Back Up and Restore with MongoDB Tools

**NOTE**

**MongoDB Atlas**

MongoDB Atlas, the official MongoDB cloud service, provides 2 fully-managed methods for backups:

1. Continuous Backups, which take incremental backups of data in your cluster, ensuring your backups are typically just a few seconds behind the operational system. Atlas continuous backups allow you to restore from stored snapshots or from a selected point in time within the last 24 hours. You can also query a continuous backup snapshot.
2. Cloud Provider Snapshots, which provide localized backup storage using the native snapshot functionality of the cluster's cloud service provider.

Considerations

**Deployments**

The mongodump and mongorestore utilities work with BSON data dumps, and are useful for creating backups of small deployments. For resilient and non-disruptive backups, use a file system or block-level disk snapshot function, such as the methods described in the MongoDB Backup Methods document.

**NOTE**

mongodump and mongorestore **cannot** be part of a backup strategy for 4.2+ sharded clusters that have sharded transactions in progress, as backups created with mongodump *do not maintain* the atomicity guarantees of transactions across shards.

For 4.2+ sharded clusters with in-progress sharded transactions, use one of the following coordinated backup and restore processes which *do maintain* the atomicity guarantees of transactions across shards:

* MongoDB Atlas,
* MongoDB Cloud Manager, or
* MongoDB Ops Manager.

**Performance Considerations**

Because mongodump and mongorestore operate by interacting with a running mongod instance, they can impact the performance of your running database. Not only do the tools create traffic for a running database instance, they also force the database to read all data through memory. When MongoDB reads infrequently used data, it can evict more frequently accessed data, causing a deterioration in performance for the database's regular workload.

When backing up your data with MongoDB's tools, consider the following guidelines:

* Label files so that you can identify the contents of the backup as well as the point in time that the backup reflects.
* Use an alternative backup strategy such as Filesystem Snapshots or MongoDB Cloud Manager if the performance impact of mongodump and mongorestore is unacceptable for your use case.
* Use --oplog to capture incoming write operations during the {{out\_tool}} operation to ensure that the backups reflect a consistent data state.
* Ensure that your backups are usable by restoring them to a test MongoDB deployment.

**TIP**

**See also:**

MongoDB Backup Methods and MongoDB Cloud Manager Backup documentation for more information on backing up MongoDB instances. Additionally, consider the following reference documentation for the MongoDB Database Tools:

* mongoexport
* mongoimport
* mongodump
* mongorestore

Binary BSON Dumps

The mongorestore and mongodump utilities work with BSON data dumps, and are useful for creating backups of small deployments. For resilient and non-disruptive backups, use a file system or block-level disk snapshot function, such as the methods described in the MongoDB Backup Methods document.

Use these tools for backups if other backup methods, such as MongoDB Cloud Manager or file system snapshots are unavailable.

Procedures

**Back Up a Database with mongodump**

**NOTE**

mongodump and mongorestore **cannot** be part of a backup strategy for 4.2+ sharded clusters that have sharded transactions in progress, as backups created with mongodump *do not maintain* the atomicity guarantees of transactions across shards.

For 4.2+ sharded clusters with in-progress sharded transactions, use one of the following coordinated backup and restore processes which *do maintain* the atomicity guarantees of transactions across shards:

* MongoDB Atlas,
* MongoDB Cloud Manager, or
* MongoDB Ops Manager.

**Exclude local Database**

mongodump excludes the content of the local database in its output.

**Required Access**

To run mongodump against a MongoDB deployment that has access control enabled, you must have privileges that grant find action for each database to back up. The built-in backup role provides the required privileges to perform backup of any and all databases.

*Changed in version 3.2.1*: The backup role provides additional privileges to back up the system.profile collection that exists when running with database profiling. Previously, users required read access on this collection.

**Basic mongodump Operations**

The mongodump utility backs up data by connecting to a running mongod.

The utility can create a backup for an entire server, database or collection, or can use a query to backup just part of a collection.

When you run mongodump without any arguments, the command connects to the MongoDB instance on the local system (e.g. localhost) on port 27017 and creates a database backup named dump/ in the current directory.

To backup data from a mongod instance running on the same machine and on the default port of 27017, use the following command:

|  |
| --- |
| mongodump |

You can also specify the --host and --port of the MongoDB instance that the mongodump should connect to. For example:

|  |
| --- |
| mongodump --host=mongodb.example.net --port=27017 |

mongodump will write BSON files that hold a copy of data accessible via the mongod listening on port 27017 of the mongodb.example.net host. See Create Backups from Non-Local mongod Instances for more information.

To specify a different output directory, you can use the --out or -o option:

|  |
| --- |
| mongodump --out=/data/backup/ |

To limit the amount of data included in the database dump, you can specify --db and --collection as options to mongodump. For example:

|  |
| --- |
| mongodump --collection=myCollection --db=test |

This operation creates a dump of the collection named myCollection from the database test in a dump/ subdirectory of the current working directory.

mongodump overwrites output files if they exist in the backup data folder. Before running the mongodump command multiple times, either ensure that you no longer need the files in the output folder (the default is the dump/ folder) or rename the folders or files.

**Point in Time Operation Using Oplogs**

Use the --oplog option with mongodump to collect the oplog entries to build a point-in-time snapshot of a database within a replica set. With --oplog, mongodump copies all the data from the source database as well as all of the oplog entries from the beginning to the end of the backup procedure. This operation, in conjunction with mongorestore --oplogReplay, allows you to restore a backup that reflects the specific moment in time that corresponds to when mongodump completed creating the dump file.

**Create Backups from Non-Local mongod Instances**

The --host and --port options for mongodump allow you to connect to and backup from a remote host. Consider the following example:

|  |
| --- |
| mongodump --host=mongodb1.example.net --port=3017 --username=user --password=**"pass"** --out=/opt/backup/mongodump-2013-10-24 |

On any mongodump command you may, as above, specify username and password credentials to specify database authentication.

**Restore a Database with mongorestore**

**NOTE**

mongodump and mongorestore **cannot** be part of a backup strategy for 4.2+ sharded clusters that have sharded transactions in progress, as backups created with mongodump *do not maintain* the atomicity guarantees of transactions across shards.

For 4.2+ sharded clusters with in-progress sharded transactions, use one of the following coordinated backup and restore processes which *do maintain* the atomicity guarantees of transactions across shards:

* MongoDB Atlas,
* MongoDB Cloud Manager, or
* MongoDB Ops Manager.

**Access Control**

To restore data to a MongoDB deployment that has access control enabled, the restore role provides the necessary privileges to restore data from backups *if* the data does not include system.profile collection data and you run mongorestore without the --oplogReplay option.

If the backup data includes system.profile collection data or you run with --oplogReplay, you need additional privileges:

|  |  |
| --- | --- |
| system.profile | If the backup data includes system.profile collection data and the target database does not contain the system.profile collection, mongorestore attempts to create the collection even though the program does not actually restore system.profile documents. As such, the user requires additional privileges to perform createCollection and convertToCapped actions on the system.profile collection for a database.  Both the built-in roles dbAdmin and dbAdminAnyDatabase provide the additional privileges. |
| --oplogReplay | To run with --oplogReplay, create a user-defined role that has anyAction on anyResource.  Grant only to users who must run mongorestore with --oplogReplay. |

**Basic mongorestore Operations**

The mongorestore utility restores a binary backup created by mongodump. By default, mongorestore looks for a database backup in the dump/ directory.

The mongorestore utility restores data by connecting to a running mongod directly.

mongorestore can restore either an entire database backup or a subset of the backup.

**NOTE**

*New in version 3.6:*

All MongoDB collections have UUIDs by default. When MongoDB restores collections, the restored collections retain their original UUIDs. When restoring a collection where no UUID was present, MongoDB generates a UUID for the restored collection.

For more information on collection UUIDs, see Collections.

To use mongorestore to connect to an active mongod, use a command with the following prototype form:

|  |
| --- |
| mongorestore --port=<port number> <path to the backup> |

Consider the following example:

|  |
| --- |
| mongorestore dump-2013-10-25/ |

Here, mongorestore imports the database backup in the dump-2013-10-25 directory to the mongod instance running on the localhost interface on the default port 27017.

**Restore Point in Time Oplog Backup**

If you created your database dump using the --oplog option to ensure a point-in-time snapshot, call mongorestore with the --oplogReplay option, as in the following example:

|  |
| --- |
| mongorestore --oplogReplay |

You may also consider using the mongorestore --objcheck option to check the integrity of objects while inserting them into the database, or you may consider the mongorestore --drop option to drop each collection from the database before restoring from backups.

**Restore Backups to Non-Local mongod Instances**

By default, mongorestore connects to a MongoDB instance running on the localhost interface and on the default port (27017). If you want to restore to a different host or port, use the --host and --port options.

The following example that specifies the --host and --port options:

|  |
| --- |
| mongorestore --host=mongodb1.example.net --port=3017 |

If restoring to an instance that enforces access control, include the --username and the --authenticationDatabase as well. Omit the --password option to have mongorestore prompt for the password:

|  |
| --- |
| mongorestore --host=mongodb1.example.net --port=3017 --username=user --authenticationDatabase=admin /opt/backup/mongodump-2013-10-24 |

←  Back Up and Restore with Filesystem Snapshots