A Comprehensive Guide to MongoDB to MongoDB Migration

MongoDB is a popular NoSQL database known for its scalability and flexibility. As your applications grow, you might find the need to migrate data from one MongoDB instance to another. This could be due to various reasons, such as upgrading MongoDB versions, moving to a new infrastructure, or consolidating databases. In this comprehensive guide, we will explore the steps and strategies for performing a MongoDB to MongoDB migration successfully.

1. Understanding the Migration Types

Before we dive into the migration process, it's essential to understand the two primary types of MongoDB migrations:

1.1. Full Dump and Restore

Full Dump and Restore is a straightforward and commonly used method for migrating MongoDB data. It involves the following steps:

1. **Backup**: First, you use the mongodump utility to create a full backup of your source MongoDB database. This tool generates BSON (Binary JSON) dumps of your data, indexes, and schema.
2. **Transfer**: Next, you transfer the backup files to the target MongoDB environment. This can be done via a secure file transfer protocol like SCP or by copying the files to a shared storage location accessible to both the source and target systems.
3. **Restore**: On the target MongoDB server, you use the mongorestore utility to restore the data from the backup files. This recreates the data, indexes, and schema in the target database.

1.1.1. Advantages of Full Dump and Restore

- It's a reliable method for one-time migrations.
- It's relatively simple and can be easily automated.
- It provides a clean, identical copy of the source data in the target database.

1.1.2. Drawbacks of Full Dump and Restore

- It can be time-consuming and resource-intensive, especially for large databases.
- There may be downtime during the restore process, impacting the availability of your application.

1.2. Incremental Migration

Incremental Migration is a more complex but often more practical approach, especially for large databases with ongoing updates. It involves synchronizing changes made in the source database with the target database as they occur. Here's how it works:

1. **Initial Sync**: Start by creating an initial copy of your source database on the target server using tools like mongodump and mongorestore as in the full dump and restore method.
2. **Replica Set**: Set up the source and target MongoDB instances as a replica set. In this configuration, the target server acts as a secondary replica member to the primary source replica.
3. **Oplog Tail**: MongoDB maintains an oplog (short for "operation log") that records all write operations performed on the primary server. The target server continuously tails the oplog of the source server, applying changes in near real-time.
4. **Data Propagation**: As new data is inserted, updated, or deleted in the source database, these changes are captured in the oplog and replicated to the target database. This process ensures that the target database remains synchronized with the source.

1.2.1. Advantages of Incremental Migration

- Minimal downtime: Since data is continuously synced, there's often no need for extended downtime during migration.
- Efficient for large databases: Ideal for databases with substantial data volumes and frequent updates.
- Low impact on source: The source database can continue to operate without significant disruption.

1.2.2. Drawbacks of Incremental Migration

- Complexity: Setting up and maintaining a replica set can be more complex than a full dump and restore.
- Compatibility: Ensuring the compatibility of the source and target MongoDB versions is crucial for this method.

Choosing between full dump and restore or incremental migration depends on your specific use case and requirements. Full dump and restore is simpler and suits one-time migrations, while incremental migration is more suitable for minimizing downtime and maintaining data continuity during migration.

2. Preparation Steps

Let's explore the preparation steps for a MongoDB to MongoDB migration in more detail. Adequate preparation is crucial for ensuring a smooth and successful migration process.

1. **Backup Your Data:** Before attempting any migration, ensure you have a recent and reliable backup of your source database. Use MongoDB's built-in backup tools or third-party solutions for this purpose.

2. **Plan Your Migration:** Create a detailed migration plan, including timelines, resource allocation, and rollback strategies. Document the entire process to ensure a smooth transition.
3. **Ensure Compatibility:** Ensure that the source and target MongoDB versions are compatible. Incompatible versions may require additional data transformations during the migration process.

By thoroughly preparing for your MongoDB migration, you reduce the likelihood of unexpected issues and minimize the risk of data loss or downtime. A well-executed migration plan can help ensure a seamless transition to your new MongoDB environment.

3. Strategies to perform the Migration

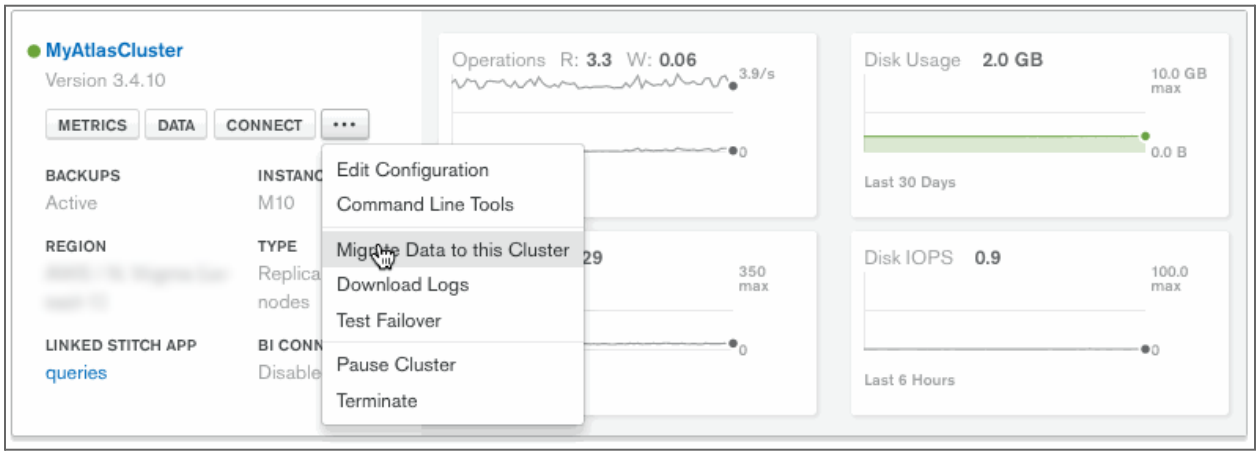
We are covering today three migration strategies (listed below) that offer different approaches to migrating data in MongoDB based on your specific requirements, including the scale of your data, downtime tolerance, and the source/target MongoDB deployment types. Choose the one that aligns best with your migration needs and resources.

	Live Migration	MongoMirror	Dump & Restore
Sharded cluster migrations	✅	❌	❌
Replica set migrations	✅	✅	✅
Standalone to replica set migrations	❌	❌	✅
Sharded cluster to replica set migrations	❌	❌	✅
Write operations can be performed during migration	✅	✅	❌
Cutover time / Downtime	Very short	Very short	Long
Does not require open inbound ports to the source mongod / mongos servers	❌	✅	✅
Merge multiple replica sets	❌	✅	✅

Below we will briefly detail about each of the three mentioned strategies.

3.1. Atlas Live Migration

Atlas Live Migrate is a feature provided by Atlas for easy data migration. It is available through the Atlas dashboard under the ellipse (...) on the cluster overview page by clicking "Migrate Data to this Cluster."



Live Migration feature transfers the data between source and destination cluster using the public internet however it encrypts the data using TLS 1.2 when transferring data. It is the only officially supported option for sharded clusters.

Reasons why Live Migration might not be suitable:

- While Live Migration infrastructure[1] is geographically distributed, the live migration servers may not be in the same data center as the target cluster.
    - There may be regulatory requirements that make this problematic.
    - Increased latency means Live Migration cannot keep up with the write workload on the source or makes the migration time unacceptably long.
- The source deployment is a standalone.
- If the MongoDB instance is inside a private subnet, and it is not possible to convert it to a public subnet.
- The source and destination topology are different.
- The Live Migration requires switching from the source to target within a 72 hour window after data is synchronized. Extending the window is simple but can only be accomplished in 24 hour windows, which may be problematic over weekends.
- Live Migration cannot utilize a VPC peering connection; if using a VPC peering connection for the migration is required, use mongomirror instead.
- Live migrate migrates the entire data in a cluster and does not provide flexibility to migrate specific databases.

[1] https://docs.atlas.mongodb.com/import/live-import/#destination-cluster-configuration

3.2. Mongomirror

Mongomirror allows copying data from a source to a target like the target is a hidden secondary. The mongomirror process can run for extended periods of time and is designed to keep the source and destination clusters in-sync.

As mentioned in the previous Atlas Live Migration section, there are situations where mongomirror will provide lower latency and/or allow for overcoming other limitations of Live Migration. Mongomirror also allows for more control over the migration process, troubleshooting issues, restarting, etc.

The major reason to use MongoMirror would be to transfer the data between source cluster and destination cluster within the internal network using either Vpc peering / private endpoint allowing for greater security and meeting any compliance requirements which need to be followed while migrating data.

3.2.1. Placement of Mongomirror

MongoMirror could be deployed in the following fashion:

1. Running Mongomirror on a dedicated server.
2. Running Mongomirror on the machine where the secondary is running.

Mongomirror is CPU and network-bound, meaning that limiting either resource risks extending time for migration. There must be enough CPU and network resources so mongomirror can keep the source and destination in-sync after the initial data copy completes.

If cost is a concerning factor then the second option could be chosen; however it is recommended to run Mongomirror in a separate machine.

3.3. Mongodump & mongorestore

mongodump is a utility that reads data from a MongoDB database and creates BSON files into a single directory. Once the backup is created, it can be restored using mongorestore which is a program that loads data from either a binary database dump created by mongodump or standard input into a MongoDB database.

Using mongodump & mongorestore requires more downtime (for writes) as compared to Live Migration/Mongomirror, as you would need to ensure that your applications are not writing to the source database while the mongodump is in progress, and not want to resume writes to the destination until the mongorestore is complete on the destination. Hence, to minimise the downtime associated with mongodump/restore, it is recommended to use the --oplog option provided with mongodump utility. When used with `--oplog` option, mongodump creates a file named `oplog.bson` as part of the output. This file contains oplog entries that occur during the mongodump operation. To restore to a specific point-in-time backup, use the output created with this option in conjunction with `mongorestore --oplogReplay`.

It should be noted that this approach can minimize the downtime, but can't eliminate it completely. You'll still have to stop writes to your database during the restore process, as any change occurred during the process will not be stored. Your data would need to be re-sharded(in case of sharded cluster) on the destination, which could require varying amounts of effort depending on the complexity of the data/sharding requirements. This approach would also need to ensure that the MongoDB user in question can `mongodump` all of the data within all namespaces on the cluster itself when pointed at a given mongos in the source deployment.

4. Data Verification and Validation

Ensuring Data Integrity

Data integrity refers to the accuracy, consistency, and reliability of your data throughout the migration process. To ensure data integrity:

1. **Data Consistency Checks:** Use MongoDB's built-in validation rules or custom scripts to check data consistency before and after migration; Verify that data constraints, such as unique indexes or schema validations, are maintained in the target database.
2. **Hashing or Checksums:** Compute hashes or checksums of critical data elements in both the source and target databases; Compare these hashes to ensure data consistency.
3. **Data Sampling:** Sample a subset of data records and compare them between the source and target databases to identify discrepancies.
4. Error Logging: Implement comprehensive error logging during the migration process. Log any issues or errors encountered for later analysis and resolution.

Comparing Source and Target Data

Comparing data between the source and target MongoDB databases is a fundamental step in data validation. Here's how you can do it:

- **Document Counts:** Verify that the total number of documents in collections matches between the source and target databases.
- **Document Sampling:** Select a random sample of documents from key collections and compare them between the source and target databases; Pay close attention to documents with complex structures or nested arrays.
- **Field-Level Comparison:** Compare individual fields within documents to ensure that data values match; Identify any missing fields or extra fields in documents.
- **Index Verification:** Confirm that indexes in the target database match those in the source database; Ensure that indexed fields are consistent.

- **Data Types:** Verify that data types are consistent between the source and target databases. MongoDB is flexible with data types, but mismatches can cause issues.
- **Data Transformation:** If data required transformation during migration (e.g., data type changes, field renaming), verify that the transformation was applied correctly.
- **Referential Integrity:** If your database relies on relationships between documents (e.g., foreign keys), ensure that these relationships are maintained in the target database.
- **Application Testing:** Perform comprehensive testing of your application with the new target MongoDB database to identify any functional issues that might have arisen during migration.

By thoroughly validating your data after migration, you can ensure that your application continues to operate correctly and that there are no hidden data issues that might cause problems down the line. This step is critical to maintaining data consistency and reliability in your new MongoDB environment.

5. Post-Migration Steps

Post-migration steps are essential to ensure the stability, performance, and security of your MongoDB database after completing the migration. These steps help you transition smoothly to your new MongoDB environment.

5.1. Updating Connection Strings

One of the first tasks after migration is updating the connection strings used by your applications and services to point to the new MongoDB environment. This ensures that they are connected to the correct database instance.

a. Application Configuration: Review and update application configuration files to reflect the new MongoDB connection details, including hostname, port, and authentication credentials.

b. DNS and Load Balancers: If you're using DNS aliases or load balancers, ensure they are correctly configured to direct traffic to the new MongoDB server.

5.2. Testing and Validation

Once your applications are connected to the new MongoDB environment, conduct thorough testing to verify that everything is functioning correctly. This should include:

- **Functionality Testing:** Ensure that all application features and functionality work as expected.
- **Performance Testing:** Evaluate the performance of your applications under real-world conditions.
- **Data Validation:** Re-run data validation checks to ensure that data integrity is maintained in the new environment.

5.6. Rollback Plan

Although not an ideal scenario, it's crucial to have a rollback plan in case severe issues arise post-migration. This plan should include detailed steps for reverting to the previous MongoDB environment.

By following these post-migration steps, you can ensure the long-term stability, performance, and security of your MongoDB database in the new environment. Continuously monitor and maintain your database to address any evolving needs or challenges that may arise over time.

6. Common Challenges and Troubleshooting

During migration, you may encounter challenges such as:

- **Network Issues:** Address network latency and bandwidth limitations, especially when migrating large datasets over long distances.
- **Data Type Compatibility:** Ensure that data types in the source and target databases match or can be appropriately converted to prevent data loss or corruption.
- **Version Incompatibilities:** Be aware of version-specific issues and seek guidance from MongoDB's official documentation or support channels.

7. Conclusion

Migrating data between MongoDB instances is a critical operation that requires careful planning and execution. By following the steps outlined in this guide, you can ensure a successful MongoDB to MongoDB migration, whether you opt for a full dump and restore or an incremental migration strategy. Remember that thorough testing and validation are key to a seamless transition and minimal disruption to your applications.