

Replication

Objectives

By the end of this session, you should be able to:

- ☐ Understand Cluster Concepts and Administration.
- ☐ Replication
- ☐ Client and Server Connections
- ☐ Deploying Clusters
- ☐ Cluster Operations
- ☐ Failover
- ☐ Rollback

Introduction

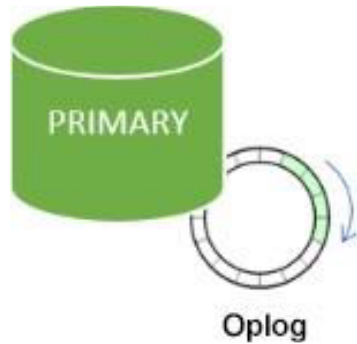
- ❑ Predicting failures and disasters is an impossible task, as it is not possible to guess the exact time when they will strike. Therefore, the business strategy should focus on solutions for these, by allocating redundant hardware and software resources.
- ❑ In the case of MongoDB, the solution to high availability and disaster recovery is to deploy MongoDB clusters instead of a single-server database.
- ❑ MongoDB doesn't require expensive hardware to build high-availability clusters, and they are relatively easy to deploy. This is where replication comes in handy.

High-Availability Clusters

- ☐ Cluster Nodes
- ☐ Share-Nothing
- ☐ Cluster Names
- ☐ Replica Sets

Cluster

- ❑ Primary-Secondary
- ❑ The Oplog



```
db.oplog.rs.stats().maxSize
```

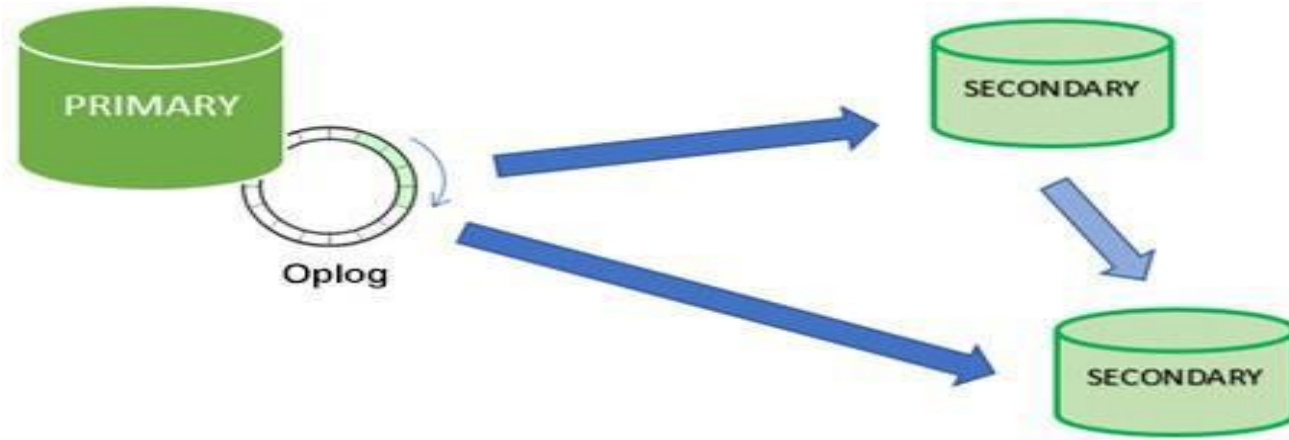
The following JS script will print the size of the Oplog in megabytes:

- ❑ use local
- ❑

```
var opl = db.oplog.rs.stats().maxSize/1024/1024 print("Oplog size: " +  
~~opl + " MB")
```

Replication Architecture

- ❑ The following diagram depicts the architecture diagram of a simple replica set cluster with only three server nodes – one primary node and two secondary nodes:



Cluster Members

- ❑ In Atlas, you can see the cluster member list from the `clusters` page .
- ❑ Click on the cluster name `cluster0` from `SANDBOX`. Then the list of servers and their roles will be displayed in the Atlas application

Cluster Members

- ❑ In Atlas, you can see the cluster member list from the **clusters** page, as shown in the following screenshot:

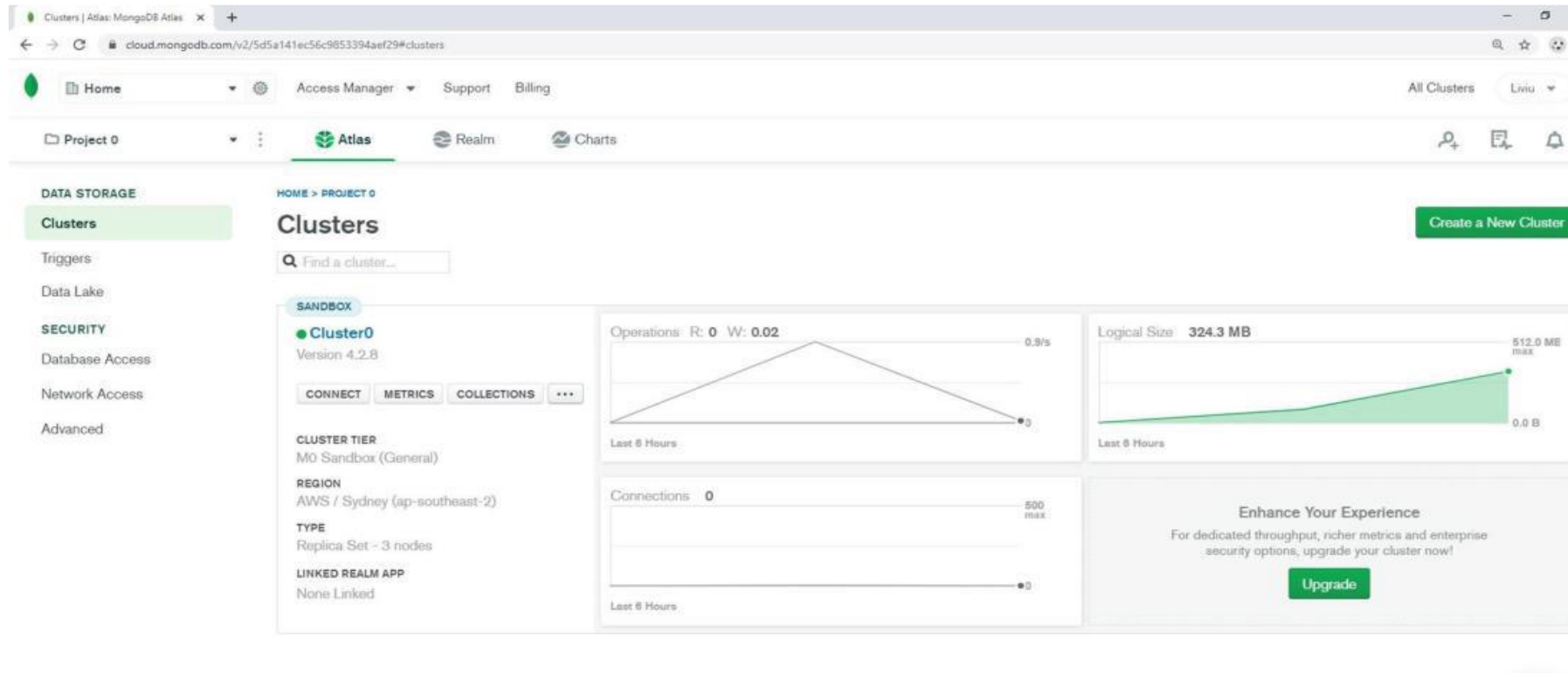


Figure : Atlas web interface

Exercise 10.01: Checking Atlas Cluster Members

- ❑ In this exercise, you will connect to the Atlas cluster using mongo shell and identify the cluster name and all cluster members, together with their current state. Use JavaScript to list the cluster members:

1. Connect to your Atlas database using mongo shell:

```
mongo "mongodb+srv://cluster0.u7n6b.mongodb.net/test" --username admindb
```

2. The replica set status function `rs.status()` gives detailed information about the cluster that is not visible from the Atlas web interface. A simple JS script to list all nodes and their member roles for `rs.status` is as follows:

```
var rs_srv = rs.status().members for (i=0; i<rs_srv.length; i++) {  
print (rs_srv[i].name, ' - ', rs_srv[i].stateStr) }
```

Output after running JS script

```
MongoDB Enterprise atlas-rzhbg7-shard-0:PRIMARY>  
MongoDB Enterprise atlas-rzhbg7-shard-0:PRIMARY> for (i=0; i<rs_srv.length; i++) {  
...   print (rs_srv[i].name, ' - ', rs_srv[i].stateStr)  
... }  
cluster0-shard-00-00.u7n6b.mongodb.net:27017 - SECONDARY  
cluster0-shard-00-01.u7n6b.mongodb.net:27017 - SECONDARY  
cluster0-shard-00-02.u7n6b.mongodb.net:27017 - PRIMARY  
MongoDB Enterprise atlas-rzhbg7-shard-0:PRIMARY>
```


Client Connections

- ❑ MongoDB supports only plain arrays and no other types of collections, However, you may want your array to contain unique elements only. MongoDB provides a way to do that by using the **\$addToSet** operator.

Connecting to a Replica Set

- ❑ In general, the same rules apply for the MongoDB connection string.

```
"mongodb+srv://cluster0.<id#>.mongodb.net/<db_name>"
```

```
connecting to: mongodb://cluster0-shard-00-00.u7n6b.mongodb.
net:27017,cluster0-shard-00-01.u7n6b.mongodb.net:27017,cluster0-shard-00-
02.u7n6b.mongodb.net:27017/test?authSource=admin&compressors=dis
abled&gssapiServiceName=mongodb&replicaSet=atlas-rzhbg7-shard-0&ssl=true
```

The first thing to notice is that the second string is significantly longer than the first. That is because the original connection string is substituted (after a successful DNS SRV lookup) into the equivalent string with the **mongodb://** URI prefix.

Client Connections (contd.)

- ❑ Following a successful connection and user authentication, the shell prompt will have the following format:

```
MongoDB Enterprise atlas-rzhbg7-shard-0:PRIMARY>
```

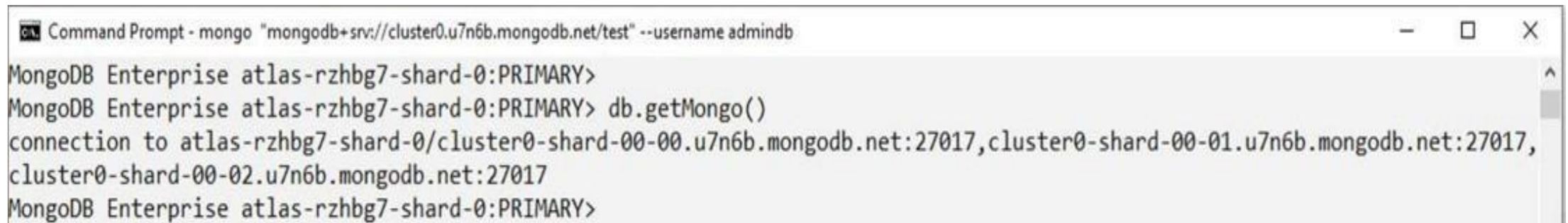
- ❑ The connection shows the MongoDB cluster, in the following form:

```
replicaset/server1:port1, server2:port2, server3:port3...
```

- ❑ To verify the current connection from mongo shell, use the following function:

```
db.getMongo()
```

This results in the following output:

A screenshot of a Windows Command Prompt window. The title bar reads "Command Prompt - mongo 'mongodb+srv://cluster0.u7n6b.mongodb.net/test' --username admin". The window contains the following text:

```
MongoDB Enterprise atlas-rzhbg7-shard-0:PRIMARY>  
MongoDB Enterprise atlas-rzhbg7-shard-0:PRIMARY> db.getMongo()  
connection to atlas-rzhbg7-shard-0/cluster0-shard-00-00.u7n6b.mongodb.net:27017,cluster0-shard-00-01.u7n6b.mongodb.net:27017,  
cluster0-shard-00-02.u7n6b.mongodb.net:27017  
MongoDB Enterprise atlas-rzhbg7-shard-0:PRIMARY>
```

Single-Server Connections

- ❑ In the same way we connect to a non-clustered MongoDB database, we have the option to connect to individual cluster members separately.

```
mongo "mongodb://cluster0-shard-00-00.u7n6b.mongodb.net:27017/ test?authSource=admin&ssl=true" --  
username admindb
```



```
Command Prompt - mongo "mongodb://cluster0-shard-00-00.u7n6b.mongodb.net:27017/test?authSource=admin&ssl=true" --username admindb  
C:\>mongo "mongodb://cluster0-shard-00-00.u7n6b.mongodb.net:27017/test?authSource=admin&ssl=true" --username admindb  
MongoDB shell version v4.4.0  
Enter password:  
connecting to: mongodb://cluster0-shard-00-00.u7n6b.mongodb.net:27017/test?authSource=admin&compressors=disabled&gssapiServiceName=mongodb&ssl=true  
Implicit session: session { "id" : UUID("972385b0-ff46-4251-abe6-e768e448a606") }  
MongoDB server version: 4.2.8  
WARNING: shell and server versions do not match  
MongoDB Enterprise atlas-rzhbg7-shard-0:SECONDARY>  
MongoDB Enterprise atlas-rzhbg7-shard-0:SECONDARY> db.getMongo()  
connection to cluster0-shard-00-00.u7n6b.mongodb.net:27017  
MongoDB Enterprise atlas-rzhbg7-shard-0:SECONDARY>
```

Figure Connecting to individual cluster members

Exercise 10.02: Checking the Cluster Replication

- ❑ In this exercise, you will connect to the Atlas cluster database using mongo shell and observe the data replication between the primary and secondary cluster nodes:
- ❑ 1. Connect to your Atlas cluster with mongo shell and user **admin**:

```
mongo "mongodb+srv://cluster0.u7n6b.mongodb.net/test" --username
admin
```

- ❑ 2. Execute the following script to create a new collection on the primary node and insert a few new documents with random numbers:

```
use sample_mflix db.createCollection("new_collection")
for (i=0; i<=100; i++) {
db.new_collection.insert({_id:i, "value":Math.random() })
}
```

Exercise 10.02: Checking the Cluster Replication (contd..)

- ❑ 3. Connect to a secondary node by entering the following code:

```
mongo "mongodb://cluster0-shard-00-00.u7n6b.mongodb.net:27017/ test?authSource=admin&ssl=true" --username admindb
```

- ❑ Query the collection to see whether data is replicated on the secondary nodes. To enable the reading of data on the secondary nodes, run the following command:

```
rs.slaveOk()
```



```
Command Prompt - mongo "mongodb://cluster0-shard-00-00.u7n6b.mongodb.net:27017/test?authSource=admin&ssl=true" --username admindb
MongoDB Enterprise atlas-rzhbg7-shard-0:SECONDARY> rs.slaveOk()
MongoDB Enterprise atlas-rzhbg7-shard-0:SECONDARY> use sample_mflix
switched to db sample_mflix
MongoDB Enterprise atlas-rzhbg7-shard-0:SECONDARY> db.new_collection.count()
101
MongoDB Enterprise atlas-rzhbg7-shard-0:SECONDARY> db.new_collection.findOne()
{ "_id" : 0, "value" : 0.02208506645285291 }
MongoDB Enterprise atlas-rzhbg7-shard-0:SECONDARY>
```

Figure : Reading data on the secondary nodes

Read Preference

<code>null</code>	This is the default for database sessions if the read preference is not set. All operations are performed on a local connection.
<code>primary</code>	All operations are performed on <code>primary</code> . This option is similar to the default <code>null</code> option if you are connected to a cluster or primary node. Please note that the <code>primary</code> read preference option can only be used if the cluster has a majority and has already elected a primary node. Otherwise, all nodes will remain in the secondary role.
<code>primaryPreferred</code>	Using this option will redirect reads to secondary nodes, but only if the primary node is not available.
<code>secondary</code>	All read operations are redirected to secondary nodes in this case.
<code>secondaryPreferred</code>	All read operations are redirected to secondary nodes. However, if no secondary node is available, then the queries are run on the primary node.
<code>nearest</code>	Reads are directed to a member with the least network latency (reply time in milliseconds), irrespective of their state as primary or secondary nodes.

Figure : Read preferences in MongoDB

Write Concern

- ❑ By default, a Mongo client receives a confirmation for each write operation (insert/ update/delete) on the primary node. The confirmation return code can be used in applications to make sure that data is securely written into the database.
- ❑ In the case of replica set clusters, though, the situation is more complex. For example, it is possible to insert rows in a primary instance, but if the primary node crashes before replication Oplog records are applied to secondary nodes, then there is a risk of data loss.
- ❑ Write concern addresses this issue by ensuring that the write is confirmed on multiple cluster nodes. Therefore, in the event of an unexpected crash of the primary node, the inserted data will not be lost.
- ❑ By default, the write concern is `{w: 1}`, which indicates acknowledgment from the primary instance only. `{w: 2}` will require confirmation from two nodes for each write operation.
- ❑ The following is an example of write concern at the statement level:

```
db.new_collection.insert({"_id":1, "info": "test writes"}, {w:2})
```


Deploying Clusters

- ❑ Deploying MongoDB clusters requires more technical and operational knowledge than installing a single server database. Planning and preparation are essential and should never be overlooked before cluster deployments.
- ❑ That is because users need to carefully plan the cluster architecture, the underlying infrastructure, and database security to provide the best performance and availability for their database.
- ❑ Regarding the method used for MongoDB replica set cluster deployments, there are a few tools that can help with the automatization and management of the deployments. The most common method is manual deployment.
- ❑ Nevertheless, the manual method is probably the most laborious option—especially for complex clusters. Automatization tools are available from MongoDB and other third-party software providers

Atlas Deployment

- ❑ Deploying MongoDB clusters on the Atlas cloud is the easiest option available for developers as it saves on effort and money.
- ❑ The MongoDB company manages the infrastructure, including the server hardware, OS, network, and `mongod` instances.
- ❑ As a result, users can focus on application development and DevOps, rather than spending time on the infrastructure. In many cases, this is the perfect solution for fast-delivery projects.
- ❑ Deploying a cluster on Atlas requires nothing more than a few clicks in the Atlas web application.
- ❑ Atlas offers more cluster options for larger deployments, which are charged services. If required, Atlas clusters can scale up easily—both vertically (adding server resources) and horizontally (adding more members). It is possible to build multi- region, replica set clusters on dedicated Atlas servers M10 and higher.
- ❑ Therefore, high availability can extend across geographical regions, between Europe and North America. This option is ideal for allocating read-only secondary nodes in a remote data center.

Manual Deployment

- ❑ Manual deployment is the most common form of MongoDB cluster deployment. For many developers, building a MongoDB cluster manually is also the preferred option for database installation because this method gives them full control over the infrastructure and cluster configuration. Manual deployment is more laborious compared with other methods, however, which makes this method less scalable for large environments.
- ❑ You would perform the following steps to manually deploy MongoDB clusters:
 - ❑ 1. Choose the server members of the new cluster. Whether they are physical servers or virtual, they must meet the minimum requirements for the MongoDB database. Also, all cluster members should have identical hardware and software specifications (CPU, memory, disk, and OS).
 - ❑ 2. MongoDB binaries must be installed on each server. Use the same installation path on all servers.
 - ❑ 3. Run one `mongod` instance per server. Servers should be on separate hardware with a separate power supply and network connections. For testing, however, it is possible to deploy all cluster members on a single physical server.

Manual Deployment (cont..)

- ❑ Start the Mongo server with the `--bind_ip` parameter. By default, `mongod` binds only to the localhost IP address (`127.0.0.1`). In order to communicate with other cluster members, `mongod` must bind to external private or public IP addresses.
- ❑ 5. Set the network properly. Each server must be able to communicate freely with other members without firewalls. Also, servers' IPs and DNS names must match in the DNS domain configuration.
- ❑ 6. Create the directory structure for database files and database instance logs. Use the same path on all servers. For example, use `/data/db` for database files (WiredTiger storage) and `/var/log/mongod` for log files on Unix/macOS systems, and in the case of Windows OSes, use `C:\data\db` directories for datafiles and `C:\log\mongo` for log files. Directories must be empty (create a new database cluster).
- ❑ 7. Start up the `mongod` instance on each server with the replica set parameter `replicaSet`.
- ❑ 8. Connect to the new cluster with mongo shell `mongo mongodb://hostname1.domain/cluster0`
- ❑ 9. Create the cluster config JSON document and save it in a JS variable (`cfg`)
- ❑ 10. Activate the cluster `rs.initiate(cfg)`

Exercise 10.03: Building Your Own MongoDB Cluster

- ❑ 1. Create the file directories.
- ❑ 2. Start the `mongod` instances from Windows Command Prompt. Use `start` to run the `mongod` startup command.
- ❑ 3. Check the startup messages in the log destination folder (`C:\data\log`).

```
16.613+1000 I NETWORK [initandlisten] waiting for connections on port 27001
```

- ❑ 4. In a separate terminal (or Windows Command Prompt), connect to the cluster using mongo shell using the following command:

```
mongo mongodb://localhost:27001/replicaSet=my_cluster
```

- ❑ 5. Edit the cluster configuration JSON document (in the JS variable `cfg`)
- ❑ 6. Activate the cluster configuration as follows: `rs.initiate(cfg)`
- ❑ 7. Verify the cluster configuration. `rs.status()`

Run the following command to verify the current cluster configuration:

```
rs.conf()
```

Enterprise Deployment

- ❑ For large-scale enterprise applications, MongoDB provides integrated tools for managing deployments. It is easy to imagine why deploying and managing hundreds of MongoDB cluster servers could be an incredibly challenging task. Therefore, the ability to manage all deployments in an integrated interface is essential for large, enterprise-scale MongoDB environments.

MongoDB provides two different interfaces:

- ❑ • **MongoDB OPS Manager** is a package available for MongoDB Enterprise Advanced. It typically requires installation on-premises.
- ❑ • **MongoDB Cloud Manager** is a cloud-hosted service to manage MongoDB Enterprise deployments.
- ❑ Both applications provide similar functionality for enterprise users, with integrated automation for deployments, advanced graphical monitoring, and backup management. Using Cloud Manager, administrators are able to deploy all types of MongoDB servers (both single and clusters), while maintaining full control over the underlying infrastructure.

Cluster Operations

- ❑ MongoDB cluster operations refer to such day-to-day administration tasks that are necessary for cluster maintenance and monitoring.
- ❑ This is especially important for clusters deployed manually, where users must fully manage and operate replica set clusters.
- ❑ In the case of the Atlas DBaaS managed service, the only interaction is through the Atlas web application and most of the work is done behind the scenes by MongoDB. Therefore, our discussion will be limited to MongoDB clusters deployed manually, either in the local infrastructure or in cloud IaaS (Infrastructure as a Service).

Cluster Operations(cont..)

Adding and Removing Members

Before we add a new member to an existing replica set, though, we need to decide on the type of cluster member. The following options are available for this:

Member Type	Description
SECONDARY	Add a new member to a replica set as a secondary member. This is the most common scenario.
READ ONLY	READ ONLY is a SECONDARY database that is never elected as the primary database. The main purpose of a READ ONLY instance is to report client query routing, using the read preference secondary.
HIDDEN	Similar to READ ONLY, but with one difference. Hidden instances do not participate in the normal client read preference. They are designed for isolated applications (such as data warehousing) that do not need to interfere with normal client activity.
DELAY	Sometimes, reporting applications need an older snapshot of the database (a delayed copy; 12 hours for example), which is continuously updated from the primary. A DELAY instance must also be hidden.
ARBITER	ARBITER is a special type of member that has no user data. Therefore, arbiter nodes do not replicate Oplog operations from the primary and they never become the primary. The purpose of an arbiter is to help during the election to create a majority of cluster nodes.
PRIMARY	While it is possible to add a new primary member (added with higher priority that will initiate election), this is NOT recommended because the new instance needs time to synchronize datafiles from the former primary. Always add new members as secondary first and switch over later when they are in sync.

Adding and Removing a Member

Adding a Member

- ❑ There are a few arguments that can be passed when we add a new cluster member, depending on the member type. In its simplest form, the **add** command has only one parameter—a string containing the hostname and port of the new instance:

```
rs.add ( "node4.domain.com:27004" )
```

Removing a Member

- ❑ Cluster members can be removed by connecting to the cluster and running the following command:

```
rs.remove({ <hostname.com> })
```

Reconfiguring a Cluster

- ❑ Cluster reconfiguration may be necessary if you want to make more complex changes to a replica set, such as adding multiple nodes in one step or editing the default values for votes and priority. Clusters can be reconfigured by running the following command:

```
rs.reconfig()
```


Failover

In certain situations, the MongoDB cluster could initiate an election process. In data center operations terminology, these types of events are usually called **Failover** and **Switchover**:

- ❑ **Failover** is always a result of an incident. When one or more cluster members become unavailable (usually because of a failure or network outage) the cluster fails over. The replica set detects that some of the nodes become unavailable, and the replica set election is automatically started.
- ❑ **Switchover** is a user-initiated process (that is, initiated by a server command). The purpose of switchover is to perform planned maintenance on the cluster. For example, the server running the primary member needs to restart for OS patching, and the administrator switches the primary over to another cluster member.

Rollback

- ❑ The cluster forms a new majority, and the activity will continue with a new primary. When the former primary is back up, it needs to roll back those (previously un-replicated) transactions before it can get in sync with the new primary.
- ❑ The chances of rollback could be reduced by setting write concern to **majority** (**w: 'majority'**)—that is, by obtaining acknowledgment from most cluster nodes (the majority) for every database write. On the downside, this could slow down the writes for the application.
- ❑ Normally, failures and outages are remedied quickly, and the affected nodes rejoin the cluster when they are back up. However, if the outage is taking a long time (for example, a week), then the secondary instances could become **stale**.
- ❑ A stale instance will not be able to resynchronize data with the primary member after a restart. In that case, the instance should be added as a new member (empty data directory) or from a recent database backup.

Switchover (Stepdown)

- ❑ For maintenance activities, we often need to transfer the primary state from one instance to another. For this, the user admin command to be executed on the primary is as follows:

```
rs.stepDown()
```

- ❑ You can verify the current master node by running the following command:

```
rs.isMaster()
```

- ❑ Note that in order for a switchover to be successful, the target cluster member must be configured with a higher priority. A member with a default priority (**priority = 0**) will never become a primary.

Exercise 10.04: Performing Database Maintenance

- ❑ In this exercise, you will perform cluster maintenance on a primary node.
- ❑ 1. Start up all cluster members (if not already started), connect with mongo shell, and verify the configuration and the current master node with `rs.isMaster().primary`.
- ❑ 2. Reconfigure the cluster. For this, copy the existing cluster configuration into a variable, `sw_over`, and set the read-only member priority. For `inst3`, the priority should be set to 0 (read-only).

```
var sw_over = rs.conf()  
sw_over.member[2].priority = 0  
rs.reconfig(sw_over)
```

- ❑ 3. Switch over to `inst2`. On the primary node, run the `stepDown` command as follows:

```
rs.stepDown()
```

- ❑ 4. Verify that the new primary is `inst2` by using the following command

```
rs.isMaster().primary
```

- ❑ 5. Shut down the instance locally using the following command: `db.shutdownServer()`

Activity 10.01: Testing a Disaster Recovery Procedure for a MongoDB Database

- ❑ 1. Configure a **sale-cluster** cluster with three members:
 - sale-prod: Primary**
 - sale-dr: Secondary**
 - sale-ab: Arbiter** (third location)
- ❑ 2. Insert test data records into the primary collection.
- ❑ 3. Simulate a disaster. Reboot the primary node (that is, kill the current **mongod** primary instance).
- ❑ 4. Perform testing on DR by inserting a few documents.
- ❑ 5. Shut down the DR instance.
- ❑ 6. Restart all nodes for the main office.
- ❑ 7. After 10 minutes, start up the DR instance.
- ❑ 8. Observe the rollback of inserted test records and re-sync with the primary.

Summary

- ❑ In this session, you learned about:
 - MongoDB replica sets are essential for providing high availability and load sharing in a MongoDB database environment.
 - While Atlas transparently provides support for infrastructure and software (including for replica set cluster management), not all MongoDB clusters are deployed in Atlas.
- ❑ In this chapter, we discussed the concepts and operations of replica set clusters. Learning about simple concepts for clusters, such as read preference, can help developers build more reliable, high-performance applications in the cloud.
- ❑ . In the next chapter, you will learn about backup and restore operations in MongoDB.