compared to direct-attached storage, providing excellent response time for the most demanding analytics applications built on MongoDB
- Inline efficiency, achieved by inline deduplication and compression to help minimize the flash storage required for MongoDB
- Instant space-efficient and cost-efficient database cloning for rapid setup of dev/test or QA environments without the need to buy new storage
- Backup and recovery based on space-efficient NetApp Snapshot® technology and remote replication to the cloud
- Scalability provided by the scale-up and scale-out AFF array, which can modularly scale storage with MongoDB
- Nondisruptive operations with high availability to deliver maximum uptime and consistent performance, specifically during drive failure
- Deployment of MongoDB in a heterogeneous environment, providing flexibility and additional cost savings
- Single-interface management using Snap Creator with the MongoDB plug-in

2 Solution Overview

In the end-to-end solution shown in Figure 1, virtualized MongoDB is hosted on NetApp A300 storage and VMware vSphere 6. The NetApp AFF array provides low latency and inline deduplication and compression to deliver high performance and reduce flash storage requirements. NetApp Snap Creator backup software provides the framework to invoke instant NetApp Snapshot copies and clones for making zero-cost copies of the entire environment and for use in backup and recovery. Snap Creator also provides the capability to manage remote replication of the entire environment to NetApp ONTAP Cloud instances that run in Amazon Web Services (AWS). The data that is replicated in AWS can be used for disaster recovery and in dev/test environments.

Figure 1) End-to-end solution for MongoDB on the NetApp Data Fabric.



2.1 NetApp Data Fabric

The Data Fabric is NetApp's vision for the future of data management. A Data Fabric seamlessly connects different data management environments across disparate clouds into a cohesive, integrated whole. The NetApp Data Fabric helps organizations maintain control and choose the way they manage, secure, protect, and access their data across the hybrid cloud, no matter where it is.

Although a Data Fabric evolves constantly, organizations can start taking advantage of it today by using NetApp technologies that enable data management and seamless data movement across the hybrid cloud. For more information about the Data Fabric powered by NetApp, see WP-7218: NetApp Data Fabric Architecture Fundamentals: Building a Data Fabric Today.

2.2 NetApp All Flash FAS

NetApp All Flash FAS (AFF) is an enterprise-grade all-flash array that offers powerful benefits:

- High performance at scale
- Inline deduplication and compression
- Modular scale-out
- Best-in-class data management
- Deep application integration

These capabilities, coupled with MongoDB, can help you build a highly scalable, high-performing, and cost-efficient analytics and scale-out database solution.

The key benefits of AFF also include the capability to:

- Accelerate databases with 4 to 12 times higher IOPS and 20 times faster response, powered by NetApp Data ONTAP® FlashEssentials
- Reduce SSD storage fivefold to tenfold on average by using data reduction technologies

- Scale out to 24 storage nodes in a cluster and move data nondisruptively between flash and hard disk drive tiers
- Safeguard your data with an integrated data protection suite that is included in the system and that has a starting cost as low as $25,000
- Set up your system easily

For more information about AFF, see NetApp AFF.

2.3 ONTAP Cloud

ONTAP Cloud for Amazon Web Services is a software-only storage appliance in AWS that is built on ONTAP. ONTAP Cloud delivers leading universal storage management platforms, from on-premises data centers to the cloud. It includes multiple storage-consumption models, providing the flexibility that allows you to truly use just what you need, when you need it. Rapid point-and-click deployment from NetApp OnCommand® Cloud Manager allows you to deploy enterprise cloud storage on AWS in minutes. For more information about ONTAP Cloud for AWS, see NetApp ONTAP Cloud for Amazon Web Services.

2.4 NetApp Private Storage

NetApp Private Storage offers a family of storage solutions to help customers optimize their data use through public cloud providers. For workloads that require high performance and capacity, or for when you want to maintain data control while using the public cloud, NPS is the ideal choice.

2.5 OnCommand Cloud Manager

OnCommand Cloud Manager provides storage management for your hybrid cloud environment. It simplifies the installation and resource assignment of all your cloud storage instances and is the deployment environment for ONTAP Cloud. Cloud Manager also eases the day-to-day requirements of your ONTAP Cloud and NPS for public cloud environment, including configuring, provisioning, and monitoring your active virtual and hardware storage nodes.

Cloud Manager key features include:

- Simplifying configuration and deployment of ONTAP Cloud
- Offering a central point of control for all ONTAP Cloud instances
- Facilitating hybrid environments that include ONTAP Cloud and NetApp Private Storage

2.6 Snap Creator Framework

Snap Creator uses NetApp Snapshot technology to provide application-consistent data protection for both physical and virtualized environments. It also operates with infrastructure-as-a-service cloud environments. It provides a centralized solution for backing up critical information, and it integrates with existing application architectures to support data consistency and reduce operating costs. It enables you to invoke remote replication for use in disaster recovery. For virtualized MongoDB deployments, Snap Creator provides the capability for instant backup and recovery of MongoDB VMs and databases. It also enables you to create rapid clones of the virtualized MongoDB environment for use in dev/test and QA environments. For more information about Snap Creator, see Snap Creator Framework.

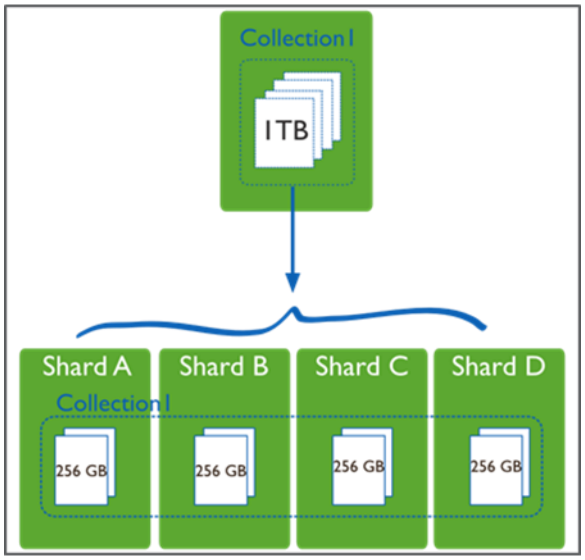
3 Solution Design

This section describes the MongoDB architecture, including its components and layout. It also describes the validated architecture and the storage architecture.

3.1 MongoDB Architecture

Sharding, or horizontal scaling, divides the dataset and distributes the data over multiple servers, or shards. Each shard is an independent database. Collectively, the shards make up a single logical database. Figure 2 shows how shards in a collection can be spread across multiple databases.

Figure 2) Shards spread across multiple servers.

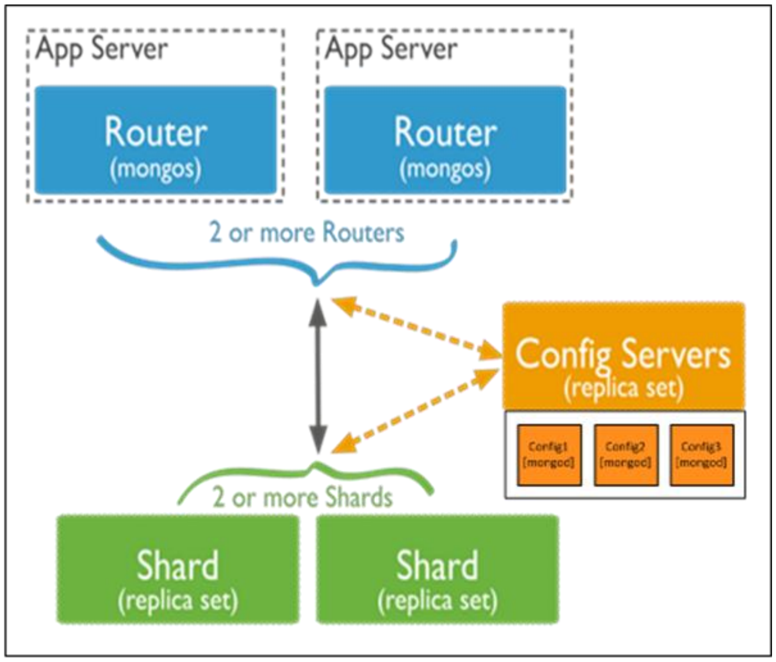


Shards store data. In a production sharded cluster, each shard is a replica set. This strategy provides high availability and data consistency.

Query routers, or mongos instances, interface with client applications and direct operations to the appropriate shard or shards. A client sends requests to a mongos, which then routes the operations to the shards and returns the results to the clients. A sharded cluster can contain more than one mongos to divide the client request load. For this reason, most sharded clusters have more than one mongos.

Configuration servers (config servers) store the cluster's metadata, which contains a mapping of the cluster's dataset to the shards. The query router uses this metadata to target operations to specific shards, as Figure 3 shows.

Figure 3) Query routers (mongos instances) using metadata to route data to shards.



Note: Starting in MongoDB 3.2, config servers for sharded clusters can be deployed as a replica set. The replica set config servers must run the WiredTiger storage engine. MongoDB 3.2 deprecates the use of three mirrored mongod instances for config servers.

Validated Architecture

This solution uses a MongoDB sharding cluster that is virtualized on VMware vSphere 6. In keeping with the MongoDB recommendation for a production setup, during testing we deployed a sharding cluster consisting of two shards. Each shard held a subset of a collection's data. In this solution, each shard is a replica set. Three MongoDB config servers and two query router (mongos) instances are used for the sharding cluster, and the MongoDB setup is configured with the WiredTiger storage engine. The WiredTiger storage engine is responsible for managing how data is stored, both in memory and on disk, and for creating the data files in the `--dbpath` or `storage.dbPath`. For more information, see WiredTiger Storage Engine and Production Cluster Architecture.

Storage Architecture

NFS datastores are NetApp volumes that are accessed through the NFS protocol. Six of these NFS datastores are provisioned on the NetApp A300 all-flash array. The NetApp Virtual Storage Console (VSC) plug-in in VMware vCenter is used to host the various components of the architecture. Table 1 lists these components.

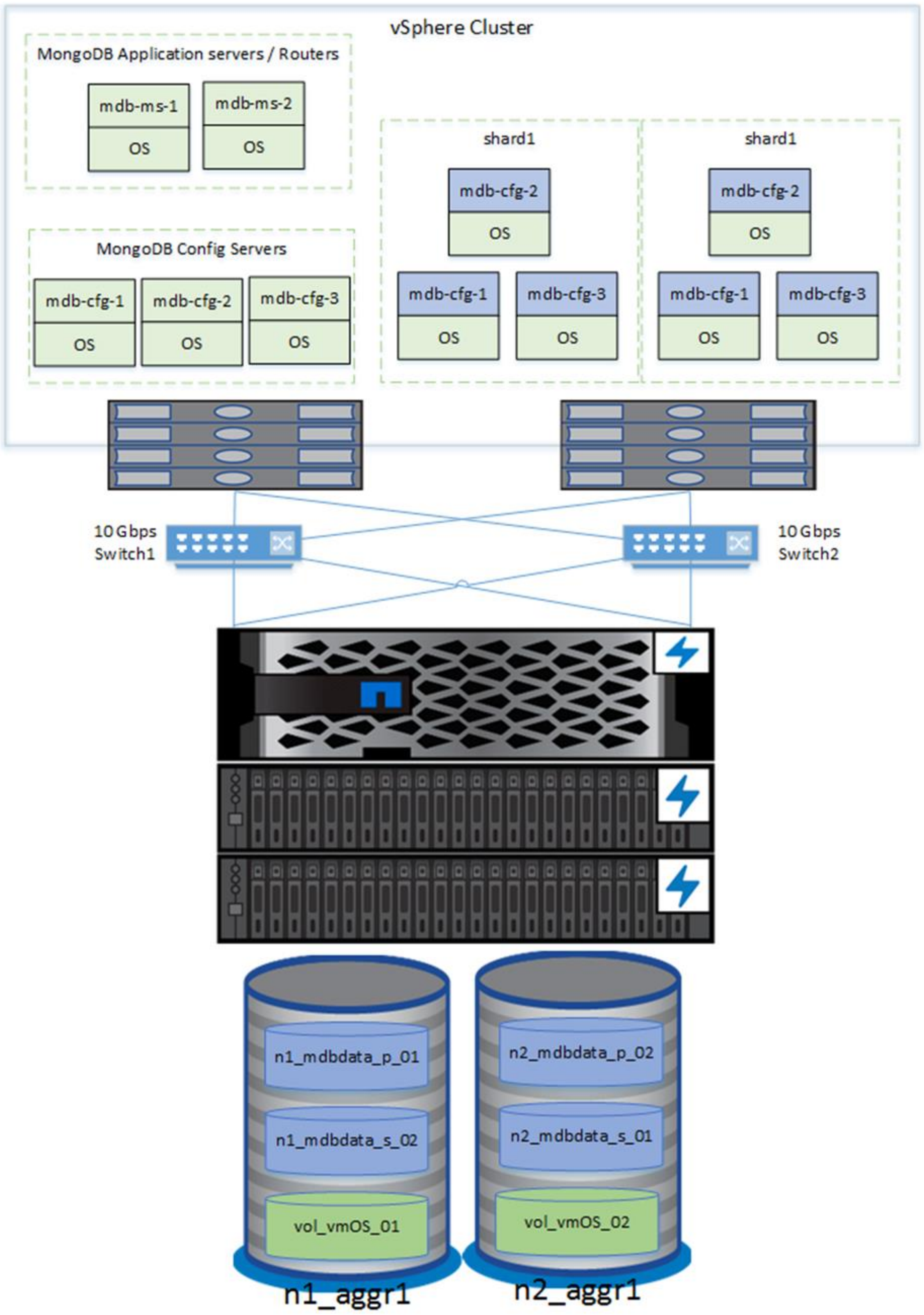
Table 1) Components of the solution architecture.

Serial Number	NetApp Volume	MongoDB Role	Number of Servers with MongoDB Role	Six NetApp Volumes for VMware Datastores (NFS)	VMware Provision
1	Operating systems images volume (example:	Query/router servers	2	2 volumes shared for all	Eager zero thick (.vmdk)

Serial Number	NetApp Volume	MongoDB Role	Number of Servers with MongoDB Role	Six NetApp Volumes for VMware Datastores (NFS)	VMware Provision
	vol_vmOS_01 and vol_vmOS_02)			operating systems	
2	Operating systems images volume (example: vol_vmOS_01 and vol_vmOS_02)	Config servers	3	2 volumes shared for all operating systems	Eager zero thick (.vmdk)
3	Operating systems images volume (example: vol_vmOS_01 and vol_vmOS_02)	Primary shard servers	2	2 volumes shared for all operating systems	Eager zero thick (.vmdk)
4	Operating systems images volume (example: vol_vmOS_01 and vol_vmOS_02)	Secondary shard servers	4	2 volumes shared for all operating systems	Eager zero thick (.vmdk)
5	Primary databases volume (example: vol_mdbdata_p_01 and vol_mdb_data_p_02)	Primary shard database servers	2	2 volumes shared for all operating systems	Eager zero thick (.vmdk)
6	Secondary databases volume (example: vol_mdbdata_s_01 for first sharding secondary servers and vol_mdbdata_s_02 for second sharding secondary servers)	Secondary shard database servers	4	2 volumes shared for all operating systems	Eager zero thick (.vmdk)

Figure 4 shows the storage layout.

Figure 4) Storage layout on NetApp A300.



4 Solution Validation

This section describes the validation of the end-to-end solution, focusing specifically on the key highlights listed in section 1.

4.1 Building the MongoDB Sharding Cluster

Deployment of the MongoDB sharding cluster involves the following tasks:

1. Check the mounted partitions for the MongoDB database.
2. Check and run the shard cluster members in the MongoDB sharding cluster; create the required folders and provide the required permissions.

   Note: The mongod process starts with default ports (27018 and 28018). Alternatively, you can specify ports.

3. Verify that all config servers in MongoDB folders such as logpath (`/mongodb/data/configdb`) and dbpath (`/data/db`) are created. Verify that they have mongodb user permission (700) and that the mongod process is running in fork mode and on the right port (27019).
4. Start the mongos process in the MongoDB router or application server and verify that the port (27017) is listening.
5. Initiate the replication set by connecting to the MongoDB shard primary server from the MongoDB application server.
6. Check the replication set status from the shard member.
7. Add a shard from the application server. Try to add the same shard from another application server by connecting to a replication set member. You should get a message that the shard already exists.
8. Check the sharding status from the application server.
9. Add another shard into an existing sharding cluster.
10. Check the MongoDB sharding cluster status by running the `sh.status()` command from the MongoDB application server.

Note: For the detailed MongoDB steps, see Appendix B: MongoDB Operations.

4.2 Backup and Restore with NetApp Snap Creator

This solution leverages the NetApp Snap Creator framework to back up, restore, and clone the virtualized MongoDB database environment by using space-efficient NetApp Snapshot technology. In our testing, we were able to achieve NetApp Snapshot backups for the complete environment in less than one minute. Restore of the NetApp volumes that hosted the MongoDB database also took less than one minute.

In this solution, MongoDB uses the WiredTiger storage engine.

For our testing, we configured Snap Creator to carry out the following operations:

1. Stop the balancer.
1. Perform an fsync and lock, if necessary.

   Note: The WiredTiger storage engine does not require you to perform an fsync and to lock every shard for writes. If you use the MMAPv1 storage engine, however, you must perform an fsync and lock every shard for writes.

2. Make NetApp Snapshot copies of all volumes used in the configuration.
3. Unlock the databases for writes, if necessary.

   Note: The WiredTiger storage engine does not require you to unlock the databases for writes. If you use the MMAPv1 storage engine, however, you must unlock the databases for writes.

4. Restart the balancer.

Note: The scripts used in the testing are shown in the appendixes.

Snap Creator Backup

This section describes how Snap Creator was configured during our testing to back up the MongoDB shard cluster database. Snap Creator is a flexible framework that has the flexibility to include MongoDB-specific quiesce and unquiesce scripts before and after the NetApp Snapshot backup is created. The NetApp Snapshot backup is immaterial to the size of the database. Only seconds are required to create the NetApp Snapshot backup.

We followed the steps in this procedure to configure Snap Creator for our testing:

1. Install the Snap Creator server in the Snap Creator VM.
2. Install the Snap Creator agent in the MongoDB router or application server.

   Note: This step is required because the Snap Creator server communicates to the MongoDB database through the Snap Creator agent. The Snap Creator agent communicates with MongoDB as a client. Python scripts are executed on the mongodb database in this same way for quiesce and unquiesce operations.

3. In the Snap Creator UI, create the MongoDB profile and configure the following settings:

   a. Configure the plug-in type as `None`.

   b. Configure the Snap Creator agent on one of the MongoDB application servers.

   c. Select the transport `https` with port 443.

   d. Select the storage virtual machine (SVM, formerly Vserver) IP and provide the SVM user name and password.

   e. Include four MongoDB database volumes (primary and secondary):

   – `n1_mdbdata_p_01`

   – `n1_mdbdata_s_02`

   – `n2_mdbdata_p_02`

   – `n2_mdbdata_s_01`

   f. Provide the Snapshot copy name and indicate the policies:

   – Daily: 30

   – Hourly: 24

   – Weekly: 7

   – Monthly: 12

   Note: The default value is monthly, but you can customize the policies to fit your requirements.

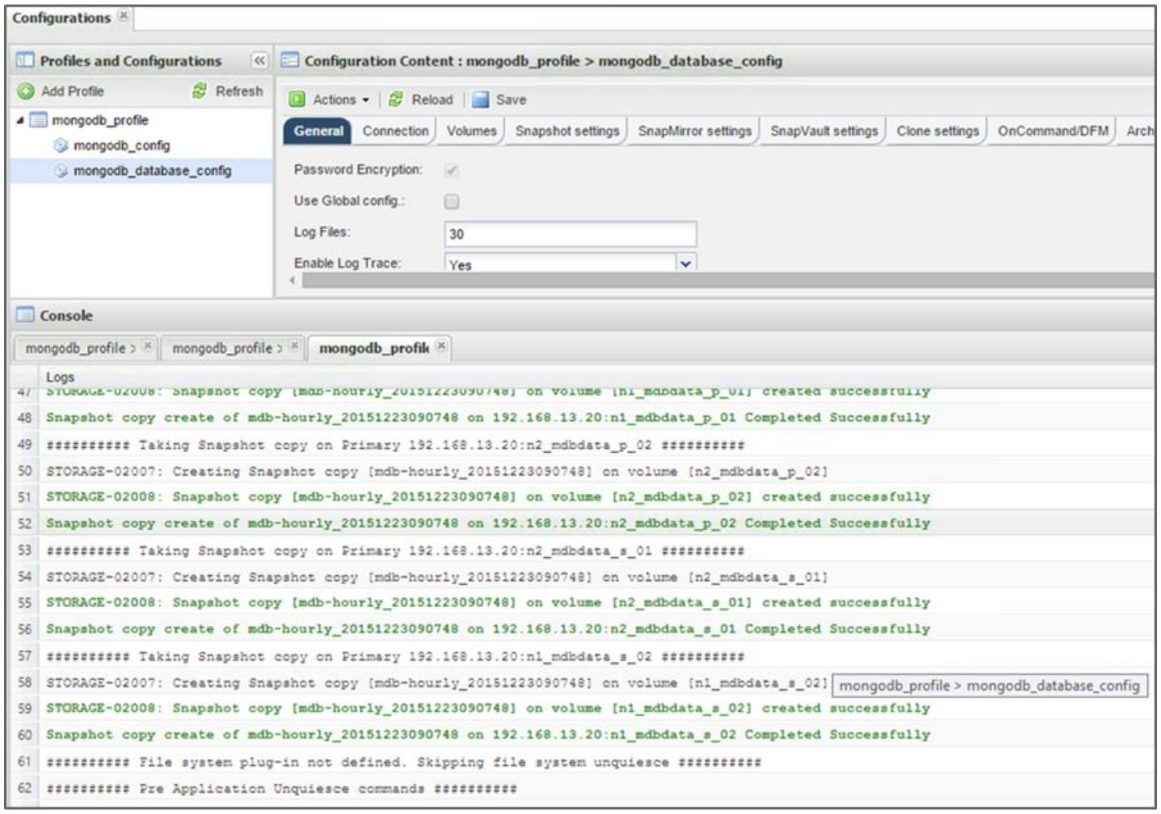
   g. Provide this application quiesce command: `APP_QUIESCE_CMD01=/usr/bin/python2.7 /usr/local/bin/wait-for-cleared-locks.py mdb-ms-1 27017`.

   h. Provide this application unquiesce command: `APP_UNQUIESCE_CMD01=/usr/bin/python2.7 /usr/local/bin/start-balancer.py mdb-ms-1 27017`.

   i. Provide this preexit command: `PRE_EXIT_CMD01=/usr/bin/python2.7 /usr/local/bin/wait-for-cleared-locks.py mdb-ms-1 27017`.

Figure 5 shows the Snap Creator backup operation for the MongoDB database.

Figure 5) Snap Creator backup operation.



Password Encryption: ✓
Use Global config.: ☐
Log Files: 30
Enable Log Trace: Yes

```
STORAGE-02008: Snapshot copy [mdb-hourly_20151223090748] on volume [n1_mdbdata_p_01] created successfully
Snapshot copy create of mdb-hourly_20151223090748 on 192.168.13.20:n1_mdbdata_p_01 Completed Successfully
########## Taking Snapshot copy on Primary 192.168.13.20:n2_mdbdata_p_02 ##########
STORAGE-02007: Creating Snapshot copy [mdb-hourly_20151223090748] on volume [n2_mdbdata_p_02]
STORAGE-02008: Snapshot copy [mdb-hourly_20151223090748] on volume [n2_mdbdata_p_02] created successfully
Snapshot copy create of mdb-hourly_20151223090748 on 192.168.13.20:n2_mdbdata_p_02 Completed Successfully
########## Taking Snapshot copy on Primary 192.168.13.20:n2_mdbdata_s_01 ##########
STORAGE-02007: Creating Snapshot copy [mdb-hourly_20151223090748] on volume [n2_mdbdata_s_01]
STORAGE-02008: Snapshot copy [mdb-hourly_20151223090748] on volume [n2_mdbdata_s_01] created successfully
Snapshot copy create of mdb-hourly_20151223090748 on 192.168.13.20:n2_mdbdata_s_01 Completed Successfully
########## Taking Snapshot copy on Primary 192.168.13.20:n1_mdbdata_s_02 ##########
STORAGE-02007: Creating Snapshot copy [mdb-hourly_20151223090748] on volume [n1_mdbdata_s_02]
STORAGE-02008: Snapshot copy [mdb-hourly_20151223090748] on volume [n1_mdbdata_s_02] created successfully
Snapshot copy create of mdb-hourly_20151223090748 on 192.168.13.20:n1_mdbdata_s_02 Completed Successfully
########## File system plug-in not defined. Skipping file system unquiesce ##########
########## Pre Application Unquiesce commands ##########
```

Snap Center enables you to back up very large MongoDB databases in seconds. Snap Creator has an option to schedule the backup and notify you about the job status.

Snap Creator Restore

As a database's size increases, problems arise in backing it up and restoring it. NetApp Snapshot technology provides the simplest and most efficient way to back up and restore a distributed database of any size. If backups are based on Snapshot copies, the time required to restore a large database composed of multiple shards is related more to the time spent in stopping and starting the database processes than to the act of restoring the data itself. The restore of a Snapshot backup often takes only seconds, or minutes at most.

The following example from our testing shows how to remove the c7 collection, stop all mongodb processes across all the shards, unmount the `/data` file systems, restore the volumes by using Snap Creator, remount the file systems, remove the lock files, and restart all processes:

1. Before restore, check the database and its documents from one of the application server.

Note: The c7 collection has more documents than the number of documents at the time of backup. The following counts are queried before the restore process begins.

```
[mongodb@mdb-ms-1 ~]$  mongo  mdb-ms-1:27017/admin
MongoDB shell version: 3.0.7
connecting to: mdb-ms-1:27017/admin
mongos> show dbs;
admin        (empty)
bulkdb2      0.000GB
config       0.016GB
db1         26.282GB
db2         39.771GB
```

```
db3          44.171GB
db4          34.444GB
db5          34.416GB
db6          34.225GB
db7          71.155GB
newbulkdb     0.000GB
posts         0.000GB
mongos> use db7;
switched to db db7
mongos> show collections;
c7
mongos> db.c7.count();
144569144
mongos> db.c7.drop();
true
mongos> db.c7.count();
0
mongos>
```

2. Stop the first replica set (mongod) process by using the `init` script. Alternatively, you can use the kill command to stop the mongod process first sharding or replica set members (`mdb-srv-1`, `mdb-srv-10`, `mdb-srv-11`) and unmount the database volume.
   - From `mdb-srv-1`:

```
[mongodb@mdb-srv-1 ~]$ pkill mongod
[mongodb@mdb-srv-1 ~]$ netstat -lntp
(No info could be read for "-p": geteuid()=8000 but you should be root.)
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN      -
tcp6       0      0 :::22                   :::*                    LISTEN      -
tcp6       0      0 ::1:631                 :::*                    LISTEN      -
tcp6       0      0 ::1:25                  :::*                    LISTEN      -
[mongodb@mdb-srv-1 ~]$ umount /data
umount: /data: umount failed: Operation not permitted
[mongodb@mdb-srv-1 ~]$ logout
[root@mdb-srv-1 ~]# umount /data
[root@mdb-srv-1 ~]#
```

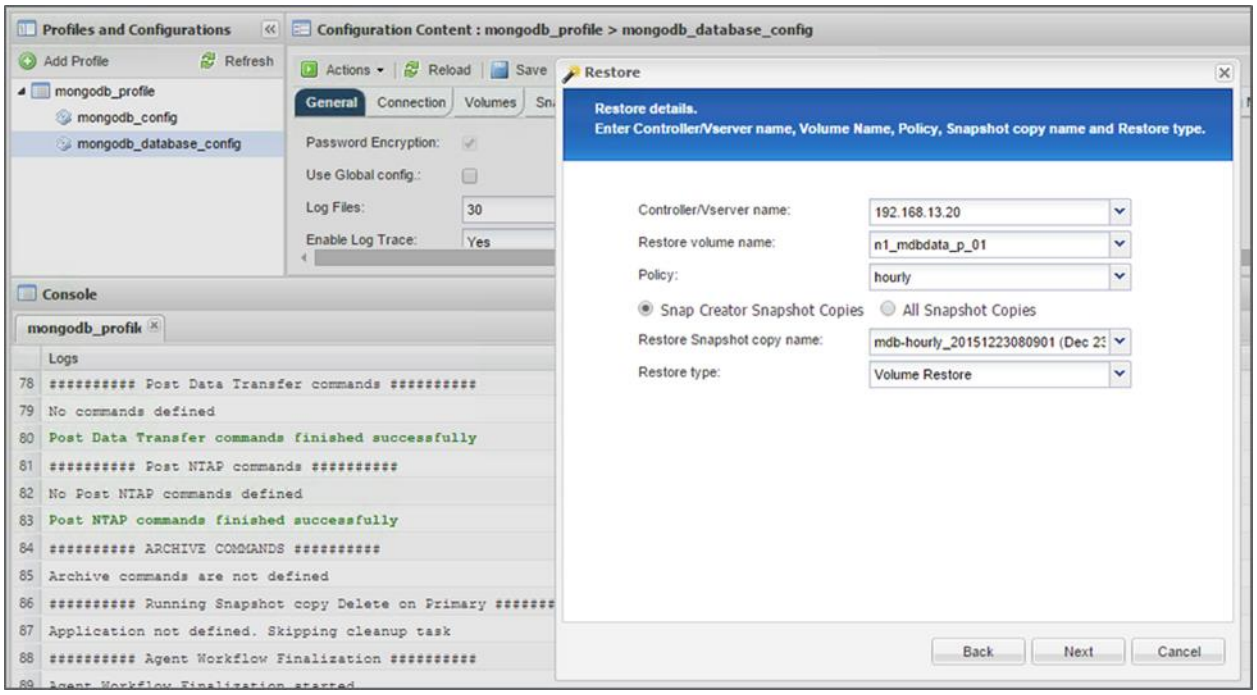
Note: You can kill the mongod process from the `mdb-srv-10` and `mdb-srv-11` servers in the same way.

3. Stop the second replica set (mongod) process by using the `init` script. Alternatively, you can use the kill command to stop the mongod process second sharding or replica set members (`mdb-srv-2`, `mdb-srv-20`, `mdb-srv-21`) and unmount the database volume.
   - From `mdb-srv-2`:

```
[mongodb@mdb-srv-2 ~]$ pkill mongod
[mongodb@mdb-srv-2 ~]$ netstat -lntp
(No info could be read for "-p": geteuid()=8000 but you should be root.)
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN      -
tcp6       0      0 :::22                   :::*                    LISTEN      -
tcp6       0      0 ::1:631                 :::*                    LISTEN      -
tcp6       0      0 ::1:25                  :::*                    LISTEN      -
[mongodb@mdb-srv-2 ~]$ logout
[root@mdb-srv-2 ~]# umount /data
[root@mdb-srv-2 ~]#
```

Note: You can kill the mongod process from the `mdb-srv-20` and `mdb-srv-21` servers in the same way.

4. Delete/remove the existing harddrive (MongoDB database volumes) from the VMs, which are mounted in both shard cluster members, including the primary and secondary servers, for `database/dbpath`.
5. Restore all database volumes by using Snap Creator for each database volume.



Password Encryption: ✓
Use Global config.: ☐
Log Files: 30
Enable Log Trace: Yes

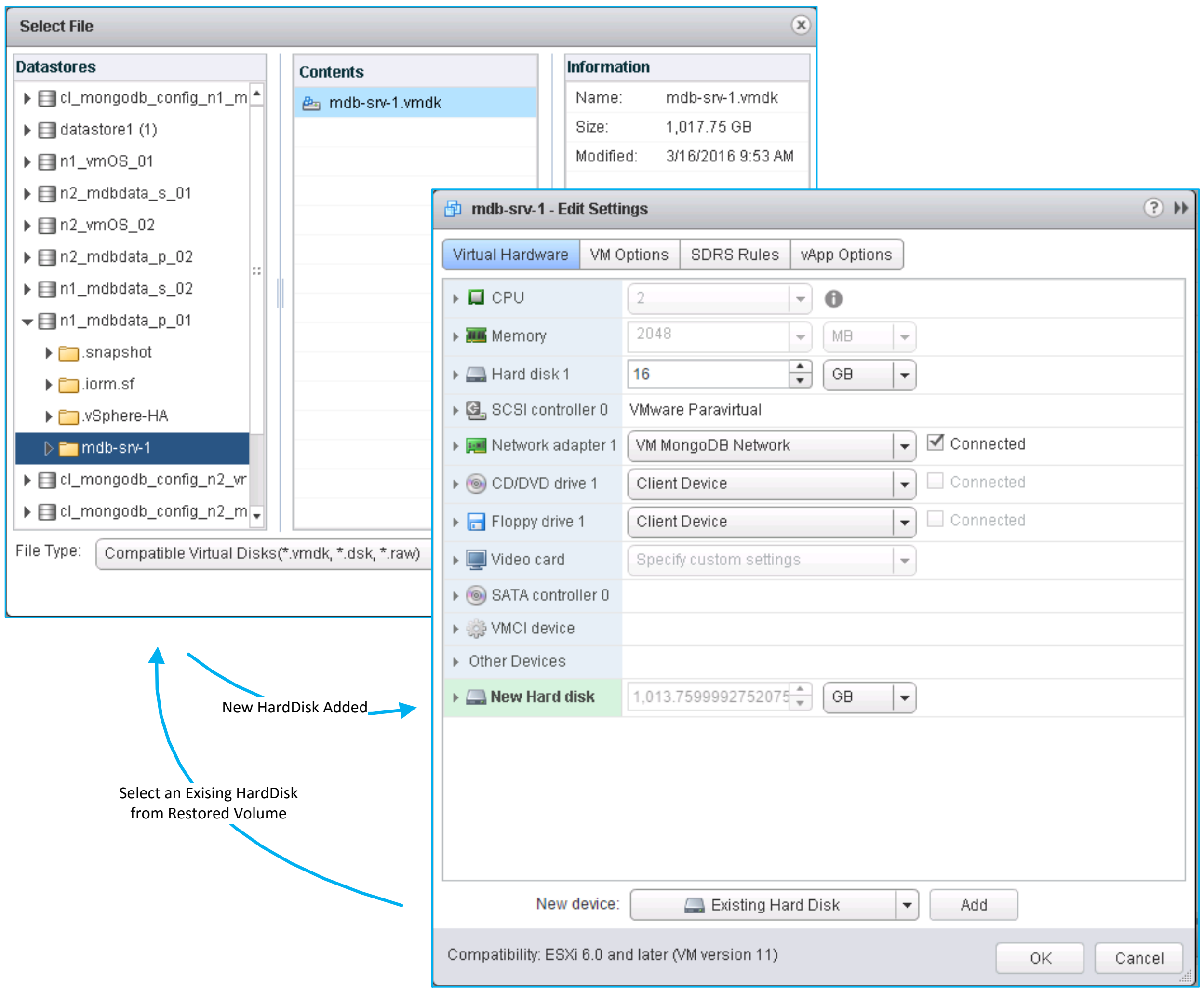
```
########## Post Data Transfer commands ##########
No commands defined
Post Data Transfer commands finished successfully
########## Post NTAP commands ##########
No Post NTAP commands defined
Post NTAP commands finished successfully
########## ARCHIVE COMMANDS ##########
Archive commands are not defined
########## Running Snapshot copy Delete on Primary #######
Application not defined. Skipping cleanup task
########## Agent Workflow Finalization ##########
Agent Workflow Finalization started
```

Restore details.
Enter Controller/Vserver name, Volume Name, Policy, Snapshot copy name and Restore type.

Controller/Vserver name:	192.168.13.20
Restore volume name:	n1_mdbdata_p_01
Policy:	hourly
[✓ Snap Creator Snapshot Copies]	All Snapshot Copies
Restore Snapshot copy name:	mdb-hourly_20151223080901 (Dec 2:
Restore type:	Volume Restore

Note: This screenshot shows only one volume (`n1_mdbdata_p_01`), but volumes `n1_mdbdata_s_02`, `n2_mdbdata_p_02`, and `n2_mdbdata_s_01` must all be restored.

6. Reconnect the restored database volumes as existing drives to the MongoDB sharding or replica set primary and secondary servers.



Contents: mdb-srv-1.vmdk

Name:	mdb-srv-1.vmdk
Size:	1,017.75 GB
Modified:	3/16/2016 9:53 AM

File Type: Compatible Virtual Disks(*.vmdk, *.dsk, *.raw)

CPU	2		
Memory	2048	MB	
Hard disk 1	16	GB	
SCSI controller 0	VMware Paravirtual		
Network adapter 1	VM MongoDB Network		✓ Connected
CD/DVD drive 1	Client Device		Connected
Floppy drive 1	Client Device		Connected
Video card	Specify custom settings		
SATA controller 0			
VMCI device			
Other Devices			
New Hard disk	1,013.7599992752075	GB	

Compatibility: ESXi 6.0 and later (VM version 11)

7. Mount the volumes.
8. Remove the `mongod.lock` file from the mounted MongoDB database volumes.
9. Start the mongod process on the sharding cluster servers.
10. Verify that the mongod ports (27018 and 28018) are listening.

```
[root@mdb-srv-1 ~]# fdisk /dev/sdb
Disk /dev/sdb: 1088.5 GB, 1088516510720 bytes, 2126008810 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk label type: dos
Disk identifier: 0xd35931be

   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1            2048  2126008809  1063003381   83  Linux
[root@mdb-srv-1 ~]# mount /data
[root@mdb-srv-1 ~]# su - mongodb
Last login: Wed Dec 23 09:21:59 EST 2015 on pts/1
[mongodb@mdb-srv-1 ~]$ cd /data/db_wt/
[mongodb@mdb-srv-1 db_wt]$ rm -f mongod.lock
[mongodb@mdb-srv-1 db_wt]$ mongod --replSet shard1/mdb-srv-10,mdb-srv-11 --journal --rest --
shardsvr --fork --storageEngine wiredTiger --dbpath /data/db_wt --logpath
/mongodb/data/mongod.log --directoryperdb
2015-12-23T09:24:46.629-0500 I CONTROL  ** WARNING: --rest is specified without --httpinterface,
2015-12-23T09:24:46.630-0500 I CONTROL  **          enabling http interface
about to fork child process, waiting until server is ready for connections.
```

```
forked process: 32080
child process started successfully, parent exiting
[mongodb@mdb-srv-1 db_wt]$ netstat -lntp
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:27018           0.0.0.0:*               LISTEN      32080/mongod
tcp        0      0 0.0.0.0:28018           0.0.0.0:*               LISTEN      32080/mongod
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN      -
tcp6       0      0 :::22                   :::*                    LISTEN      -
tcp6       0      0 ::1:631                 :::*                    LISTEN      -
tcp6       0      0 ::1:25                  :::*                    LISTEN      -
[mongodb@mdb-srv-1 db_wt]$
```

Note: Repeat the same procedure on `mdb-srv-10`, `mdb-srv-11`, `mdb-srv-2`, `mdb-srv-20`, and `mdb-srv-srv-21` servers. Mount the restored volume, remove the `mongod.lock` file, and start the mongod process for the replication set.

11. Check the sharding status in the restored database and the document values.

```
[mongodb@mdb-ms-1 ~]$  mongo  mdb-ms-1:27017/admin
MongoDB shell version: 3.0.7
connecting to: mdb-ms-1:27017/admin
mongos> sh.status();
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "minCompatibleVersion" : 5,
        "currentVersion" : 6,
        "clusterId" : ObjectId("565315e55e022cd500e768d5")
}
  shards:
        {  "_id" : "shard1",  "host" : "shard1/mdb-srv-1:27018,mdb-srv-10:27018,mdb-srv-11:27018"
}
        {  "_id" : "shard2",  "host" : "shard2/mdb-srv-2:27018,mdb-srv-20:27018,mdb-srv-21:27018"
}
  balancer:
        Currently enabled:  yes
        Currently running:  no
        Failed balancer rounds in last 5 attempts:  0
        Migration Results for the last 24 hours:
                No recent migrations
  databases:
        {  "_id" : "admin",  "partitioned" : false,  "primary" : "config" }
        {  "_id" : "newbgdb",  "partitioned" : true,  "primary" : "shard1" }
        {  "_id" : "test",  "partitioned" : true,  "primary" : "shard2" }
        {  "_id" : "newdb",  "partitioned" : true,  "primary" : "shard1" }
        {  "_id" : "ndb",  "partitioned" : true,  "primary" : "shard2" }
        {  "_id" : "mdb",  "partitioned" : true,  "primary" : "shard1" }
        {  "_id" : "ldb",  "partitioned" : true,  "primary" : "shard2" }
        {  "_id" : "newbulkdb",  "partitioned" : true,  "primary" : "shard1" }
        {  "_id" : "bulkdb2",  "partitioned" : true,  "primary" : "shard2" }
        {  "_id" : "posts",  "partitioned" : true,  "primary" : "shard1" }
        {  "_id" : "db1",  "partitioned" : true,  "primary" : "shard2" }
        {  "_id" : "db2",  "partitioned" : true,  "primary" : "shard1" }
        {  "_id" : "db3",  "partitioned" : true,  "primary" : "shard1" }
        {  "_id" : "db4",  "partitioned" : true,  "primary" : "shard2" }
        {  "_id" : "db5",  "partitioned" : false,  "primary" : "shard2" }
        {  "_id" : "db6",  "partitioned" : true,  "primary" : "shard2" }
        {  "_id" : "db7",  "partitioned" : true,  "primary" : "shard1" }
        {  "_id" : "netbgdb",  "partitioned" : true,  "primary" : "shard2" }

mongos> show dbs;
admin        (empty)
bulkdb2      0.000GB
config       0.016GB
db1         26.282GB
db2         39.771GB
```

```
db3         44.171GB
db4         34.894GB
db5         34.869GB
db6         34.674GB
db7         71.155GB
db8        849.595GB
newbulkdb    0.000GB
posts        0.000GB
mongos> use admin
switched to db admin
mongos> db.runCommand({enableSharding: "db5"})
{ "ok" : 1 }
mongos> sh.status();
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "minCompatibleVersion" : 5,
        "currentVersion" : 6,
        "clusterId" : ObjectId("565315e55e022cd500e768d5")
}
  shards:
        {  "_id" : "shard1",  "host" : "shard1/mdb-srv-1:27018,mdb-srv-10:27018,mdb-srv-11:27018"
}
        {  "_id" : "shard2",  "host" : "shard2/mdb-srv-2:27018,mdb-srv-20:27018,mdb-srv-21:27018"
}
  balancer:
        Currently enabled:  yes
        Currently running:  no
        Failed balancer rounds in last 5 attempts:  0
        Migration Results for the last 24 hours:
                No recent migrations
  databases:
        {  "_id" : "admin",  "partitioned" : false,  "primary" : "config" }
        {  "_id" : "newbgdb",  "partitioned" : true,  "primary" : "shard1" }
        {  "_id" : "test",  "partitioned" : true,  "primary" : "shard2" }
        {  "_id" : "newdb",  "partitioned" : true,  "primary" : "shard1" }
        {  "_id" : "ndb",  "partitioned" : true,  "primary" : "shard2" }
        {  "_id" : "mdb",  "partitioned" : true,  "primary" : "shard1" }
        {  "_id" : "ldb",  "partitioned" : true,  "primary" : "shard2" }
        {  "_id" : "newbulkdb",  "partitioned" : true,  "primary" : "shard1" }
        {  "_id" : "bulkdb2",  "partitioned" : true,  "primary" : "shard2" }
        {  "_id" : "posts",  "partitioned" : true,  "primary" : "shard1" }
        {  "_id" : "db1",  "partitioned" : true,  "primary" : "shard2" }
        {  "_id" : "db2",  "partitioned" : true,  "primary" : "shard1" }
        {  "_id" : "db3",  "partitioned" : true,  "primary" : "shard1" }
        {  "_id" : "db4",  "partitioned" : true,  "primary" : "shard2" }
        {  "_id" : "db5",  "partitioned" : true,  "primary" : "shard2" }
        {  "_id" : "db6",  "partitioned" : true,  "primary" : "shard2" }
        {  "_id" : "db7",  "partitioned" : true,  "primary" : "shard1" }
        {  "_id" : "netbgdb",  "partitioned" : true,  "primary" : "shard2" }

mongos>
```

12. Check the db7 database and find the number of documents in the c7 collection.

```
[mongodb@mdb-ms-1 ~]$  mongo  mdb-ms-1:27017/admin
MongoDB shell version: 3.0.7
connecting to: mdb-ms-1:27017/admin
mongos> show dbs;
admin        (empty)
bulkdb2      0.000GB
config       0.016GB
db1         26.282GB
db2         39.771GB
db3         44.171GB
db4         34.723GB
db5         34.698GB
db6         34.505GB
db7         71.155GB
db8        849.595GB
```

```
newbulkdb    0.000GB
posts        0.000GB
mongos> use db7
switched to db db7
mongos> show collections;
c7
mongos> db.c7.count();
144569144
mongos>
```

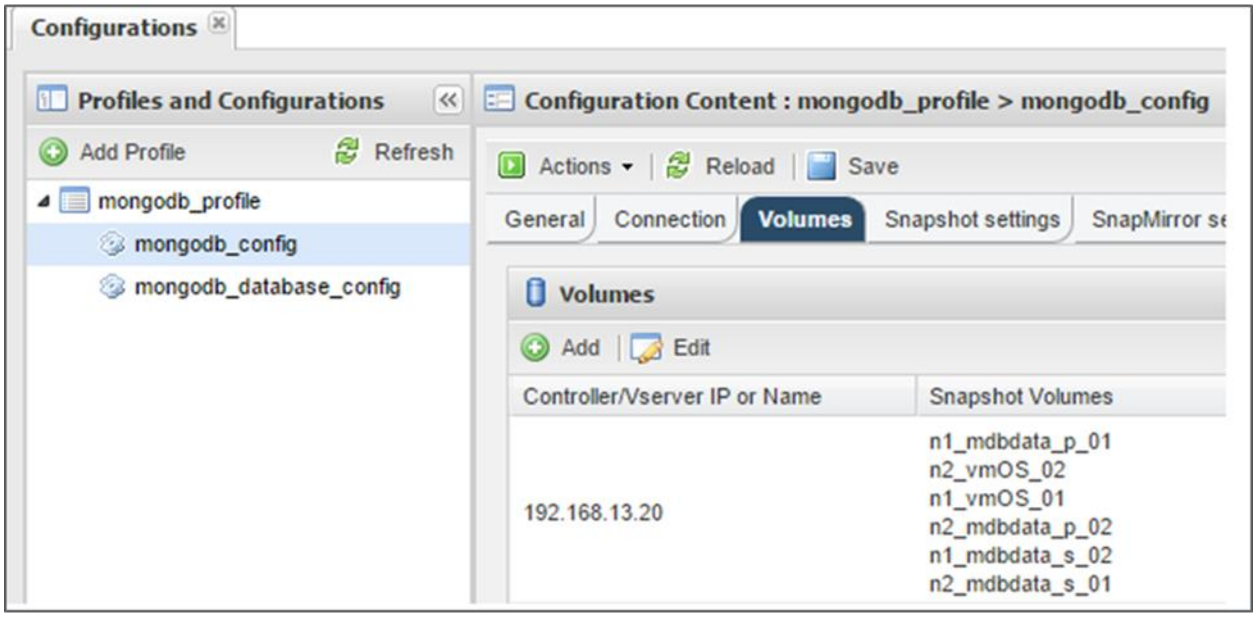
This example demonstrates that a MongoDB database can be restored by using Snap Creator and leveraging NetApp Snapshot copies and NetApp SnapRestore® technology. The c7 collection is displayed with the number of documents from the db7 database.

4.3 MongoDB Cloning for Dev/Test and QA with Snap Creator

Snap Creator enables you to build preproduction or development environments from existing production NetApp Snapshot backups. By using FlexClone® flexible clones, you can present volumes that have Snapshot copies to any clone host. This technology enables you to build dev/test environments quickly without having to double the amount of storage required. FlexClone clones are a fast, scalable, and efficient way to provide dev/test environments on demand.

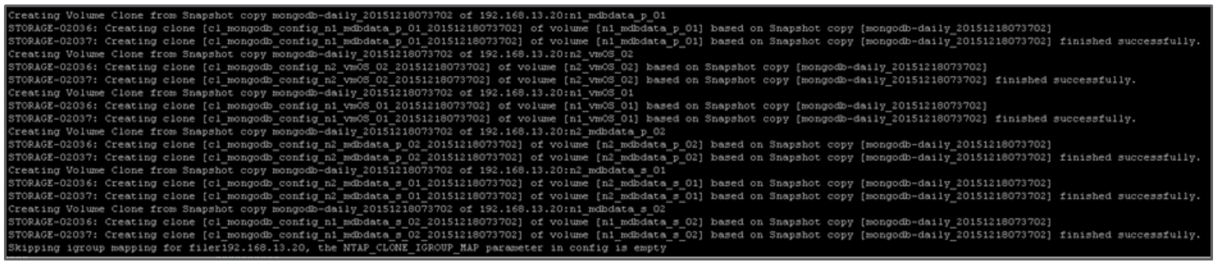
The following example demonstrates how to clone all databases across all shard servers into new shard servers. In this example, the operations system volumes are also cloned.

1. Create the Snap Creator profile for database volumes and for operating system volumes.

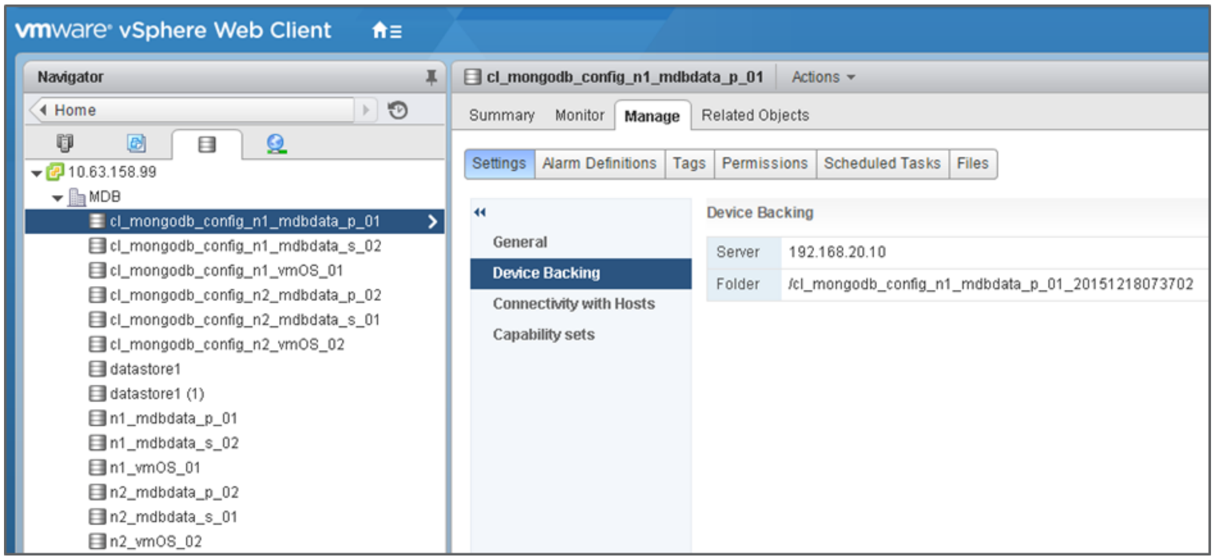


Controller/Vserver IP or Name	Snapshot Volumes
192.168.13.20	n1_mdbdata_p_01 n2_vmOS_02 n1_vmOS_01 n2_mdbdata_p_02 n1_mdbdata_s_02 n2_mdbdata_s_01

2. Using Snap Creator, create FlexClone volumes for operating system and database volumes from a specific Snapshot copy.
3. Create a datastore that includes the cloned operating system and database volumes in the VMware servers.



```
Creating Volume Clone from Snapshot copy mongodb-daily_20151218073702 of 192.168.13.20:n1_mdbdata_p_01
STORAGE-02036: Creating clone [cl_mongodb_config_n1_mdbdata_p_01_20151218073702] of volume [n1_mdbdata_p_01] based on Snapshot copy [mongodb-daily_20151218073702]
STORAGE-02037: Creating clone [cl_mongodb_config_n1_mdbdata_p_01_20151218073702] of volume [n1_mdbdata_p_01] based on Snapshot copy [mongodb-daily_20151218073702] finished successfully.
Creating Volume Clone from Snapshot copy mongodb-daily_20151218073702 of 192.168.13.20:n2_vmOS_02
STORAGE-02036: Creating clone [cl_mongodb_config_n2_vmOS_02_20151218073702] of volume [n2_vmOS_02] based on Snapshot copy [mongodb-daily_20151218073702]
STORAGE-02037: Creating clone [cl_mongodb_config_n2_vmOS_02_20151218073702] of volume [n2_vmOS_02] based on Snapshot copy [mongodb-daily_20151218073702] finished successfully.
Creating Volume Clone from Snapshot copy mongodb-daily_20151218073702 of 192.168.13.20:n1_vmOS_01
STORAGE-02036: Creating clone [cl_mongodb_config_n1_vmOS_01_20151218073702] of volume [n1_vmOS_01] based on Snapshot copy [mongodb-daily_20151218073702]
STORAGE-02037: Creating clone [cl_mongodb_config_n1_vmOS_01_20151218073702] of volume [n1_vmOS_01] based on Snapshot copy [mongodb-daily_20151218073702] finished successfully.
Creating Volume Clone from Snapshot copy mongodb-daily_20151218073702 of 192.168.13.20:n2_mdbdata_p_02
STORAGE-02036: Creating clone [cl_mongodb_config_n2_mdbdata_p_02_20151218073702] of volume [n2_mdbdata_p_02] based on Snapshot copy [mongodb-daily_20151218073702]
STORAGE-02037: Creating clone [cl_mongodb_config_n2_mdbdata_p_02_20151218073702] of volume [n2_mdbdata_p_02] based on Snapshot copy [mongodb-daily_20151218073702] finished successfully.
Creating Volume Clone from Snapshot copy mongodb-daily_20151218073702 of 192.168.13.20:n2_mdbdata_s_01
STORAGE-02036: Creating clone [cl_mongodb_config_n2_mdbdata_s_01_20151218073702] of volume [n2_mdbdata_s_01] based on Snapshot copy [mongodb-daily_20151218073702]
STORAGE-02037: Creating clone [cl_mongodb_config_n2_mdbdata_s_01_20151218073702] of volume [n2_mdbdata_s_01] based on Snapshot copy [mongodb-daily_20151218073702] finished successfully.
Creating Volume Clone from Snapshot copy mongodb-daily_20151218073702 of 192.168.13.20:n1_mdbdata_s_02
STORAGE-02036: Creating clone [cl_mongodb_config_n1_mdbdata_s_02_20151218073702] of volume [n1_mdbdata_s_02] based on Snapshot copy [mongodb-daily_20151218073702]
STORAGE-02037: Creating clone [cl_mongodb_config_n1_mdbdata_s_02_20151218073702] of volume [n1_mdbdata_s_02] based on Snapshot copy [mongodb-daily_20151218073702] finished successfully.
Skipping igroup mapping for filer192.168.13.20, the NTAP_CLONE_IGROUP_MAP parameter in config is empty
```



Device Backing

Server	192.168.20.10
Folder	/cl_mongodb_config_n1_mdbdata_p_01_20151218073702

4. Create the cloned VM from the datastore based on the `.vmx` file:
   a. Select the datastore created in step 3 above.
   b. Find the `.vmx` file.
   c. Register the VM.
   d. Create the cloned VM name in the following format: `cl_<old VM name>`.
   e. Select the VMware cluster and server to finish the cloned VM creation.
5. The cloned (newly registered) VM accesses the database `.vmdk` file from another database cloned volume; verify the .vmdk file is identified as an existing disk. If it is not, then in the cloned VM, select Edit > Settings > Remove disk > Confirm.

   Note: Make sure that you do not remove the disk from the repository.

   Note: Remove the `.lck<16 alphanumeric characters>` files from the database cloned volume. Mount the cloned volume in the VMware or Linux server and remove the `.lck` files, as shown in the following example:

```
[root@stlrx300s8-9:/vmfs/volumes/24631f70-b7e80cce] df -h
/vmfs/volumes/cl_mongodb_config_n1_mdbdata_p_01
Filesystem   Size   Used Available Use% Mounted on
NFS          1.0T  49.6G    974.4G   5% /vmfs/volumes/cl_mongodb_config_n1_mdbdata_p_01
[root@stlrx300s8-9:/vmfs/volumes/24631f70-b7e80cce]
[ root@stlrx300s8-9:~] cd /vmfs/volumes/cl_mongodb_config_n1_mdbdata_p_01
[root@stlrx300s8-9:/vmfs/volumes/24631f70-b7e80cce] ls
mdb-srv-1
[root@stlrx300s8-9:/vmfs/volumes/24631f70-b7e80cce] cd mdb-srv-1/
[root@stlrx300s8-9:/vmfs/volumes/24631f70-b7e80cce/mdb-srv-1] ls -la
total 1067189496
drwxr-xr-x    2 root     root          4096 Dec 18  2015 .
```

```
drwxr-xr-x    5 root     root          4096 Dec 18  2015 ..
-rwxrwxr-x    1 root     root            84 Dec 18 14:41 .lck-be54694500000000
-rwxrwxr-x    1 root     root            84 Dec 18  2015 .lck-be546a4500000000
-rw-------    1 root     root     1088516510720 Dec 18 14:41 mdb-srv-1-flat.vmdk
-rw-------    1 root     root           475 Nov 22 17:51 mdb-srv-1.vmdk
[root@stlrx300s8-9:/vmfs/volumes/24631f70-b7e80cce/mdb-srv-1] rm -fr .lck-be546
.lck-be54694500000000  .lck-be546a4500000000
[root@stlrx300s8-9:/vmfs/volumes/24631f70-b7e80cce/mdb-srv-1] rm -fr .lck-be546
.lck-be54694500000000  .lck-be546a4500000000
[root@stlrx300s8-9:/vmfs/volumes/24631f70-b7e80cce/mdb-srv-1] rm -fr .lck-be546*
[root@stlrx300s8-9:/vmfs/volumes/24631f70-b7e80cce/mdb-srv-1] ls -la
total 1067189488
drwxr-xr-x    2 root     root          4096 Dec 18  2015 .
drwxr-xr-x    5 root     root          4096 Dec 18  2015 ..
-rw-------    1 root     root     1088516510720 Dec 18 14:41 mdb-srv-1-flat.vmdk
-rw-------    1 root     root           475 Nov 22 17:51 mdb-srv-1.vmdk
[root@stlrx300s8-9:/vmfs/volumes/24631f70-b7e80cce/mdb-srv-1]
```

6. Start (power on) the cloned VMs.
7. Change the IP address of the cloned hosts and update `/etc/hosts`.

   Note: In cloned VMs, NetApp recommends that you use the same host names with different IP addresses.

8. Restart the MongoDB config, application, and replica set or sharding servers with the new IP addresses from the cloned VMs.

This procedure creates a working cloned copy of the production MongoDB database. The cloned copy is isolated and available to use for writes, troubleshooting, development, or test. Because its storage is based on FlexClone technology, no additional storage is used in the process.

4.4 Replication to Cloud ONTAP in AWS for Disaster Recovery

NetApp ONTAP Cloud for AWS is a software-only storage appliance that runs the NetApp clustered Data ONTAP storage operating system in the cloud. ONTAP Cloud manages general-purpose Amazon Elastic Block Storage (GP2 EBS) with ONTAP and provides enterprise-class features on top of EBS. This configuration provides access to NFS, CIFS, and iSCSI protocol support as well as to a rich feature set that enhances the management and efficiency of your storage. You also have access to industry-leading technologies such as NetApp SnapMirror® and NetApp SnapVault® data replication, which enable seamless connectivity for hybrid cloud resources.

ONTAP Cloud is launched and managed through the NetApp OnCommand Cloud Manager application. Cloud Manager is a web front end that enables the deployment and management of AWS public cloud resources associated with ONTAP Cloud. Cloud Manager provides a flexible, intuitive interface for activities such as deployment of ONTAP Cloud working environments, intelligent allocation of additional AWS EBS storage, creation of NetApp flexible volumes, and so on.

Cloud Manager can be deployed several different ways, including:

- Into your local data center from the NetApp Support software downloads site
- Into an existing EC2 instance that runs a supported version of Windows
- From the AWS marketplace from an Amazon machine image into an EC2 instance

For more information, see the OnCommand Cloud Manager 1.0 Installation and Setup Guide and the OnCommand Cloud Manager 1.0 User Guide.

4.5 Disaster Recovery Validation to Cloud

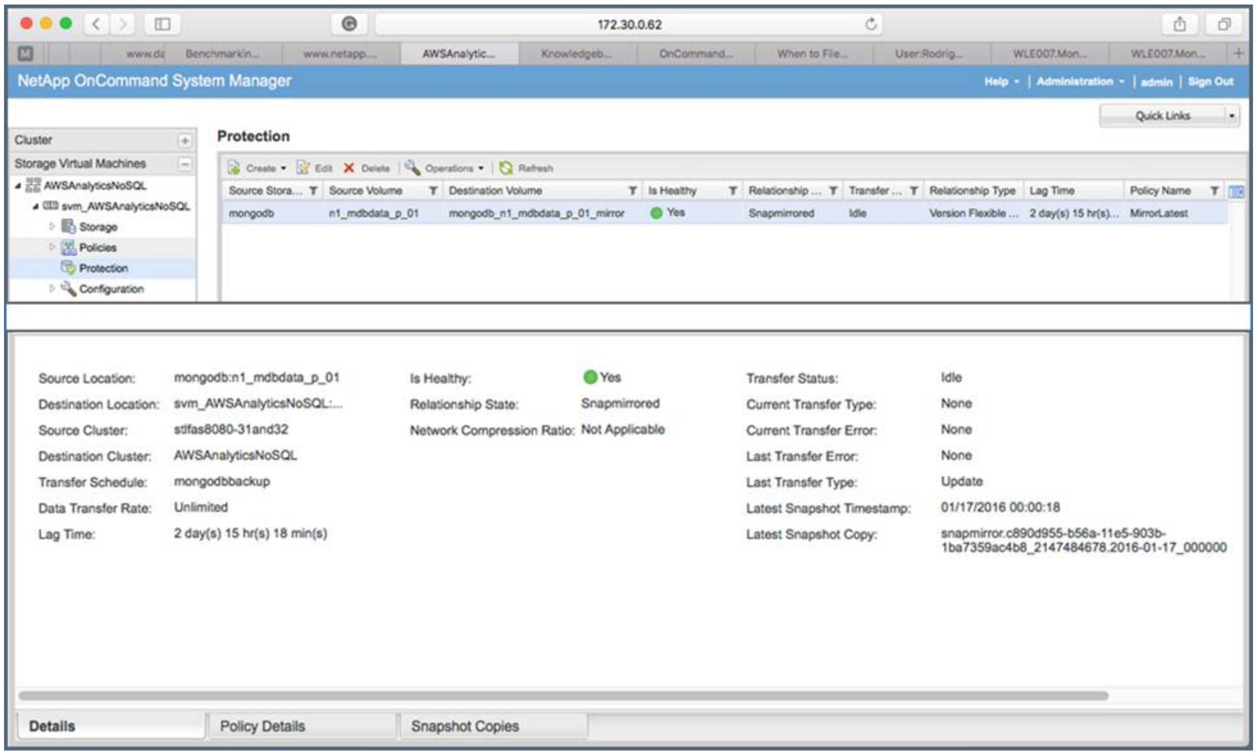
This solution protects the MongoDB database data by using SnapMirror to replicate it into ONTAP Cloud. The replication requires the following tasks:

1. In Cloud Manager, record the IP address of the intercluster LIF for the ONTAP Cloud instance.

2. Create a cluster peer relationship on ONTAP.
3. Create the peer relationship on the ONTAP Cloud instance.
4. Configure and initialize a SnapMirror relationship.

   Note: Figure 6 shows the replicated MongoDB database as viewed from the NetApp OnCommand System Manager. Figure 7 shows the same information as viewed from the NetApp ONTAP Cloud Manager.

Figure 6) Views of the replicated MongoDB database from NetApp OnCommand System Manager.

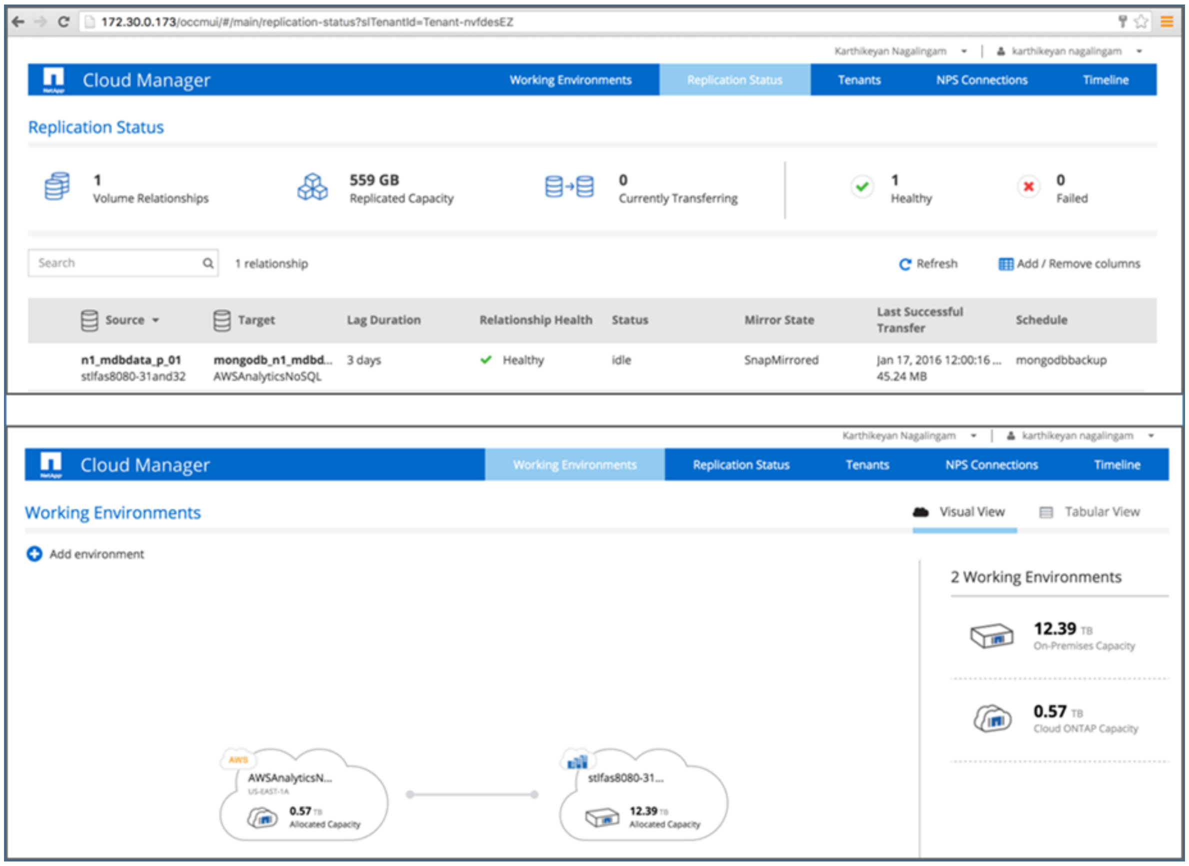


Protection

Source Stora...	Source Volume	Destination Volume	Is Healthy	Relationship ...	Transfer ...	Relationship Type	Lag Time	Policy Name
mongodb	n1_mdbdata_p_01	mongodb_n1_mdbdata_p_01_mirror	● Yes	Snapmirrored	Idle	Version Flexible ...	2 day(s) 15 hr(s)...	MirrorLatest

Source Location:	mongodb:n1_mdbdata_p_01	Is Healthy:	● Yes	Transfer Status:	Idle
Destination Location:	svm_AWSAnalyticsNoSQL:...	Relationship State:	Snapmirrored	Current Transfer Type:	None
Source Cluster:	stlfas8080-31and32	Network Compression Ratio:	Not Applicable	Current Transfer Error:	None
Destination Cluster:	AWSAnalyticsNoSQL			Last Transfer Error:	None
Transfer Schedule:	mongodbbackup			Last Transfer Type:	Update
Data Transfer Rate:	Unlimited			Latest Snapshot Timestamp:	01/17/2016 00:00:18
Lag Time:	2 day(s) 15 hr(s) 18 min(s)			Latest Snapshot Copy:	snapmirror.c890d955-b56a-11e5-903b-1ba7359ac4b8_2147484678.2016-01-17_000000

Figure 7) Views of the replicated MongoDB database from ONTAP Cloud Manager.



172.30.0.173/occmui/#/main/replication-status?slTenantId=Tenant-nvfdesEZ

Replication Status

1 Volume Relationships — 559 GB Replicated Capacity — 0 Currently Transferring — 1 Healthy — 0 Failed

Source	Target	Lag Duration	Relationship Health	Status	Mirror State	Last Successful Transfer	Schedule
n1_mdbdata_p_01 stlfas8080-31and32	mongodb_n1_mdbd... AWSAnalyticsNoSQL	3 days	✓ Healthy	idle	SnapMirrored	Jan 17, 2016 12:00:16 ... 45.24 MB	mongodbbackup

Working Environments

2 Working Environments

12.39 TB On-Premises Capacity

0.57 TB Cloud ONTAP Capacity

5. Verify that the SnapMirror status says `snapmirrored` and the relationship status says `healthy`.
6. Enable disaster recovery for this SnapMirror relationship by completing the following tasks:
   a. Build the Linux operating system VM instance on AWS and install the MongoDB database on that instance just as on the production database.
   b. In NetApp ONTAP Cloud storage, quiesce and break the SnapMirror relationship.
   c. Create the FlexClone volume from the destination volume.
   d. Mount the primary and secondary database volumes in the appropriate operating system instances.
   e. Start the config servers, the primary and secondary sharding servers, and the application or query server mongodb processes by using the `init` script or CLI commands.
   f. Verify that the database and the documents are uploaded to the SnapMirror break point and that the new disaster recovery database is still writable.

4.6 MongoDB with NetApp Private Storage

This solution protects the MongoDB database data by using SnapMirror to replicate it into NetApp Private Storage (NPS) and access the data from a cloud provider such as AWS or Azure. The replication requires the following tasks:

1. In our lab, we have private access to AWS from our lab network. We configure and initialize the SnapMirror relationship between mongodb data volumes and NPS volumes.
2. Verify that the SnapMirror status says `snapmirrored` and the relationship status says `healthy`.

3. Enable disaster recovery for this SnapMirror relationship by completing the following tasks:
   a. Build the Linux operating system VM instance on AWS and install the MongoDB database on that instance just as on the production database.
   b. In NetApp Private Storage, quiesce and break the SnapMirror relationship.
   c. Create the FlexClone volume from the destination volume.
   d. Mount the primary and secondary database volumes in the appropriate operating system instances.
   e. Start the config servers, the primary and secondary sharding servers, and the application or query server mongodb processes by using the `init` script or CLI commands.
   f. Verify that the database and the documents are uploaded to the SnapMirror break point and that the new disaster recovery database is still writable.

4.7 Performance and Certification Validation

Certification Validation

In order to validate the interoperability of NetApp storage with MongoDB, we completed the MongoDB certification process. For the certification validation, we used an A300 all-flash array. The A300 was connected through a Fibre Channel switch to an x86 server with 24 cores and 128GB RAM, running the RHEL 7.1 operating system. We installed MongoDB Enterprise v3.2 on the RHEL server.

Using this configuration, the NetApp storage system successfully passed all MongoDB certification tests and is currently listed as a MongoDB certified platform.

For more information about MongoDB certification with NetApp storage controllers, see the MongoDB NetApp partner webpage.

In addition to the certification testing, we also validated MongoDB performance with NetApp storage using the YCSB test framework. The following sections provide more details about the performance test configuration and results.

4.8 Performance Validation with YCSB

Yahoo! Cloud Serving Benchmark (YCSB) is a framework and common set of workloads for evaluating the performance of different key-value and cloud serving stores. It has two components:

- The YCSB client, an extensible workload generator
- The core workloads, a set of workload scenarios to be executed by the generator

4.9 Workloads Used

Out of six different core workloads, we chose workloads A, B, C and customized C.

Workload A

Workload A is an update heavy workload that has a mix of 50/50 reads and writes. An application example is a session store recording recent actions.

Workload B

Workload B is a read mostly workload and has a 95/5 mix of reads and writes. Application examples are photo tagging or adding a tag as an update, but most operations are to read tags.

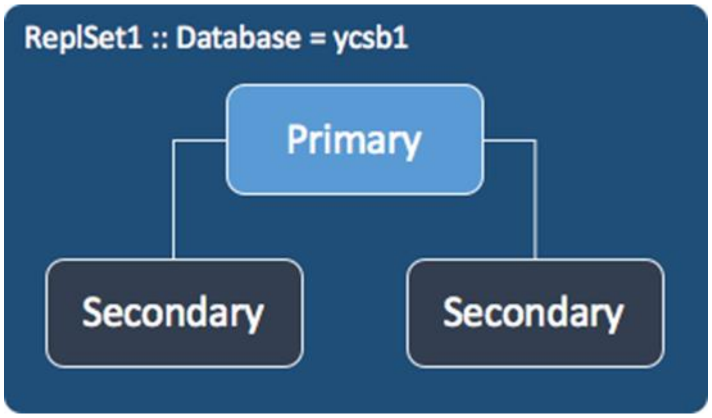
Workload C

Workload C is a read only workload with 100% read. An application example is a user profile cache, where profiles are constructed elsewhere such as Hadoop deployments.

Workload Customized C

The main purposes of YCSB is testing the database performance. It has not been designed to stress the I/O layer of the database. Workload customized C was necessary to demonstrate what level of throughput and response time you can expect from an AFF A300 if it is heavily loaded with application I/O requests.

Figure 8) Replica set used for the validation.

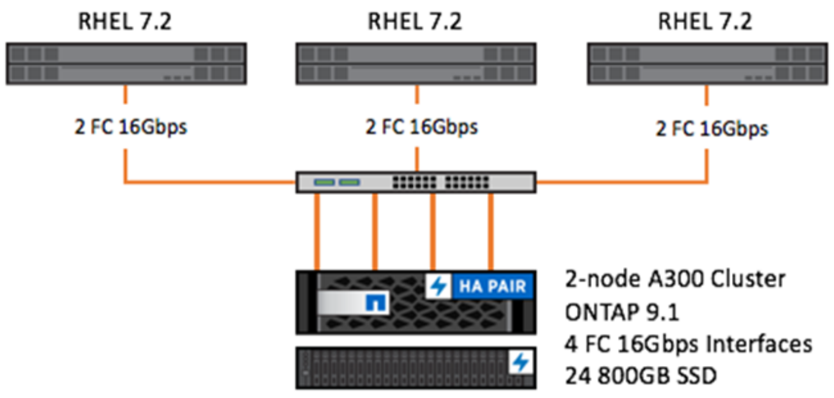


In our validation, we have a database with 700GB of data. Each document is 1,000 bytes in size with 10 fields per document. The data is loaded by the YCSB tool.

4.10 Storage Architecture Used for YCSB

We used three physical servers to build a primary, secondary, secondary (PSS) MongoDB replica set.

Figure 9) Storage architecture used for YCSB.



Each server has the following configurations:

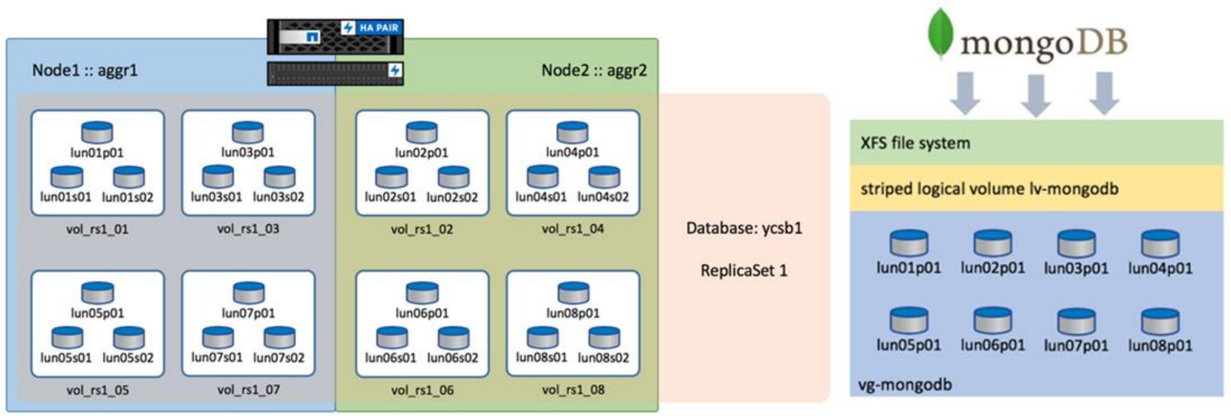
- 256GB RAM
- 2 Intel Xeon E5-2630 v2 at 2.60GHz (24 cores total)
- 2 Fibre Channel 16Gbps ports
- MongoDB servers installed with Red Hat Enterprise Linux 7.2

- Brocade Fibre Channel switch
- Storage controller is AFF A300 HA pair, and each controller has:
  - 16 cores
  - 128GB of cache
  - 8GB of NVMEM
  - 1TB Flash Cache™
  - 4 Fibre Channel 16Gbps ports
  - 1/2 DS2246 with 24 800GB SSDs

For our testing, we built the file system for replica set in the following way:

1. We created four volumes per A300 storage controller, so there were eight volumes total. Each volume size is 386GB.
2. Each volume had three LUNs with a LUN size of 128GB. We provisioned one LUN in each volume to the primary server and the remaining two LUNs to the secondary servers.
3. We created a logical volume group wrapping up eight LUNs per server; then one striped logical volume was created across the eight LUNs in the volume group.
4. The XFS file system was created on top of the striped logical volume.

Figure 10) Storage layout on A300 and layers of the file system.



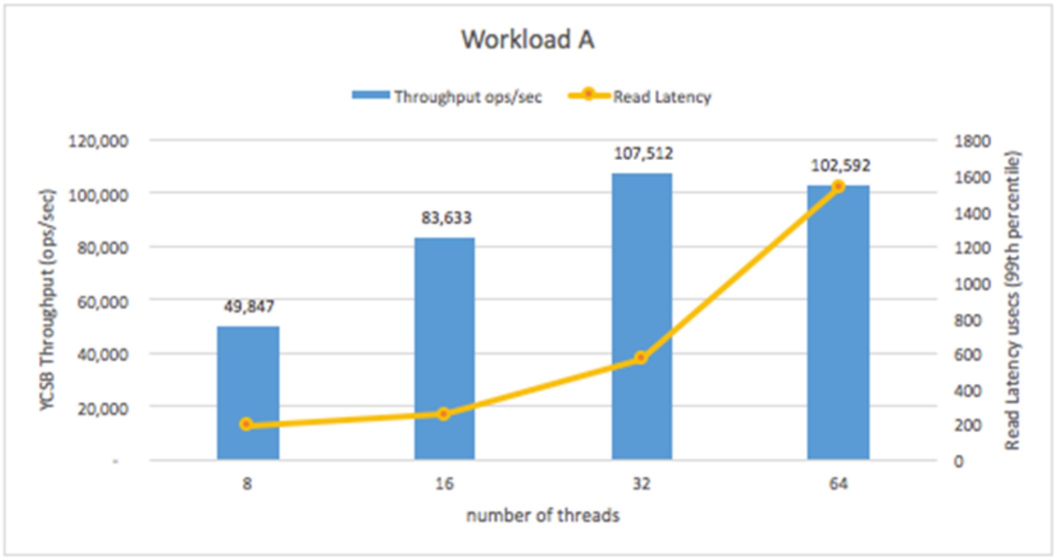
4.11 YCSB Performance Testing

The following specifications are performed against YCSB workloads A, B, and C:

- One YCSB instance runs against one database.
- The test starts with a YCSB thread count of 8.
- The number of threads is increased until latency increases and ops decreases.
- YCSB's performance metrics are collected from YCSB's output at the end of each run.

Workload A is the 50/50% query/update workloads. Workload B is the 95/5% query/update workload, and workload C is 100% queries. During YCSB execution against the data, the load on the server gradually increases. When the collection index is loaded in the server's memory, all the reads coming from servers' memory and writes go to the storage.

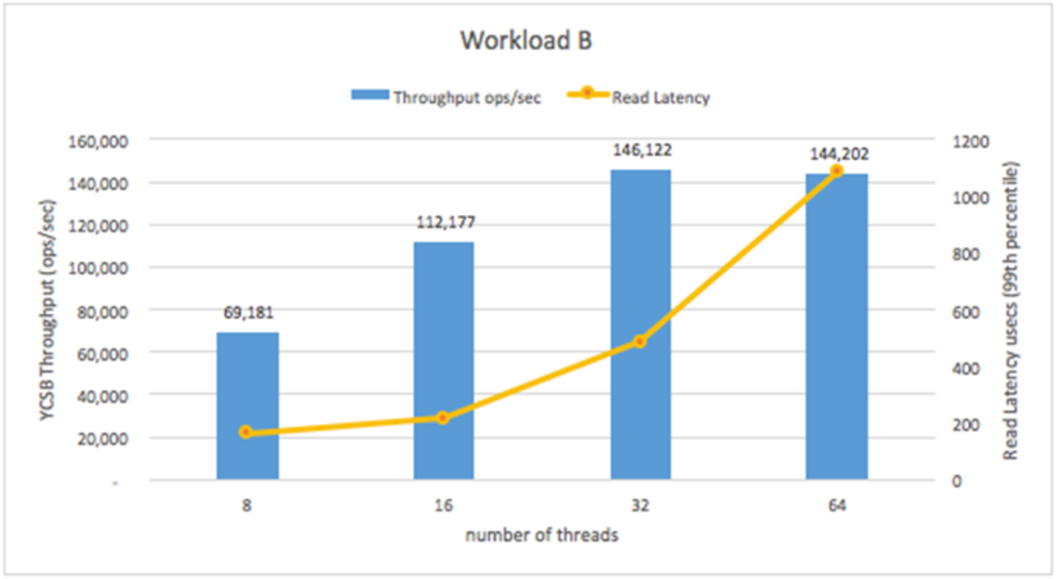
Figure 11) Workload A: 50% read, 50% update.



The following command was used to run workload A, where threads were 8, 16, 32, and 64:

```
/root/YCSB/bin/ycsb run mongodb -s -P /root/YCSB/workloads/workloada -p
mongodb.url="mongodb://mongodb-s1:27017/ycsb1" -p operationcount=120000000 -p
maxexecutiontime=1800 -threads 16
```

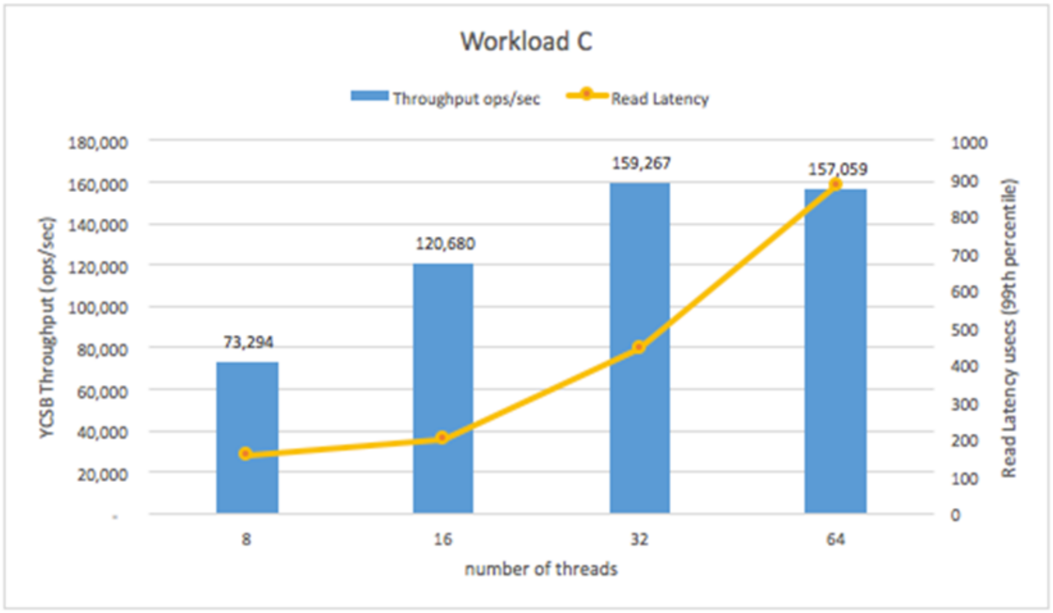
Figure 12) Workload B: 95% read, 5% update.



The following command was used to run workload B, where threads were 8, 16, 32, and 64:

```
/root/YCSB/bin/ycsb run mongodb -s -P /root/YCSB/ orkloads/workloada -p
mongodb.url="mongodb://mongodb-s1:27017/ycsb1" -p operationcount=120000000 -p
maxexecutiontime=1800 -threads 16
```

Figure 13) Workload C: 100% read.



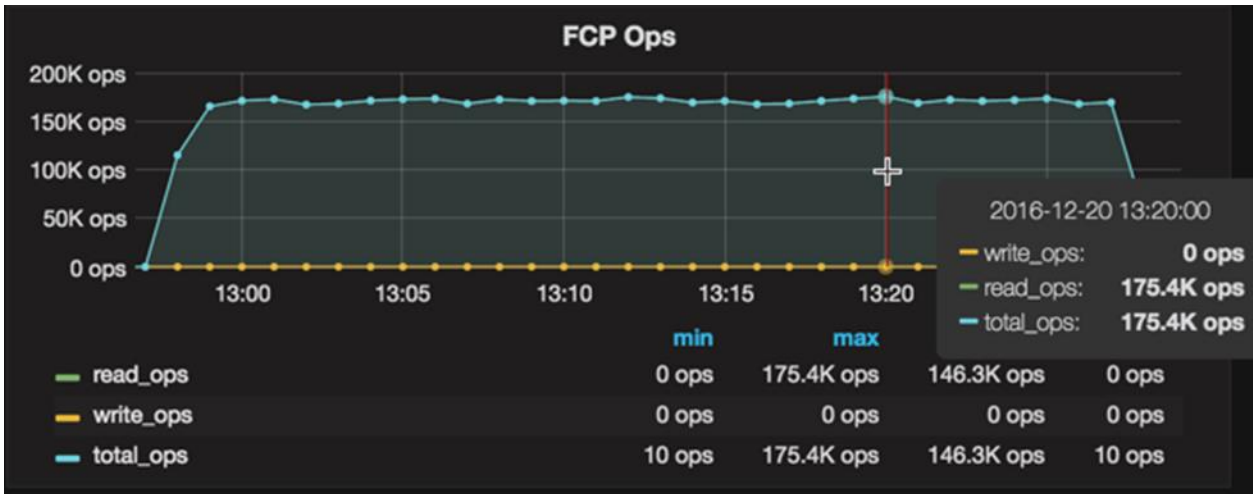
The following command was used to run workload C:

```
/root/YCSB/bin/ycsb run mongodb -s -P /root/YCSB/workloads/workloadc -p
mongodb.url="mongodb://mongodb-s1:27017/ycsb1" -p operationcount=120000000 -p
maxexecutiontime=1800 -threads 16
```

Workload customized C is using the YCSB options scanproportion and maxscanlength. These options allow YCSB to walk through all the documents returned by the MongoDB cursor. When these options are used, queries submitted to the database generate a considerable amount of random reads at the storage level.

The AFF A300 was able to handle storage measured 175,400 IOPS at 0.53ms, reading 2.1GBps of data.

Figure 14) Workload customized C: 100% read, FCP throughput.



	min	max		
read_ops	0 ops	175.4K ops	146.3K ops	0 ops
write_ops	0 ops	0 ops	0 ops	0 ops
total_ops	10 ops	175.4K ops	146.3K ops	10 ops

Figure 15) Workload customized C: 100% read, FCP latency.

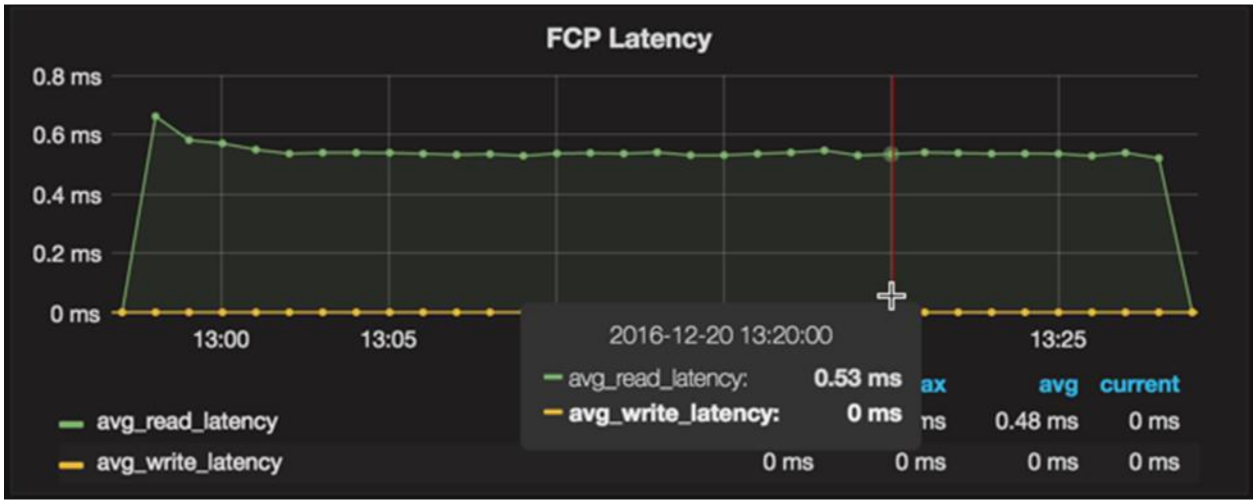
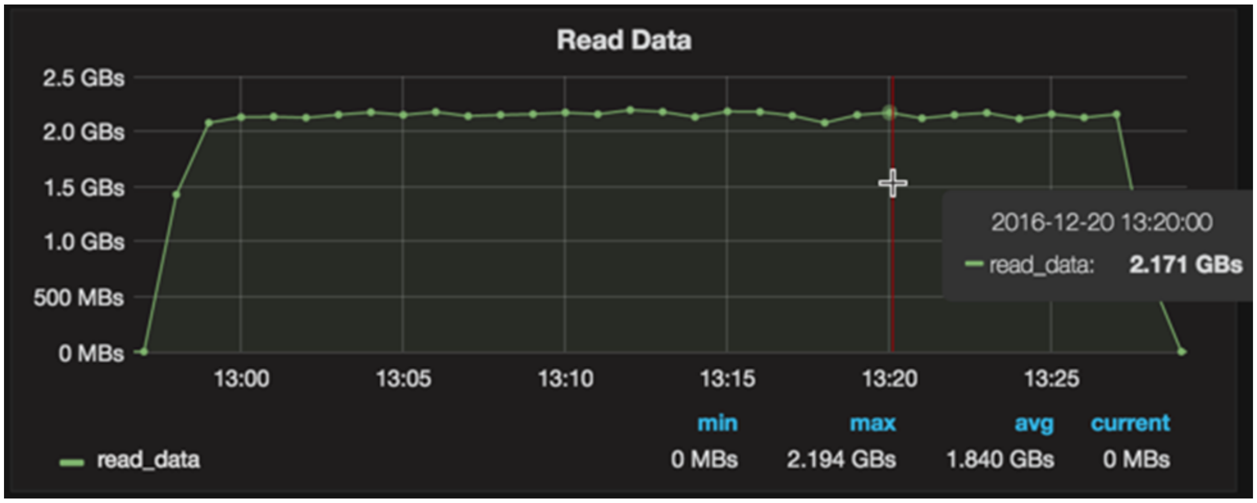


Figure 16) Workload customized C: 100% reads, read throughput in GBps.



The following command was used to run workload customized C:

```
/root/YCSB/bin/ycsb run mongodb -s -P /root/YCSB/workloads/workloadc -p
mongodb.url="mongodb://mongodb-s1:27017/ycsb1" -p operationcount=120000000 -p
maxexecutiontime=1800 -p scanproportion=1 -p updateproportion=0 -p readproportion=0 -p
maxscanlength=8000 -threads 64
```

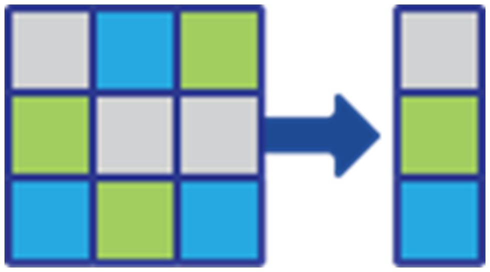
4.12 Inline Efficiency

NetApp deduplication and compression are inline storage efficiency technologies that increase the effective capacity of SSDs and thus reduce the effective cost per gigabyte. Inline storage efficiencies can also improve system performance because fewer physical blocks are written to store a given amount of logical data. Inline deduplication and inline compression are two key components of NetApp AFF that enable users to store the maximum amount of data for the lowest possible cost. These complementary technologies work together to achieve optimal savings.

4.13 Deduplication

Deduplication reduces storage capacity requirements by eliminating redundant blocks of data. Figure 17 illustrates this concept. NetApp deduplication improves storage efficiency by locating identical 4K blocks of data in a flexible volume (a NetApp FlexVol® volume) and replacing the identical blocks with references to a single shared block. Before it removes the blocks, it performs a byte-level verification check to confirm that the blocks are identical. This check prevents potential problems that are related to hash collisions.

Figure 17) Reducing storage capacity requirements through deduplication.



NetApp AFF systems support deduplication as both an inline operation and a postprocess operation. As writes enter the system, inline deduplication works to eliminate any duplicate blocks that are stored in memory, thus reducing the amount of space required to store the data.

If postprocess deduplication is enabled, it further tries to reclaim space by scanning the existing data for duplicates and eliminating them. By default, both inline deduplication and inline zero-block deduplication are enabled on AFF systems. Inline zero-block deduplication detects zero blocks inline and deduplicates them before they are written to disk.

4.14 Compression

NetApp compression provides transparent data compression at the file level. NetApp data compression does not compress the entire file as a single contiguous stream of bytes. Doing so would make it prohibitively expensive to service small reads or overwrites from part of a file because the entire file would have to be read from disk and uncompressed before the request could be served. This requirement would be especially difficult in large files.

To prevent this problem, NetApp data compression compresses a small group of consecutive blocks, known as a compression group. In this arrangement, when a read or an overwrite request arrives, only a small group of blocks must be read, not the entire file. This process optimizes read and overwrite performance and enables greater scalability in the size of the files being compressed.

NetApp supports two types of compression:

- Adaptive compression (8K compression group size)
- Secondary compression (32K compression group size)

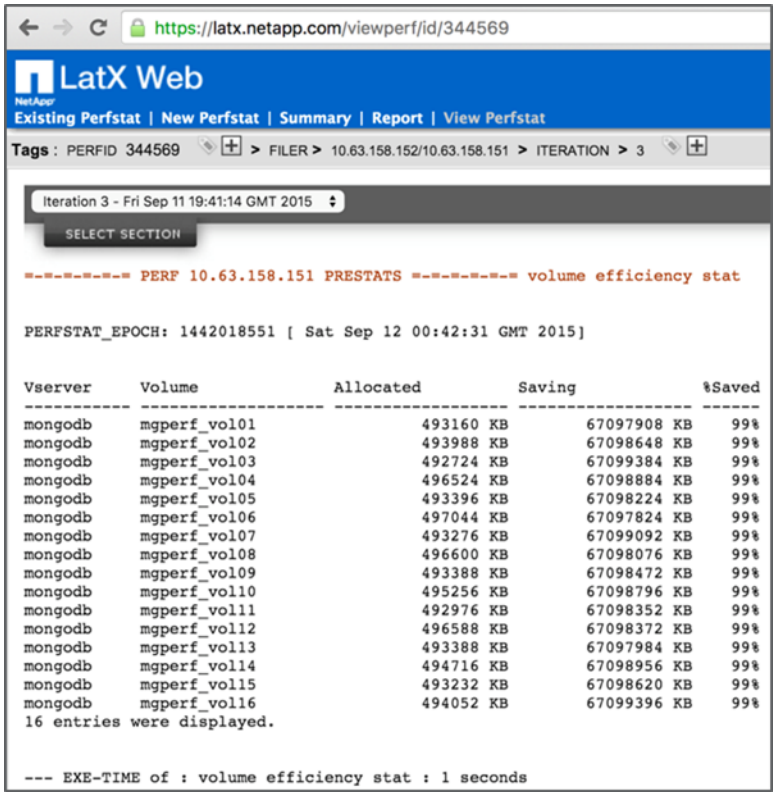
Adaptive compression is better suited for workloads that are predominantly composed of random writes or a mix of sequential and random writes. Secondary compression is better suited for workloads that are composed mostly of large sequential I/Os (32K or larger).

Compression is supported only as an inline operation on AFF systems. Adaptive inline compression is enabled by default for all new volumes on AFF systems.

If both compression and deduplication are enabled in a system, compression happens first, followed by deduplication. Both compression and deduplication can be enabled or disabled at the flexible volume level.

The results in Figure 18 show the storage efficiency statistics of MongoDB running the mongoperf workload on All Flash FAS controllers. The results in Figure 19 indicate a data reduction ratio of greater than 15:1 with both inline deduplication and inline compression.

Figure 18) Volume efficiency statistics from Perfstat.


https://latx.netapp.com/viewperf/id/344569
LatX Web
NetApp
Existing Perfstat | New Perfstat | Summary | Report | View Perfstat
Tags : PERFID 344569 > FILER > 10.63.158.152/10.63.158.151 > ITERATION > 3
Iteration 3 - Fri Sep 11 19:41:14 GMT 2015
SELECT SECTION

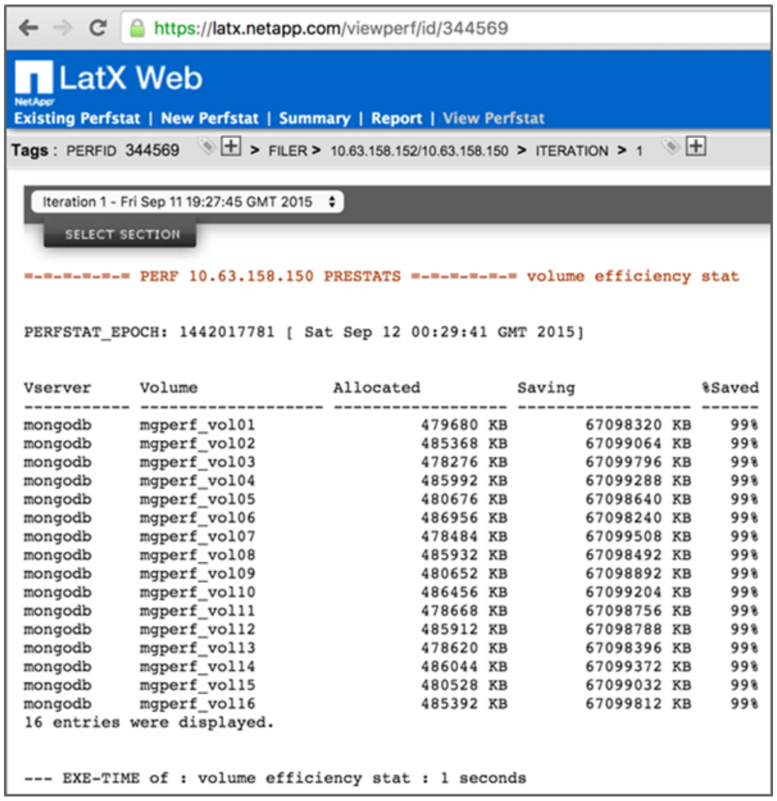
```
=-=-=-=-=-= PERF 10.63.158.151 PRESTATS =-=-=-=-=-= volume efficiency stat


PERFSTAT_EPOCH: 1442018551 [ Sat Sep 12 00:42:31 GMT 2015]


Vserver     Volume              Allocated          Saving             %Saved
----------- ------------------- ------------------ ------------------ ------
mongodb     mgperf_vol01                 493160 KB        67097908 KB    99%
mongodb     mgperf_vol02                 493988 KB        67098648 KB    99%
mongodb     mgperf_vol03                 492724 KB        67099384 KB    99%
mongodb     mgperf_vol04                 496524 KB        67098884 KB    99%
mongodb     mgperf_vol05                 493396 KB        67098224 KB    99%
mongodb     mgperf_vol06                 497044 KB        67097824 KB    99%
mongodb     mgperf_vol07                 493276 KB        67099092 KB    99%
mongodb     mgperf_vol08                 496600 KB        67098076 KB    99%
mongodb     mgperf_vol09                 493388 KB        67098472 KB    99%
mongodb     mgperf_vol10                 495256 KB        67098796 KB    99%
mongodb     mgperf_vol11                 492976 KB        67098352 KB    99%
mongodb     mgperf_vol12                 496588 KB        67098372 KB    99%
mongodb     mgperf_vol13                 493388 KB        67097984 KB    99%
mongodb     mgperf_vol14                 494716 KB        67098956 KB    99%
mongodb     mgperf_vol15                 493232 KB        67098620 KB    99%
mongodb     mgperf_vol16                 494052 KB        67099396 KB    99%
16 entries were displayed.


--- EXE-TIME of : volume efficiency stat : 1 seconds
```

Figure 19) Data reduction of over 15:1 with inline deduplication and inline compression.


https://latx.netapp.com/viewperf/id/344569
LatX Web
NetApp
Existing Perfstat | New Perfstat | Summary | Report | View Perfstat
Tags : PERFID 344569 > FILER > 10.63.158.152/10.63.158.150 > ITERATION > 1
Iteration 1 - Fri Sep 11 19:27:45 GMT 2015
SELECT SECTION

```
=-=-=-=-=-= PERF 10.63.158.150 PRESTATS =-=-=-=-=-= volume efficiency stat


PERFSTAT_EPOCH: 1442017781 [ Sat Sep 12 00:29:41 GMT 2015]


Vserver     Volume               Allocated          Saving             %Saved
----------- -------------------- ------------------ ------------------ ------
mongodb     mgperf_vol01                  479680 KB        67098320 KB    99%
mongodb     mgperf_vol02                  485368 KB        67099064 KB    99%
mongodb     mgperf_vol03                  478276 KB        67099796 KB    99%
mongodb     mgperf_vol04                  485992 KB        67099288 KB    99%
mongodb     mgperf_vol05                  480676 KB        67098640 KB    99%
mongodb     mgperf_vol06                  486956 KB        67098240 KB    99%
mongodb     mgperf_vol07                  478484 KB        67099508 KB    99%
mongodb     mgperf_vol08                  485932 KB        67098492 KB    99%
mongodb     mgperf_vol09                  480652 KB        67098892 KB    99%
mongodb     mgperf_vol10                  486456 KB        67099204 KB    99%
mongodb     mgperf_vol11                  478668 KB        67098756 KB    99%
mongodb     mgperf_vol12                  485912 KB        67098788 KB    99%
mongodb     mgperf_vol13                  478620 KB        67098396 KB    99%
mongodb     mgperf_vol14                  486044 KB        67099372 KB    99%
mongodb     mgperf_vol15                  480528 KB        67099032 KB    99%
mongodb     mgperf_vol16                  485392 KB        67099812 KB    99%
16 entries were displayed.


--- EXE-TIME of : volume efficiency stat : 1 seconds
```

Important Note

Because of its inline deduplication and compression capabilities, the NetApp All Flash FAS demonstrated a data reduction ratio of greater than 15:1 throughout the testing.

Note: The mongoperf workload writes many zeros in its read and write operations. Therefore, your storage efficiency might vary depending on the type and amount of data in your deployment.

5 Summary

The reference architecture described in this document provides an end-to-end solution for efficiently deploying a virtualized scale-out MongoDB NoSQL database on the NetApp Data Fabric. In the tested architecture of NetApp storage technology and VMware vSphere, MongoDB VMs and databases were hosted on the scale-out NetApp AFF array. Backups were created by using Snap Creator, and disaster recovery was handled through a NetApp ONTAP Cloud software-defined storage solution and NetApp Private Storage solutions in AWS.

The following key benefits were validated in this solution:

- Predictable high performance with consistent low latency, providing very fast response time to the most demanding analytics applications built on MongoDB
- Inline deduplication and compression to achieve cost efficiency for MongoDB on flash storage
- Instant, space-efficient database cloning for rapid setup of dev/test and QA environments without the need to buy new storage

- Backup and recovery based on space-efficient NetApp Snapshot copies and on remote replication to the cloud

In addition to the validation, NetApp AFF strongly complements MongoDB with the following key capabilities:

- Scale-up and scale-out all-flash array to modularly scale storage with MongoDB
- Nondisruptive operations to deliver maximum uptime and high availability

During the YCSB performance testing, NetApp AFF demonstrated excellent test results to deliver a high-performance, cost-effective all-flash array for scaling out MongoDB deployments.

Appendix A: Python Scripts

Copy the Python scripts to `/usr/local/bin` in MongoDB application servers.

1. `fsync-lock.py` (This script is required only for the MMAPv1 storage engine.)

```
import random
import time
import sys
from pymongo import Connection
server=sys.argv[1]
port=int(sys.argv[2])
connection = Connection(server, port )
db = connection.admin #connect to database admin
db.command("fsync", lock=True)
```

2. `fsync-unlock.py` (This script is required only for the MMAP storage engine.)

```
import random
import time
import sys
from pymongo import Connection
server=sys.argv[1]
port=int(sys.argv[2])
connection = Connection(server, port )
db = connection.admin #connect to database admin
db["$cmd.sys.unlock"].find_one()
```

`3. wait-for-cleared-locks.py`

```
import random
import time
import sys
from pymongo.son_manipulator import SONManipulator
from pymongo import Connection
server=sys.argv[1]
port=int(sys.argv[2])
#connection = Connection('appserver', 27017 )
connection = Connection(server, port )
class Transform(SONManipulator):
   def transform_incoming(self, son, collection):
     for (key, value) in son.items():
       if isinstance(value, Custom):
         son[key] = encode_custom(value)
       elif isinstance(value, dict): # Make sure we recurse into sub-docs
         son[key] = self.transform_incoming(value, collection)
     return son
   def transform_outgoing(self, son, collection):
     for (key, value) in son.items():
       if isinstance(value, dict):
         if "_type" in value and value["_type"] == "custom":
son[key] = decode_custom(value) else: # Again, make sure to recurse into sub-docs
           son[key] = self.transform_outgoing(value, collection)
     return son
db = connection.config #connect to database config
db.add_son_manipulator(Transform())
```

```
balancer_status = db.locks.find_one({'_id': 'balancer' })['state']
print balancer_status
if balancer_status > 0 : #not null
balancer_status = db.locks.find_one({'_id': 'balancer'})['state'] while ( balancer_status != 0 ):
#wait until balancer has stopped
     time.sleep(1)
     balancer_status = db.locks.find_one({'_id': 'balancer'})['state']
     print balancer_status
#STOP the balancer
db.settings.update( { '_id': 'balancer' }, { '$set': { 'stopped': True } } )
```

4. `start-balancer.py`

```
import random
import time
import sys
from pymongo import Connection
server=sys.argv[1]
port=int(sys.argv[2])
connection = Connection(server, port )

db = connection.config #connect to database config servers
db.settings.update( { "_id":  "balancer" }, { "$set":  { "stopped": False } } ) #start balancer
```

Appendix B: MongoDB Operations

Use the following steps to configure the sharding cluster with MongoDB operations and add two shards into the sharding cluster.

1. Check the mounted partitions for the MongoDB database:
   - `df -h` and `/etc/fstab` file for `mdb-srv-1`, `mdb-srv-10`, and `mdb-srv11`:

```
[mongodb@mdb-srv-1 db_wt]$ df -h /data
Filesystem             Size  Used Avail Use% Mounted on
/dev/sdb1             1014G 1014G  204M 100% /data
[mongodb@mdb-srv-1 db_wt]$
[mongodb@mdb-srv-1 db_wt]$ cat /etc/fstab | grep sdb
/dev/sdb1       /data    xfs     defaults        0 0
[mongodb@mdb-srv-1 db_wt]$

[mongodb@mdb-srv-10 db_wt]$ df -h /data
Filesystem             Size  Used Avail Use% Mounted on
/dev/sdb1             1014G   62G  952G   7% /data
[mongodb@mdb-srv-10 db_wt]$
[mongodb@mdb-srv-10 db_wt]$ cat /etc/fstab | grep sdb
/dev/sdb1       /data    xfs     defaults        0 0
[mongodb@mdb-srv-10 db_wt]$

[mongodb@mdb-srv-11 ~]$ df -h /data
Filesystem             Size  Used Avail Use% Mounted on
/dev/sdb1             1014G  166G  849G  17% /data
[mongodb@mdb-srv-11 ~]$
[mongodb@mdb-srv-11 ~]$ cat /etc/fstab | grep sdb
/dev/sdb1       /data    xfs     defaults       0 0
[mongodb@mdb-srv-11 ~]$
```

   - `df -h` and `/etc/fstab` file for `mdb-srv-2`, `mdb-srv-20`, and `mdb-srv21`:

```
[mongodb@mdb-srv-2 db_wt]$ df -h /data
Filesystem             Size  Used Avail Use% Mounted on
/dev/sdb1             1014G  154G  861G  16% /data
[mongodb@mdb-srv-2 db_wt]$ cat /etc/fstab | grep sdb
/dev/sdb1       /data    xfs     defaults        0 0
[mongodb@mdb-srv-2 db_wt]$

[mongodb@mdb-srv-20 ~]$ df -h /data
Filesystem             Size  Used Avail Use% Mounted on
/dev/sdb1             1014G  688G  326G  68% /data
[mongodb@mdb-srv-20 ~]$
```

```
[mongodb@mdb-srv-20 ~]$ cat /etc/fstab | grep sdb
/dev/sdb1       /data    xfs     defaults        0 0
[mongodb@mdb-srv-20 ~]$

[mongodb@mdb-srv-21 ~]$ df -h /data
Filesystem             Size  Used Avail Use% Mounted on
/dev/sdb1             1014G 1014G   20K 100% /data
[mongodb@mdb-srv-21 ~]$
[mongodb@mdb-srv-21 ~]$ cat /etc/fstab | grep sdb
/dev/sdb1       /data    xfs     defaults       0 0
[mongodb@mdb-srv-21 ~]$
```

2. Check and run the shard cluster members in the MongoDB sharding cluster; create the required folders, and provide the required permissions.

   Note: The mongod process starts with default ports (27018 and 28018). Alternatively, you can specify ports.

   a. Check and run the mongod process in the `mdb-srv-1` sharding member.

```
[root@mdb-srv-1 ~]# groupadd -g 8000 mongodb
[root@mdb-srv-1 ~]# useradd -u 8000 -g 8000 -d /mongodb mongodb
[root@mdb-srv-1 ~]# chown -R mongodb:mongodb /mongodb
[root@mdb-srv-1 ~]# chmod -R 700 /mongodb
[root@mdb-srv-1 ~]# su - mongodb
[mongodb@mdb-srv-1 ~]$ mongod --replSet shard1/mdb-srv-10,mdb-srv-11 --journal --rest --shardsvr
--fork --logpath /mongodb/data/mongod.log
2015-11-22T11:24:54.898-0500 I CONTROL  ** WARNING: --rest is specified without --httpinterface,
2015-11-22T11:24:54.898-0500 I CONTROL  **          enabling http interface
about to fork child process, waiting until server is ready for connections.
forked process: 23285
child process started successfully, parent exiting
[mongodb@mdb-srv-1 ~]$ netstat -lntp
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:27018           0.0.0.0:*               LISTEN      23285/mongod
tcp        0      0 0.0.0.0:28018           0.0.0.0:*               LISTEN      23285/mongod
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN      -
tcp6       0      0 :::22                   :::*                    LISTEN      -
tcp6       0      0 ::1:631                 :::*                    LISTEN      -
tcp6       0      0 ::1:25                  :::*                    LISTEN      -
[mongodb@mdb-srv-1 ~]$
```

   b. Check and run the mongod process in the `mdb-srv-10` sharding member.

```
[root@mdb-srv-10 ~]# groupadd -g 8000 mongodb
[root@mdb-srv-10 ~]# useradd -u 8000 -g 8000 -d /mongodb mongodb
[root@mdb-srv-10 ~]#  chown -R mongodb:mongodb /mongodb
[root@mdb-srv-10 ~]# chmod -R 700 /mongodb
[root@mdb-srv-10 ~]# mkdir -p /mongodb/data
[root@mdb-srv-10 ~]# mkdir -p /data/db
[root@mdb-srv-10 ~]#  chown -R mongodb:mongodb /mongodb
[root@mdb-srv-10 ~]#  chown -R mongodb:mongodb /data
[root@mdb-srv-10 ~]# su - mongodb
[mongodb@mdb-srv-10 ~]$  mongod --replSet shard1/mdb-srv-1,mdb-srv-11 --journal --rest --shardsvr
--fork --logpath /mongodb/data/mongod.log
2015-11-22T11:26:15.732-0500 I CONTROL  ** WARNING: --rest is specified without --httpinterface,
2015-11-22T11:26:15.732-0500 I CONTROL  **          enabling http interface
about to fork child process, waiting until server is ready for connections.
forked process: 11253
child process started successfully, parent exiting
[mongodb@mdb-srv-10 ~]$
[mongodb@mdb-srv-10 ~]$ netstat -lntp
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
```

```
tcp        0      0 0.0.0.0:27018           0.0.0.0:*               LISTEN      11253/mongod
tcp        0      0 0.0.0.0:28018           0.0.0.0:*               LISTEN      11253/mongod
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN      -
tcp6       0      0 :::22                   :::*                    LISTEN      -
tcp6       0      0 ::1:25                  :::*                    LISTEN      -
[mongodb@mdb-srv-10 ~]$
```

c. Check and run the mongod process in the `mdb-srv-11` sharding member.

```
[root@mdb-srv-11 ~]# groupadd -g 8000 mongodb
[root@mdb-srv-11 ~]# useradd -u 8000 -g 8000 -d /mongodb mongodb
[root@mdb-srv-11 ~]#  chown -R mongodb:mongodb /mongodb
[root@mdb-srv-11 ~]# chmod -R 700 /mongodb
[root@mdb-srv-11 ~]# mkdir -p /mongodb/data
[root@mdb-srv-11 ~]# mkdir -p /data/db
[root@mdb-srv-11 ~]#  chown -R mongodb:mongodb /mongodb
[root@mdb-srv-11 ~]#  chown -R mongodb:mongodb /data
[root@mdb-srv-11 ~]# su - mongodb
[mongodb@mdb-srv-11 ~]$ mongod --replSet shard1/mdb-srv-1,mdb-srv-10 --journal --rest --shardsvr
--fork --logpath /mongodb/data/mongod.log
2015-11-22T11:32:49.930-0500 I CONTROL  ** WARNING: --rest is specified without --httpinterface,
2015-11-22T11:32:49.930-0500 I CONTROL  **          enabling http interface
about to fork child process, waiting until server is ready for connections.
forked process: 11411
child process started successfully, parent exiting
[mongodb@mdb-srv-11 ~]$
[mongodb@mdb-srv-11 ~]$ netstat -lntp
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:27018           0.0.0.0:*               LISTEN      11411/mongod
tcp        0      0 0.0.0.0:28018           0.0.0.0:*               LISTEN      11411/mongod
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN      -
tcp6       0      0 :::22                   :::*                    LISTEN      -
tcp6       0      0 ::1:25                  :::*                    LISTEN      -
[mongodb@mdb-srv-11 ~]$
```

3. Verify that all config servers in MongoDB folders such as logpath (`/mongodb/data/configdb`) and dbpath (`/data/db`) are created. Verify that they have mongodb user permission (700) and that the mongod process is running in fork mode and on the right port (27019).
   – `mdb-cfg-3` config server:

```
[root@mdb-cfg-3 ~]# groupadd -g 8000 mongodb
[root@mdb-cfg-3 ~]# useradd -u 8000 -g 8000 -d /mongodb mongodb
[root@mdb-cfg-3 ~]# chown -R mongodb:mongodb /mongodb
[root@mdb-cfg-3 ~]# mkdir -p /mongodb/data
[root@mdb-cfg-3 ~]# mkdir -p /data/db
[root@mdb-cfg-3 ~]# chown -R mongodb:mongodb /mongodb
[root@mdb-cfg-3 ~]# chown -R mongodb:mongodb /data
[root@mdb-cfg-3 ~]# mkdir /mongodb/data/configdb
mkdir: cannot create directory ‘/mongodb/data/configdb’: File exists
[root@mdb-cfg-3 ~]# chmod -R 700 /mongodb
[root@mdb-cfg-3 ~]# chown -R mongodb:mongodb /mongodb
[root@mdb-cfg-3 ~]#
[root@mdb-cfg-3 ~]# su - mongodb
Last login: Sun Nov 22 16:09:05 EST 2015 on pts/1
[mongodb@mdb-cfg-3 ~]$ mongod --configsvr --dbpath /mongodb/data/configdb --port 27019 --fork --
logpath /mongodb/data/arbiter.log
about to fork child process, waiting until server is ready for connections.
forked process: 30546
child process started successfully, parent exiting
[mongodb@mdb-cfg-3 ~]$
[mongodb@mdb-cfg-3 ~]$ netstat -lntp | grep 27
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
tcp        0      0 0.0.0.0:27019           0.0.0.0:*               LISTEN      30546/mongod
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN      -
[mongodb@mdb-cfg-3 ~]$
```

– `mdb-cfg-1` config server:

```
[root@mdb-cfg-1 ~]# mongod --configsvr --dbpath /mongodb/data/configdb --port 27019 --fork --
logpath /mongodb/data/arbiter.log
about to fork child process, waiting until server is ready for connections.
forked process: 19925
child process started successfully, parent exiting
[root@mdb-cfg-1 ~]# netstat -lntp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:27019           0.0.0.0:*               LISTEN      19925/mongod
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      1035/sshd
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN      1618/cupsd
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN      1163/master
tcp6       0      0 :::22                   :::*                    LISTEN      1035/sshd
tcp6       0      0 ::1:631                 :::*                    LISTEN      1618/cupsd
tcp6       0      0 ::1:25                  :::*                    LISTEN      1163/master
[root@mdb-cfg-1 ~]#
```

– `mdb-cfg-2` config server:

```
[root@mdb-cfg-2 ~]# su - mongodb
Last login: Sun Nov 22 15:38:33 EST 2015 on pts/1
[mongodb@mdb-cfg-2 ~]$ ls -ltrh /tmp/mongodb-27019.sock
ls: cannot access /tmp/mongodb-27019.sock: No such file or directory
[mongodb@mdb-cfg-2 ~]$
[mongodb@mdb-cfg-2 ~]$ mongod --configsvr --dbpath /mongodb/data/configdb --port 27019 --fork --
logpath /mongodb/data/arbiter.log
about to fork child process, waiting until server is ready for connections.
forked process: 30020
child process started successfully, parent exiting
[mongodb@mdb-cfg-2 ~]$
```

4. Start the mongos process in the MongoDB router or application server and verify that the port (27017) is listening:

   a. Start the mongos in MongoDB application server `mdb-ms-1`.

```
[root@mdb-ms-1 ~]# groupadd -g 8000 mongodb
[root@mdb-ms-1 ~]# useradd -u 8000 -g 8000 -d /mongodb mongodb
[root@mdb-ms-1 ~]# chown -R mongodb:mongodb /mongodb
[root@mdb-ms-1 ~]# mkdir -p /mongodb/data
[root@mdb-ms-1 ~]# mkdir -p /data/db
[root@mdb-ms-1 ~]# chown -R mongodb:mongodb /mongodb
[root@mdb-ms-1 ~]# chown -R mongodb:mongodb /data
[root@mdb-ms-1 ~]# chmod -R 700 /mongodb
[root@mdb-ms-1 ~]# chown -R mongodb:mongodb /mongodb
[root@mdb-ms-1 ~]# su - mongodb
Last login: Sun Nov 22 16:16:10 EST 2015 on pts/2
[mongodb@mdb-ms-1 ~]$ mongos --configdb mdb-cfg-1:27019,mdb-cfg-2:27019,mdb-cfg-3:27019 --logpath
/mongodb/log
login.defs      logrotate.conf  logrotate.d/
[mongodb@mdb-ms-1 ~]$ mongos --configdb mdb-cfg-1:27019,mdb-cfg-2:27019,mdb-cfg-3:27019 --logpath
/mongodb/data/mongos.log --fork --port 27017
about to fork child process, waiting until server is ready for connections.
forked process: 10214
child process started successfully, parent exiting
[mongodb@mdb-ms-1 ~]$
```

   b. Start the mongos process in MongoDB application server `mdb-ms-2`. (Make sure that the required folders are already created and that they have the required permissions.)

```
[mongodb@mdb-ms-2 ~]$  mongos --configdb mdb-cfg-1:27019,mdb-cfg-2:27019,mdb-cfg-3:27019 --
logpath /mongodb/data/mongos.log
2015-11-23T09:24:30.102-0500 I CONTROL  log file "/mongodb/data/mongos.log" exists; moved to
"/mongodb/data/mongos.log.2015-11-23T14-24-30".
```

5. Connect to the MongoDB shard primary server (`mdb-srv-1`) from the MongoDB application server and initiate the replication set.

```
[mongodb@mdb-ms-1 ~]$ mongo mdb-srv-1:27018/admin
MongoDB shell version: 3.0.7
connecting to: mdb-srv-1:27018/admin
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
        http://docs.mongodb.org/
Questions? Try the support group
        http://groups.google.com/group/mongodb-user
Server has startup warnings:
2015-11-22T11:24:54.898-0 500 I CONTROL  ** WARNING: --rest is specified without --httpinterface,
2015-11-22T11:24:54.898-0500 I CONTROL  **          enabling http interface
2015-11-22T11:24:54.942-0500 I STORAGE  [initandlisten]
2015-11-22T11:24:54.943-0500 I STORAGE  [initandlisten] ** WARNING: Readahead for /data/db is set
to 4096KB
2015-11-22T11:24:54.943-0500 I STORAGE  [initandlisten] **          We suggest setting it to
256KB (512 sectors) or less
2015-11-22T11:24:54.943-0500 I STORAGE  [initandlisten] **
http://dochub.mongodb.org/core/readahead
> cfg = {
... _id : "shard1",
... members : [
...     {_id : 0, host : "mdb-srv-1:27018"},
...     {_id : 1, host : "mdb-srv-10:27018"},
...     {_id : 2, host : "mdb-srv-11:27018"}
... ]
... }
{
        "_id" : "shard1",
        "members" : [
                {
                        "_id" : 0,
                        "host" : "mdb-srv-1:27018"
                },
                {
                        "_id" : 1,
                        "host" : "mdb-srv-10:27018"
                },
                {
                        "_id" : 2,
                        "host" : "mdb-srv-11:27018"
                }
        ]
}
> rs.initiate(cfg);
{ "ok" : 1 }
shard1:OTHER> exit
bye
[mongodb@mdb-ms-1 ~]$
```

6. Check the replication set status from the shard member.

```
[mongodb@mdb-srv-1 ~]$ mongo --host mdb-srv-1 --port 27018
MongoDB shell version: 3.0.7
connecting to: mdb-srv-1:27018/test
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
        http://docs.mongodb.org/
Questions? Try the support group
        http://groups.google.com/group/mongodb-user
Server has startup warnings:
2015-11-22T11:24:54.898-0500 I CONTROL  ** WARNING: --rest is specified without --httpinterface,
2015-11-22T11:24:54.898-0500 I CONTROL  **          enabling http interface
2015-11-22T11:24:54.942-0500 I STORAGE  [initandlisten]
2015-11-22T11:24:54.943-0500 I STORAGE  [initandlisten] ** WARNING: Readahead for /data/db is set
to 4096KB
```

```
2015-11-22T11:24:54.943-0500 I STORAGE  [initandlisten] **          We suggest setting it to
256KB (512 sectors) or less
2015-11-22T11:24:54.943-0500 I STORAGE  [initandlisten] **
http://dochub.mongodb.org/core/readahead
shard1:PRIMARY> rs.printReplicationInfo()
configured oplog size:   990MB
log length start to end: 0secs (0hrs)
oplog first event time:  Sun Nov 22 2015 16:26:37 GMT-0500 (EST)
oplog last event time:   Sun Nov 22 2015 16:26:37 GMT-0500 (EST)
now:                     Mon Nov 23 2015 05:11:29 GMT-0500 (EST)
shard1:PRIMARY>

PRIMARY> rs.status()
{
        "set" : "shard1",
        "date" : ISODate("2015-11-23T10:27:01.006Z"),
        "myState" : 1,
        "members" : [
                {
                        "_id" : 0,
                        "name" : "mdb-srv-1:27018",
                        "health" : 1,
                        "state" : 1,
                        "stateStr" : "PRIMARY",
                        "uptime" : 64927,
                        "optime" : Timestamp(1448227597, 1),
                        "optimeDate" : ISODate("2015-11-22T21:26:37Z"),
                        "electionTime" : Timestamp(1448227599, 1),
                        "electionDate" : ISODate("2015-11-22T21:26:39Z"),
                        "configVersion" : 1,
                        "self" : true
                },
                {
                        "_id" : 1,
                        "name" : "mdb-srv-10:27018",
                        "health" : 1,
                        "state" : 2,
                        "stateStr" : "SECONDARY",
                        "uptime" : 46823,
                        "optime" : Timestamp(1448227597, 1),
                        "optimeDate" : ISODate("2015-11-22T21:26:37Z"),
                        "lastHeartbeat" : ISODate("2015-11-23T10:27:00.525Z"),
                        "lastHeartbeatRecv" : ISODate("2015-11-23T10:27:00.524Z"),
                        "pingMs" : 0,
                        "configVersion" : 1
                },
                {
                        "_id" : 2,
                        "name" : "mdb-srv-11:27018",
                        "health" : 1,
                        "state" : 2,
                        "stateStr" : "SECONDARY",
                        "uptime" : 46823,
                        "optime" : Timestamp(1448227597, 1),
                        "optimeDate" : ISODate("2015-11-22T21:26:37Z"),
                        "lastHeartbeat" : ISODate("2015-11-23T10:27:00.525Z"),
                                      "lastHeartbeatRecv" : ISODate("2015-11-
23T10:27:00.524Z"),
                        "pingMs" : 0,
                        "configVersion" : 1
                }
        ],
        "ok" : 1
}
shard1:PRIMARY>
```

7. Add a shard from the application server.

```
 [mongodb@mdb-ms-1 ~]$ mongo  mdb-ms-1:27017/admin
MongoDB shell version: 3.0.7
connecting to: mdb-ms-1:27017/admin
```

```
mongos> db.adminCommand( { addShard : "shard1/mdb-srv-1:27018,mdb-srv-10:27018,mdb-srv-11:27018"
} )
{ "shardAdded" : "shard1", "ok" : 1 }
mongos>
```

8. Try to add the same shard from another application server by connecting to the replication set member.

```
 [mongodb@mdb-ms-2 ~]$  mongo --host mdb-ms-2 --port 27017
MongoDB shell version: 3.0.7
connecting to: mdb-ms-2:27017/test
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
        http://docs.mongodb.org/
Questions? Try the support group
        http://groups.google.com/group/mongodb-user
mongos>  db.adminCommand( { addShard : "shard1/mdb-srv-1:27018,mdb-srv-10:27018,mdb-srv-11:27018"
} )
{ "ok" : 0, "errmsg" : "host already used" }
mongos>
```

9. Check the sharding status from the application server.

```
 [mongodb@mdb-ms-1 ~]$ mongo  mdb-ms-1:27017/admin
MongoDB shell version: 3.0.7
connecting to: mdb-ms-1:27017/admin
mongos>
mongos> sh.status();
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "minCompatibleVersion" : 5,
        "currentVersion" : 6,
        "clusterId" : ObjectId("565315e55e022cd500e768d5")
}
  shards:
        {  "_id" : "shard1",  "host" : "shard1/mdb-srv-1:27018,mdb-srv-10:27018,mdb-srv-11:27018"
}
  balancer:
        Currently enabled:  yes
        Currently running:  no
        Failed balancer rounds in last 5 attempts:  2
        Last reported error:  could not get status from server mdb-cfg-3:27019 in cluster mdb-
cfg-3:27019 to check time :: caused by :: 10276 DBClientBase::findN: transport error: mdb-cfg-
3:27019 ns: admin.$cmd query: { serverStatus: 1 }
        Time of Reported error:  Mon Nov 23 2015 09:43:07 GMT-0500 (EST)
        Migration Results for the last 24 hours:
                No recent migrations
  databases:
        {  "_id" : "admin",  "partitioned" : false,  "primary" : "config" }

mongos>
```

10. Add a new shard into an existing sharding cluster.

Note: You can build another shard as the mongodb user by using the same steps that you used for shard1. In the following example, `root` is the user.

a. Check and run the mongod process in the `mdb-srv-2` sharding member.

```
[root@mdb-srv-2 ~]# groupadd -g 8000 mongodb
[root@mdb-srv-2 ~]# useradd -u 8000 -g 8000 -d /mongodb mongodb
[root@mdb-srv-2 ~]# chown -R mongodb:mongodb /mongodb
[root@mdb-srv-2 ~]# mkdir -p /mongodb/data
[root@mdb-srv-2 ~]# mkdir -p /data/db
[root@mdb-srv-2 ~]# chown -R mongodb:mongodb /mongodb
[root@mdb-srv-2 ~]# chown -R mongodb:mongodb /data
[root@mdb-srv-2 ~]# chmod -R 700 /mongodb
[root@mdb-srv-2 ~]# chown -R mongodb:mongodb /mongodb
```

```
[root@mdb-srv-2 ~]# mongod --replSet shard2/mdb-srv-20,mdb-srv-21 --journal --rest --shardsvr --
fork --logpath /mongodb/data/mongod.log
2015-11-23T10:57:29.617-0500 I CONTROL  ** WARNING: --rest is specified without --httpinterface,
2015-11-23T10:57:29.617-0500 I CONTROL  **          enabling http interface
about to fork child process, waiting until server is ready for connections.
forked process: 12186
child process started successfully, parent exiting
[root@mdb-srv-2 ~]#
[root@mdb-srv-2 ~]# netstat -lntp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:27018           0.0.0.0:*               LISTEN      12186/mongod
tcp        0      0 0.0.0.0:28018           0.0.0.0:*               LISTEN      12186/mongod
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      1038/sshd
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN      1170/master
tcp6       0      0 :::22                   :::*                    LISTEN      1038/sshd
tcp6       0      0 ::1:25                  :::*                    LISTEN      1170/master
[root@mdb-srv-2 ~]#
```

b. Check and run the mongod process in the `mdb-srv-20` sharding member.

```
[root@mdb-srv-20 ~]# groupadd -g 8000 mongodb
[root@mdb-srv-20 ~]# useradd -u 8000 -g 8000 -d /mongodb mongodb
[root@mdb-srv-20 ~]#
[root@mdb-srv-20 ~]# chown -R mongodb:mongodb /mongodb
[root@mdb-srv-20 ~]# mkdir -p /mongodb/data
[root@mdb-srv-20 ~]# mkdir -p /data/db
[root@mdb-srv-20 ~]# chown -R mongodb:mongodb /mongodb
[root@mdb-srv-20 ~]# chown -R mongodb:mongodb /data
[root@mdb-srv-20 ~]# chmod -R 700 /mongodb
[root@mdb-srv-20 ~]# chown -R mongodb:mongodb /mongodb
[root@mdb-srv-20 ~]# mongod --replSet shard2/mdb-srv-2,mdb-srv-21 --journal --rest --shardsvr --
fork --logpath /mongodb/data/mongod.log
2015-11-23T10:58:52.341-0500 I CONTROL  ** WARNING: --rest is specified without --httpinterface,
2015-11-23T10:58:52.341-0500 I CONTROL  **          enabling http interface
about to fork child process, waiting until server is ready for connections.
forked process: 12018
child process started successfully, parent exiting
[root@mdb-srv-20 ~]# netstat -lntp | grep 27018
tcp        0      0 0.0.0.0:27018           0.0.0.0:*               LISTEN      12018/mongod
[root@mdb-srv-20 ~]#
```

c Check and run the mongod process in the `mdb-srv-21` sharding member.

```
[root@mdb-srv-21 ~]# groupadd -g 8000 mongodb
[root@mdb-srv-21 ~]# useradd -u 8000 -g 8000 -d /mongodb mongodb
[root@mdb-srv-21 ~]# chown -R mongodb:mongodb /mongodb
[root@mdb-srv-21 ~]# mkdir -p /mongodb/data
[root@mdb-srv-21 ~]# mkdir -p /data/db
[root@mdb-srv-21 ~]# chown -R mongodb:mongodb /mongodb
[root@mdb-srv-21 ~]# chown -R mongodb:mongodb /data
[root@mdb-srv-21 ~]# chmod -R 700 /mongodb
[root@mdb-srv-21 ~]# chown -R mongodb:mongodb /mongodb
[root@mdb-srv-21 ~]# mongod --replSet shard2/mdb-srv-2,mdb-srv-20 --journal --rest --shardsvr --
fork --logpath /mongodb/data/mongod.log
2015-11-23T11:02:05.693-0500 I CONTROL  ** WARNING: --rest is specified without --httpinterface,
2015-11-23T11:02:05.693-0500 I CONTROL  **          enabling http interface
about to fork child process, waiting until server is ready for connections.
forked process: 13787
child process started successfully, parent exiting
[root@mdb-srv-21 ~]#
```

d. Connect to the shard primary server from the application server to initiate the replica set for the second shard.

```
[mongodb@mdb-ms-1 ~]$ mongo mdb-srv-2:27018/admin
MongoDB shell version: 3.0.7
connecting to: mdb-srv-2:27018/admin
Server has startup warnings:
2015-11-23T10:57:29.617-0500 I CONTROL  ** WARNING: --rest is specified without --httpinterface,
2015-11-23T10:57:29.617-0500 I CONTROL  **          enabling http interface
2015-11-23T10:57:29.661-0500 I STORAGE  [initandlisten]
2015-11-23T10:57:29.661-0500 I STORAGE  [initandlisten] ** WARNING: Readahead for /data/db is set
to 4096KB
2015-11-23T10:57:29.661-0500 I STORAGE  [initandlisten] **          We suggest setting it to
256KB (512 sectors) or less
2015-11-23T10:57:29.661-0500 I STORAGE  [initandlisten] **
http://dochub.mongodb.org/core/readahead
2015-11-23T10:57:29.716-0500 I CONTROL  [initandlisten] ** WARNING: You are running this process
as the root user, which is not recommended.
2015-11-23T10:57:29.716-0500 I CONTROL  [initandlisten]
> cfg = {
... _id : "shard2",
... members : [
...    {_id : 0, host : "mdb-srv-2:27018"},
...    {_id : 1, host : "mdb-srv-20:27018"},
...    {_id : 2, host : "mdb-srv-21:27018"}
... ]
... }
{
        "_id" : "shard2",
        "members" : [
                {
                        "_id" : 0,
                        "host" : "mdb-srv-2:27018"
                },
                {
                        "_id" : 1,
                        "host" : "mdb-srv-20:27018"
                },
                {
                        "_id" : 2,
                        "host" : "mdb-srv-21:27018"
                }
        ]
}
> rs.initiate(cfg);
{ "ok" : 1 }
shard2:SECONDARY>
 exit
bye
[mongodb@mdb-ms-1 ~]$
```

e. Build the second shard and check the shard status from the MongoDB router or application server.

```
[mongodb@mdb-ms-1 ~]$ mongo  mdb-ms-1:27017/admin
MongoDB shell version: 3.0.7
connecting to: mdb-ms-1:27017/admin
mongos> db.adminCommand( { addShard : "shard2/mdb-srv-2:27018,mdb-srv-20:27018,mdb-srv-21:27018"
} )
{ "shardAdded" : "shard2", "ok" : 1 }
mongos> sh.status();
--- Sharding Status ---
  sharding version: {
        "_id" : 1,

        "minCompatibleVersion" : 5,
        "currentVersion" : 6,
        "clusterId" : ObjectId("565315e55e022cd500e768d5")
}
  shards:
        {  "_id" : "shard1",  "host" : "shard1/mdb-srv-1:27018,mdb-srv-10:27018,mdb-srv-11:27018"
}
        {  "_id" : "shard2",  "host" : "shard2/mdb-srv-2:27018,mdb-srv-20:27018,mdb-srv-21:27018"
}
  balancer:
        Currently enabled:  yes
        Currently running:  no
        Failed balancer rounds in last 5 attempts:  5
        Last reported error:  could not get status from server mdb-cfg-1:27019 in cluster mdb-
cfg-1:27019 to check time :: caused by :: 10276 DBClientBase::find N: transport error: mdb-cfg-
1:27019 ns: admin.$cmd query: { serverStatus: 1 }
        Time of Reported error:  Mon Nov 23 2015 10:24:51 GMT-0500 (EST)
        Migration Results for the last 24 hours:
                No recent migrations
  databases:
        {  "_id" : "admin",  "partitioned" : false,  "primary" : "config" }

mongos>
```

11. Check the MongoDB sharding cluster status by using the `sh.status()` command from the MongoDB application server.

```
 [mongodb@mdb-ms-1 ~]$  mongo  mdb-ms-1:27017/admin
MongoDB shell version: 3.0.7
connecting to: mdb-ms-1:27017/admin
mongos> sh.status();
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "minCompatibleVersion" : 5,
        "currentVersion" : 6,
        "clusterId" : ObjectId("565315e55e022cd500e768d5")
}
  shards:
        {  "_id" : "shard1",  "host" : "shard1/mdb-srv-1:27018,mdb-srv-10:27018,mdb-srv-11:27018"
}
        {  "_id" : "shard2",  "host" : "shard2/mdb-srv-2:27018,mdb-srv-20:27018,mdb-srv-21:27018"
}
  balancer:
        Currently enabled:  yes
        Currently running:  no
        Failed balancer rounds in last 5 attempts:  0
        Migration Results for the last 24 hours:
                No recent migrations
  databases:
        {  "_id" : "admin",  "partitioned" : false,  "primary" : "config" }
        {  "_id" : "newbgdb",  "partitioned" : false,  "primary" : "shard1" }
        {  "_id" : "test",  "partitioned" : false,  "primary" : "shard2" }
        {  "_id" : "newdb",  "partitioned" : false,  "primary" : "shard1" }
        {  "_id" : "ndb",  "partitioned" : false,  "primary" : "shard2" }
        {  "_id" : "mdb",  "partitioned" : false,  "primary" : "shard1" }
        {  "_id" : "ldb",  "partitioned" : false,  "primary" : "shard2" }
        {  "_id" : "newbulkdb",  "partitioned" : false,  "primary" : "shard1" }
        {  "_id" : "bulkdb2",  "partitioned" : false,  "primary" : "shard2" }
        {  "_id" : "posts",  "partitioned" : false,  "primary" : "shard1" }
        {  "_id" : "db1",  "partitioned" : false,  "primary" : "shard2" }
        {  "_id" : "db2",  "partitioned" : false,  "primary" : "shard1" }
        {  "_id" : "db3",  "partitioned" : false,  "primary" : "shard1" }
        {  "_id" : "db4",  "partitioned" : false,  "primary" : "shard2" }
        {  "_id" : "db5",  "partitioned" : false,  "primary" : "shard2" }
        {  "_id" : "db6",  "partitioned" : false,  "primary" : "shard2" }
        {  "_id" : "db7",  "partitioned" : false,  "primary" : "shard1" }
        {  "_id" : "db8",  "partitioned" : false,  "primary" : "shard1" }

mongos>
```

12. Enable the sharding for database and collections

```
mongos> db.runCommand({enableSharding: "db8"})
{
        "ok" : 0,
        "errmsg" : "enableSharding may only be run against the admin database.",
        "code" : 13
}
mongos> use admin
switched to db admin
mongos> db.runCommand({enableSharding: "db8"})
{ "ok" : 1 }
```

```
mongos> sh.status();
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "minCompatibleVersion" : 5,
        "currentVersion" : 6,
        "clusterId" : ObjectId("565315e55e022cd500e768d5")
}
  shards:
        {  "_id" : "shard1",  "host" : "shard1/mdb-srv-1:27018,mdb-srv-10:27018,mdb-srv-11:27018"
}
        {  "_id" : "shard2",  "host" : "shard2/mdb-srv-2:27018,mdb-srv-20:27018,mdb-srv-21:27018"
}
  balancer:
        Currently enabled:  yes
        Currently running:  no
        Failed balancer rounds in last 5 attempts:  5
        Last reported error:  clock skew of the cluster mdb-cfg-1:27019,mdb-cfg-2:27019,mdb-cfg-
3:27019 is too far out of bounds to
allow distributed locking.
        Time of Reported error:  Wed Feb 24 2016 08:55:11 GMT-0500 (EST)
        Migration Results for the last 24 hours:
                No recent migrations
  databases:
        {  "_id" : "admin",  "partitioned" : false,  "primary" : "config" }
        {  "_id" : "newbgdb",  "partitioned" : false,  "primary" : "shard1" }
        {  "_id" : "test",  "partitioned" : false,  "primary" : "shard2" }
        {  "_id" : "newdb",  "partitioned" : false,  "primary" : "shard1" }
        {  "_id" : "ndb",  "partitioned" : false,  "primary" : "shard2" }
        {  "_id" : "mdb",  "partitioned" : false,  "primary" : "shard1" }
        {  "_id" : "ldb",  "partitioned" : false,  "primary" : "shard2" }
        {  "_id" : "newbulkdb",  "partitioned" : false,  "primary" : "shard1" }
        {  "_id" : "bulkdb2",  "partitioned" : false,  "primary" : "shard2" }
        {  "_id" : "posts",  "partitioned" : false,  "primary" : "shard1" }
        {  "_id" : "db1",  "partitioned" : false,  "primary" : "shard2" }
        {  "_id" : "db2",  "partitioned" : false,  "primary" : "shard1" }
        {  "_id" : "db3",  "partitioned" : false,  "primary" : "shard1" }
        {  "_id" : "db5",  "partitioned" : false,  "primary" : "shard2" }
        {  "_id" : "db6",  "partitioned" : false,  "primary" : "shard2" }
        {  "_id" : "db7",  "partitioned" : false,  "primary" : "shard1" }
        {  "_id" : "db8",  "partitioned" : true,  "primary" : "shard1" }

mongos> db.runCommand({enableSharding: "db1"})
{ "ok" : 1 }
mongos> db.runCommand({enableSharding: "db2"})
{ "ok" : 1 }
mongos> db.runCommand({enableSharding: "db3"})
{ "ok" : 1 }
mongos> db.runCommand({enableSharding: "db4"})
{ "ok" : 1 }
mongos> db.runCommand({enableSharding: "db5"})
{ "ok" : 1 }
mongos> db.runCommand({enableSharding: "db6"})
{ "ok" : 1 }
mongos> db.runCommand({enableSharding: "db7"})
{ "ok" : 1 }
mongos> db.runCommand({enableSharding: "db8"})
{ "ok" : 0, "errmsg" : "already enabled" }
mongos> sh.status();
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "minCompatibleVersion" : 5,
        "currentVersion" : 6,
        "clusterId" : ObjectId("565315e55e022cd500e768d5")
}
  shards:
        {  "_id" : "shard1",  "host" : "shard1/mdb-srv-1:27018,mdb-srv-10:27018,mdb-srv-11:27018"
}
        {  "_id" : "shard2",  "host" : "shard2/mdb-srv-2:27018,mdb-srv-20:27018,mdb-srv-21:27018"
}
```

```
  balancer:
        Currently enabled:  yes
        Currently running:  no
        Failed balancer rounds in last 5 attempts:  5
        Last reported error:  clock skew of the cluster mdb-cfg-1:27019,mdb-cfg-2:27019,mdb-cfg-
3:27019 is too far out of bounds to
allow distributed locking.
        Time of Reported error:  Wed Feb 24 2016 08:55:51 GMT-0500 (EST)
        Migration Results for the last 24 hours:
                No recent migrations
  databases:
        {  "_id" : "admin",  "partitioned" : false,  "primary" : "config" }
        {  "_id" : "newbgdb",  "partitioned" : false,  "primary" : "shard1" }
        {  "_id" : "test",  "partitioned" : false,  "primary" : "shard2" }
        {  "_id" : "newdb",  "partitioned" : false,  "primary" : "shard1" }
        {  "_id" : "ndb",  "partitioned" : false,  "primary" : "shard2" }
        {  "_id" : "mdb",  "partitioned" : false,  "primary" : "shard1" }
        {  "_id" : "ldb",  "partitioned" : false,  "primary" : "shard2" }
        {  "_id" : "newbulkdb",  "partitioned" : false,  "primary" : "shard1" }
        {  "_id" : "bulkdb2",  "partitioned" : false,  "primary" : "shard2" }
        {  "_id" : "posts",  "partitioned" : false,  "primary" : "shard1" }
        {  "_id" : "db1",  "partitioned" : true,  "primary" : "shard2" }
        {  "_id" : "db2",  "partitioned" : true,  "primary" : "shard1" }
        {  "_id" : "db3",  "partitioned" : true,  "primary" : "shard1" }
        {  "_id" : "db4",  "partitioned" : true,  "primary" : "shard2" }
        {  "_id" : "db5",  "partitioned" : true,  "primary" : "shard2" }
        {  "_id" : "db6",  "partitioned" : true,  "primary" : "shard2" }
        {  "_id" : "db7",  "partitioned" : true,  "primary" : "shard1" }
        {  "_id" : "db8",  "partitioned" : true,  "primary" : "shard1" }

mongos>
```

For an example in this document, we enable sharding to c7 collection with key as “_id”, you can do the same procedure to all the collections.

Important Note

In addition to database sharding, we need to enable collection sharding using “db.runcommand ({ shardCollection: <dbname>.<collectionname> , key : { <shardkey> } }) command. As a best practice, it’s not recommended to use the incremental field as the shardkey.

```
mongos> use admin
switched to db admin
mongos> db.runCommand( { shardCollection: "db7.c7", key: { _id : 1 } } )

{ "collectionsharded" : "db7.c7", "ok" : 1 }
mongos>
mongos> sh.status();
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "minCompatibleVersion" : 5,
        "currentVersion" : 6,
        "clusterId" : ObjectId("565315e55e022cd500e768d5")
}
  shards:
        {  "_id" : "shard1",  "host" : "shard1/mdb-srv-1:27018,mdb-srv-10:27018,mdb-srv-11:27018"
}
        {  "_id" : "shard2",  "host" : "shard2/mdb-srv-2:27018,mdb-srv-20:27018,mdb-srv-21:27018"
}
  balancer:
        Currently enabled:  yes
        Currently running:  yes
```

```
                Balancer lock taken at Wed Mar 16 2016 03:33:25 GMT-0400 (EDT) by mdb-ms-
1:27017:1456308560:1804289383:Balancer:846930886
        Collections with active migrations:
                db7.c7 started at Wed Mar 16 2016 03:33:25 GMT-0400 (EDT)
        Failed balancer rounds in last 5 attempts:  0
        Migration Results for the last 24 hours:
                1 : Success
  databases:
        {  "_id" : "admin",  "partitioned" : false,  "primary" : "config" }
        {  "_id" : "newbgdb",  "partitioned" : true,  "primary" : "shard1" }
        {  "_id" : "test",  "partitioned" : true,  "primary" : "shard2" }
        {  "_id" : "newdb",  "partitioned" : true,  "primary" : "shard1" }
        {  "_id" : "ndb",  "partitioned" : true,  "primary" : "shard2" }
        {  "_id" : "mdb",  "partitioned" : true,  "primary" : "shard1" }
        {  "_id" : "ldb",  "partitioned" : true,  "primary" : "shard2" }
        {  "_id" : "newbulkdb",  "partitioned" : true,  "primary" : "shard1" }
        {  "_id" : "bulkdb2",  "partitioned" : true,  "primary" : "shard2" }
        {  "_id" : "posts",  "partitioned" : true,  "primary" : "shard1" }
        {  "_id" : "db1",  "partitioned" : true,  "primary" : "shard2" }
        {  "_id" : "db2",  "partitioned" : true,  "primary" : "shard1" }
        {  "_id" : "db3",  "partitioned" : true,  "primary" : "shard1" }
        {  "_id" : "db4",  "partitioned" : true,  "primary" : "shard2" }
        {  "_id" : "db5",  "partitioned" : true,  "primary" : "shard2" }
        {  "_id" : "db6",  "partitioned" : true,  "primary" : "shard2" }
        {  "_id" : "db7",  "partitioned" : true,  "primary" : "shard1" }
                db7.c7
                        shard key: { "_id" : 1 }
                        chunks:
                                shard1  16122
                                shard2  1
                        too many chunks to print, use verbose if you want to force print
        {  "_id" : "netbgdb",  "partitioned" : true,  "primary" : "shard2" }

mongos>
```

Note: At this point, the database is already loaded, and the data is spread across the sharding cluster (shard1 and shard2).

Acknowledgements

The authors would like to thank the following people for their contributions to this report:

- Sheena Badani, Director, Technology and VAR Partners, MongoDB
- Nilesh Bagad, Senior Product Manager, Big Data Analytics, NetApp
- Scott Lane, Senior Manager, Workload Engineering, NetApp
- Chris Lemmons, Director, DSG Technical Marketing for Business Processing Workloads, NetApp

References

The following documents and resources are mentioned in or related to this report:

- Clustered Data ONTAP 8.3 Cluster Peering Express Guide
  https://library.netapp.com/ecm/ecm_download_file/ECMP1547469
- Introduction to MongoDB
  https://docs.mongodb.org/manual/core/introduction/
- MongoDB NetApp partner webpage
  https://www.mongodb.com/partners/netapp
- mongoperf
  https://docs.mongodb.org/manual/reference/program/mongoperf/

- NetApp AFF
  http://www.netapp.com/us/products/storage-systems/all-flash-fas/
- NetApp ONTAP Cloud for Amazon Web Services
  http://www.netapp.com/us/media/ds-3618.pdf
- NetApp Support
  http://mysupport.netapp.com/
- OnCommand Cloud Manager 1.0 Installation and Setup Guide
  https://library.netapp.com/ecm/ecm_download_file/ECMP1651524
- YCSB

  https://github.com/brianfrankcooper/YCSB/wiki
- Production Cluster Architecture
  https://docs.mongodb.org/manual/core/sharded-cluster-architectures-production/
- Snap Creator Framework
  http://www.netapp.com/us/products/management-software/snapcreator-framework.aspx
- The MongoDB 3.2 Manual
  https://docs.mongodb.org/manual
- TR-4391: NetApp Data Fabric with FlexPod and Cisco Intercloud Fabric
  http://www.netapp.com/us/media/tr-4391.pdf
- WiredTiger Storage Engine
  https://docs.mongodb.org/manual/core/wiredtiger/
- WP-7218: NetApp Data Fabric Architecture Fundamentals: Building a Data Fabric Today
  http://www.netapponcloud.com/hubfs/Data-Fabric/datafabric-wp.pdf

Refer to the Interoperability Matrix Tool (IMT) on the NetApp Support site to validate that the exact product and feature versions described in this document are supported for your specific environment. The NetApp IMT defines the product components and versions that can be used to construct configurations that are supported by NetApp. Specific results depend on each customer's installation in accordance with published specifications.