



MongoDB Plugin

Bacula Systems Documentation

Contents

1	Why Use Bacula Enterprise with MongoDB	3
2	MongoDB Plugin: Scope	4
2.1	Supported MongoDB Server Versions	4
2.2	Supported Deployment Types	4
2.3	Supported Security Modes	4
2.4	Supported Backup Levels and Recovery Semantics	5
2.5	Use-Case Callouts	6
3	MongoDB Plugin: Features	6
3.1	Feature Summary	6
3.2	Standalone MongoDB	7
3.3	Replica Sets	7
3.4	Sharded Clusters	7
3.5	Security-Aware Backup	8
3.6	TLS and MONGODB-X509	8
4	MongoDB Plugin: Architecture	8
4.1	Backup Data Flow	9
4.2	Restore Data Flow	10
4.3	Backup Artifact Layout	10
4.4	Metadata Files	10
5	MongoDB Plugin: Installation	11
5.1	Prerequisites	11
5.2	Instructions	12
5.3	Result	12
6	MongoDB Plugin: Configuration	12
6.1	Basic Job Example	13
6.2	Plugin Configuration	13
7	MongoDB Plugin: Operations	32
7.1	Backup	32
7.2	Restore	38
7.3	List and Query	48
7.4	Sharded Balancer RunScript Hooks	49
8	MongoDB Plugin: Limitations	51
9	MongoDB Plugin: Best Practices	52
9.1	General Practices	52
9.2	Replica-Set Practices	53
9.3	Sharded-Cluster Practices	53
9.4	Security Practices	53
9.5	Restore Practices	53
10	MongoDB Plugin: Troubleshooting	53
10.1	Common Issues	54
10.2	Diagnostic Checklist	54
10.3	Useful Bacula Checks	55

Contents

Note

You can download this article as a [PDF](#)

Enterprise

Bacula Enterprise Only

This solution is **only** available for Bacula Enterprise. For subscription inquiries, please reach out to sales@baculasystems.com.

This document describes the **Bacula Enterprise MongoDB Plugin**, a dedicated Bacula Enterprise File Daemon plugin that protects MongoDB deployments using MongoDB-supported logical backup and restore tools integrated with Bacula Enterprise scheduling, cataloging, retention, storage, and restore workflows.

MongoDB is a distributed document database commonly used for application data, event records, service metadata, analytics pipelines, and high-volume operational workloads. Production MongoDB environments may be deployed as standalone servers, replica sets, or sharded clusters. They may also use authentication, TLS, client certificate authentication, and different recovery expectations depending on the business criticality of each database.

Traditional filesystem backups are not sufficient for many MongoDB deployments because live database files can change while they are being copied. The MongoDB Plugin addresses this by coordinating logical backups through `mongodump` and logical restores through `mongorestore` while adding Bacula-specific safety metadata, checksum validation, recovery-chain metadata, and operator guardrails.

1 Why Use Bacula Enterprise with MongoDB

Bacula Enterprise provides centralized policy control, cataloged recovery, long-term retention, encryption, deduplication-compatible storage targets, tape, object storage, and off-site or air-gapped protection. The MongoDB Plugin adds MongoDB-aware backup and restore behavior so backup administrators can protect MongoDB data without writing ad-hoc scripts around Bacula jobs.

The plugin is especially useful when organizations need:

- Consistent logical backups of MongoDB application databases
- Replica-set incremental and differential backup chains based on oplog capture
- Point-in-time recovery within the available oplog window for replica-set chains
- Controlled sharded-cluster backup and restore through `mongos`
- Explicit isolated disaster-recovery workflows for sensitive `admin` database data
- Authentication, TLS, and client certificate support in Bacula-managed jobs

- Clear dry-run reporting before backup or restore changes are made.

The next sections explain supported topologies, installation, Fileset configuration, operational workflows, limitations, and troubleshooting guidance.

2 MongoDB Plugin: Scope

Enterprise

Bacula Enterprise Only

This solution is **only** available for Bacula Enterprise. For subscription inquiries, please reach out to sales@baculasystems.com.

The **Bacula Enterprise MongoDB Plugin** is designed for Linux File Daemon hosts that can reach the MongoDB deployment and can run the MongoDB Database Tools, specifically `mongodump` and `mongorestore`. The plugin does not require `mongosh` for backup or restore operations. The plugin is available starting with **Bacula Enterprise 18.2.5**.

2.1 Supported MongoDB Server Versions

At the time of the plugin release, the following MongoDB server versions are supported: 6.0, 7.0, and 8.0. Future official versions are expected to be supported by the **Bacula Enterprise MongoDB Plugin**.

For more information about supported MongoDB server platforms and versions, contact Bacula Systems [Support Team](#).

Restore to the same MongoDB major version, or to a newer major version, is supported, subject to MongoDB compatibility and feature compatibility version checks. Downgrade restores to an older MongoDB major version are rejected by the plugin before restore starts.

2.2 Supported Deployment Types

The plugin supports the following deployment types:

- Standalone `mongod` instances
- Writable replica-set primaries
- Sharded clusters accessed through `mongos`.

The plugin intentionally does not support direct backup of secondaries, direct backup of shard members, or direct backup of config servers. For sharded clusters, always connect to `mongos`, the MongoDB query router. This is not `mongosh`, the interactive MongoDB shell.

2.3 Supported Security Modes

The plugin supports:

- No-authentication MongoDB deployments
- SCRAM username/password authentication
- Passwords provided directly, through a password file, or through an environment variable
- TLS server validation with a CA file

- TLS client certificate/key files
- MONGODB-X509 authentication using a client certificate identity.

2.4 Supported Backup Levels and Recovery Semantics

Bacula Full, Incremental, and Differential job levels are accepted. Their behavior depends on the source topology:

Table 1: MongoDB backup level behavior by topology

Source topology	Full	Incremental	Differential	Recovery semantics
Standalone mongod	Full logical artifact	Full-style artifact	Full-style artifact	Restore from the selected logical artifact. Point-in-time replay is not available because there is no replica-set oplog recovery chain.
Writable replica-set primary	Full logical artifact with base oplog window	Oplog-chain segment from the previous chain boundary	Oplog-chain segment from the latest Full boundary	Full plus Incremental or Differential artifacts can be restored as a validated oplog chain and replayed up to the selected recovery boundary.
Sharded cluster through mongos	Coordinated logical full artifact with routing metadata	Full-style artifact	Full-style artifact	Restore through mongos with sharded planning metadata and balancer guardrails. Sharded oplog point-in-time recovery is not supported.
Direct replica-set secondary, shard member, or config server	Not supported	Not supported	Not supported	Connect to a writable replica-set primary for replica sets, or to mongos for sharded clusters.

The entries in the matrix mean:

- **Full logical artifact:** a logical MongoDB backup produced with MongoDB Database Tools and described by the plugin manifest. It contains the selected databases, collections, indexes, and eligible security metadata.
- **Full logical artifact with base oplog window:** a Full replica-set backup that also records the oplog boundary needed to start a later recovery chain.
- **Coordinated logical full artifact with routing metadata:** a sharded-cluster logical backup taken through mongos. The plugin captures sharded planning metadata and enforces balancer guardrails, but it does not replace config server disaster recovery.
- **Full-style artifact:** an Incremental or Differential Bacula job that is accepted for schedule continuity, but is executed as a complete logical artifact for that job. It is safe to restore, but it is not an oplog delta and does not provide point-in-time replay.
- **Oplog-chain segment:** a replica-set Incremental or Differential payload containing the validated `local.oplog.rs` range since the previous recovery boundary. Restore replays these segments after the base Full artifact.

- **Recovery boundary:** the oplog timestamp where replay stops. The plugin can auto-select the chain tip or use an explicit operator-provided replay point when it falls inside the protected oplog range.

2.5 Use-Case Callouts

- Application teams needing reliable logical MongoDB recovery without scripting.
- Backup administrators who need MongoDB data in Bacula retention policies and storage pools.
- Regulated environments requiring off-site, tape, or air-gapped copies of MongoDB data.
- Replica-set deployments requiring incremental recovery chains and point-in-time replay.
- Sharded deployments where cluster routing metadata must be captured and used during restore planning.

3 MongoDB Plugin: Features

Enterprise

Bacula Enterprise Only

This solution is **only** available for Bacula Enterprise. For subscription inquiries, please reach out to sales@baculasystems.com.

The **Bacula Enterprise MongoDB Plugin** protects MongoDB using logical backup and restore operations. It relies on the official MongoDB Database Tools for the actual dump and restore payloads, and it adds Bacula-specific orchestration, metadata, validation, streaming, and recovery-chain handling.

3.1 Feature Summary

- Full logical backup and restore of selected MongoDB application databases
- Database and collection include/exclude filters
- Application-scoped user and custom role backup and restore
- Optional `admin` database capture for isolated disaster recovery
- Replica-set Full, Incremental, and Differential recovery chains using oplog segments
- Point-in-time restore for replica-set chains using `replay_to`
- Sharded-cluster logical backup and restore through `mongos`
- Sharded routing metadata capture and restore planning
- Dry-run backup and restore reports
- Checksum validation of artifact files before restore
- Manifest and chain metadata for auditability and recovery-chain validation
- TLS, SCRAM, and `MONGODB-X509` connection support
- Bacula LIST and QUERY operations for inventory discovery.

3.2 Standalone MongoDB

For standalone MongoDB servers, the plugin creates a logical backup using `mongodump`. This is suitable for many application databases where full logical restore is sufficient.

Consistency guarantee:

- Full: logical full snapshot as produced by MongoDB Database Tools
- Incremental and Differential: accepted by Bacula, but stored as full-style logical artifacts because standalone servers do not provide an oplog recovery chain.

3.3 Replica Sets

A replica set in MongoDB is a group of `mongod` processes that maintain the exact same dataset. It provides data redundancy, fault tolerance, and high availability, serving as the foundational building block for production MongoDB deployments.

A standard replica set consists of multiple MongoDB nodes, typically at least three, that work together in a primary/secondary relationship.

For replica sets, connect the plugin to a writable primary. Prefer a replica set URI or seed list (`replicaSet=...`, multiple hosts, `directConnection=false`) so the MongoDB driver can discover the topology and route operations to the current primary.

If you use `directConnection=true` or a single fixed endpoint, make sure that endpoint is the writable primary. Connecting directly to a secondary makes the plugin reject backup or restore operations that require primary-side guarantees or target writes.

A Full backup captures a logical snapshot and records enough oplog metadata to seed a recovery chain. Incremental and Differential jobs capture oplog segments.

Consistency guarantee:

- Full: logical snapshot plus base oplog window metadata
- Incremental: oplog segment since the previous chain boundary
- Differential: oplog segment since the latest Full boundary
- Restore: validated chain replay, optionally stopped at a point in time with `replay_to`.

The plugin validates chain continuity before replay. It checks artifact identity, parent linkage, source fingerprint, oplog boundaries, and boundary digests. This prevents common unsafe restore scenarios such as missing incrementals or replay after an oplog rollback/divergence.

3.4 Sharded Clusters

A MongoDB sharded cluster is a horizontal scaling architecture that distributes massive datasets and high-throughput operations across multiple servers. Instead of overloading a single machine, it breaks data into smaller, manageable partitions called “chunks” and stores them across separate database instances to ensure high availability and performance.

A sharded cluster consists of three primary components that work together to route and store information:

- Shards: individual servers or replica sets that store subsets of data. In production environments, each shard is typically a replica set.
- Config servers: replica-set members that store cluster metadata and routing configuration, such as which chunks live on which shards.
- Query routers (`mongos`): processes that receive client requests, route them to the relevant shards, and merge the results.

For sharded clusters, connect to mongos. This can be done through the `uri` parameter, for example: `uri=mongodb://backup:secret@mongos1.example.com:27017/admin?authSource=admin`

The plugin captures selected application data through the router and records cluster routing metadata used during restore planning.

Consistency guarantee:

- Full: sharded coordinated logical artifact with captured routing metadata
- Incremental and Differential: full-style logical artifacts
- Restore: sharded application restore through mongos with routing preparation where supported by the captured metadata.

The balancer must be stopped before sharded backup and restore operations. This is a deliberate guardrail: running with an active balancer can move chunks during backup or restore and make the resulting artifact harder to reason about.

3.5 Security-Aware Backup

By default, the plugin attempts to capture users and custom roles that belong to the selected application databases. This enables application restore that also recreates the database-level security model.

Use `skip_security=yes` when:

- The backup user has only data read permissions.
- User and role restore is managed manually outside Bacula.
- A migration should restore data only.

The `admin` database is not included by default. It is protected by explicit isolated disaster-recovery guardrails because restoring `admin` can affect cluster users, roles, and administrative state.

3.6 TLS and MONGODB-X509

The plugin supports TLS for both the Java driver connection and MongoDB Database Tools. When using `MONGODB-X509`, the MongoDB username must match the client certificate subject exactly as MongoDB reports it, for example:

Listing 1: **Example X.509 username**

```
O=Bacula,OU=Backup,CN=bacula-mongodb-client
```

For X.509 authentication, set `authentication_mechanism=MONGODB-X509`, `authentication_database=$external`, `tls=yes`, and `tls_certificate_key_file`. Do not provide a password.

4 MongoDB Plugin: Architecture

Enterprise

Bacula Enterprise Only

This solution is **only** available for Bacula Enterprise. For subscription inquiries, please reach out to sales@baculasystems.com.

The **Bacula Enterprise MongoDB Plugin** is a Bacula File Daemon plugin. Bacula loads the plugin through the File Daemon, and the plugin launches a Java backend that communicates with MongoDB and with the Bacula MetaPlugin transport layer.

The plugin uses two MongoDB-facing components:

- The MongoDB Java driver for connection validation, topology inspection, inventory discovery, compatibility checks, and metadata reads.
- The MongoDB Database Tools, especially `mongodump` and `mongorestore`, for logical backup and restore payloads.

The plugin stores MongoDB payloads in the Bacula catalog under the virtual path `/@mongodb`. Database archive payloads are transferred through Bacula storage, while small control files, such as manifests and chain metadata, are staged locally under the plugin working path for planning and validation. Compatible normal Bacula restores stream full dump archives directly into `mongorestore` when the restore mode supports streaming.

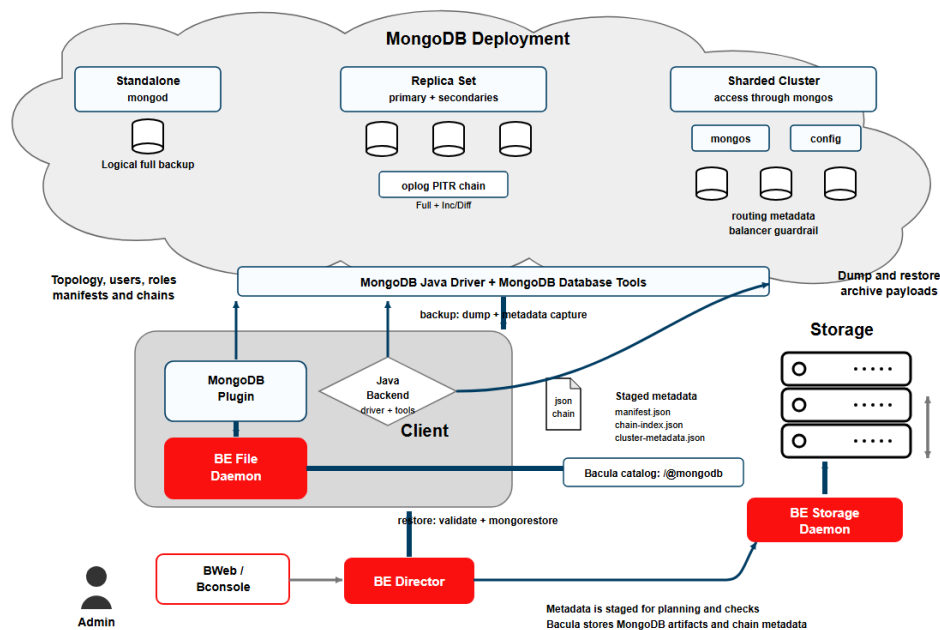


Fig. 1: Bacula Enterprise MongoDB Plugin Architecture

4.1 Backup Data Flow

1. Bacula starts a backup job using a Fileset that contains `Plugin = "mongodb: ..."`.
2. The File Daemon loads the MongoDB Plugin and passes job level, working path, and plugin parameters.

3. The plugin validates the connection, topology, selected databases, permissions, tools, and available local working space.
4. The plugin invokes `mongodump` for selected databases and collections.
5. Full database archives are sent to Bacula storage.
6. Metadata files are sent to Bacula under `/@mongodb`.
7. For replica-set Incremental or Differential jobs, the plugin captures oplog segments and chain metadata.

4.2 Restore Data Flow

1. The operator selects MongoDB files under `/@mongodb` in a Bacula restore session.
2. The plugin reads restored metadata files and validates the artifact set.
3. Compatible full archive payloads are streamed from Bacula to MongoDB restore tooling without writing complete dump archives locally.
4. Small metadata files are staged locally because restore planning requires local access to them.
5. The plugin validates target version, topology, conflicts, chain continuity, and isolated-DR guardrails.
6. The plugin invokes `mongorestore` and, when needed, replays oplog segments.

4.3 Backup Artifact Layout

A typical Bacula backup appears under `/@mongodb` with paths similar to:

Listing 2: MongoDB backup files in Bacula

```
/@mongodb/manifest.json
/@mongodb/chain-index.json
/@mongodb/cluster-metadata.json
/@mongodb/dump/appdb.archive
/@mongodb/dump/analytics.archive
/@mongodb/oplog/local/oplog.rs.bson
```

Not every file is present in every backup. For example, `chain-index.json` and `oplog/local/oplog.rs.bson` are used for replica-set recovery chains, while `cluster-metadata.json` is used for sharded-cluster artifacts.

4.4 Metadata Files

The most important metadata file is `manifest.json`. It records source version, feature compatibility information, topology, selected databases, restore profile, checksums, and file inventory.

For recovery chains, `chain-index.json` records chain identity, artifact identity, parent linkage, source fingerprint, and oplog boundaries.

For sharded clusters, `cluster-metadata.json` records shard membership and routing metadata used by restore planning.

5 MongoDB Plugin: Installation

Enterprise

Bacula Enterprise Only

This solution is **only** available for Bacula Enterprise. For subscription inquiries, please reach out to sales@baculasystems.com.

This article describes how to install the Bacula Enterprise MongoDB Plugin.

The Bacula File Daemon and the MongoDB Plugin must be installed on a Linux host that can reach the MongoDB deployment.

5.1 Prerequisites

Java Requirement

Java 11 or newer must be installed and available in the system PATH.

File Daemon Plugin Directory

The Bacula Enterprise MongoDB Plugin is distributed as a standard package for every supported operating system.

After installing the package, a few aspects must be considered: The Plugin Directory directive of the File Daemon resource in `/opt/bacula/etc/bacula-fd.conf` has to point to where the `mongodb-fd.so` plugin is installed. The usual Bacula plugin directory is `/opt/bacula/plugins`. Thus, the configuration file `bacula-fd.conf` should look like in the following example:

```
FileDaemon {
  Name = bacula-fd
  Plugin Directory = /opt/bacula/plugins
  ...
}
```

File Daemon Working Directory

The plugin working directory must have enough free space for metadata, restore planning files, oplog replay payloads, and any restore mode that cannot use direct archive streaming.

MongoDB-Specific Prerequisites

- The MongoDB Database Tools must be installed on the File Daemon host. The plugin uses `mongodump` and `mongorestore`.

Note

For most deployments, install [MongoDB Database Tools](#) from the same MongoDB family as the protected server version. This avoids tool/server compatibility problems during backup or restore. A straightforward approach is to deploy the Bacula File Daemon and MongoDB Plugin on one of the database hosts providing the service.

Note

The MongoDB Plugin requires MongoDB Database Tools on the Bacula File Daemon host, because backup and restore operations use `mongodump` and `mongorestore`. The plugin validates these commands before starting backup or restore work and fails the Bacula job with a clear error if either command is missing. The interactive shell `mongosh` is not part of this runtime requirement.

- The File Daemon host must have network access to the MongoDB port or to the `mongos` router for sharded clusters.
- The backup user must have sufficient MongoDB privileges for the selected mode.
- For TLS, the File Daemon user must be able to read the CA file and optional client certificate/key file.

5.2 Instructions

- General installation instructions with BIM (recommended)
- General installation instructions with Package Manager

5.3 Result

The installation package includes the following components:

- JAR libraries in `/opt/bacula/lib`, such as `bacula-mongodb-plugin-x.x.x.jar` and the corresponding library JAR.
- A plugin connection file, `mongodb-fd.so`, in the Bacula plugins directory, usually `/opt/bacula/plugins`.
- A backend launcher, `mongodb_backend`, in `/opt/bacula/bin`.

After installation, verify that the File Daemon loaded the plugin with a status client command in Bconsole. The `Plugin:` line must contain `mongodb`.

Listing 3: **Status client**

```
*status client=mongodb-fd
Connecting to Client mongodb-fd at mongodb.example.com:9102

mongodb-fd Version: 18.3.0 x86_64-pc-linux-gnu
Daemon started 08-May-26 10:14. Jobs: run=2 running=0 max=100.
Plugin: bpipe(2) mongodb(1.0.0)
```

6 MongoDB Plugin: Configuration

Enterprise

Bacula Enterprise Only

This solution is **only** available for Bacula Enterprise. For subscription inquiries, please reach out to sales@baculasystems.com.

This chapter explains how to configure Bacula jobs and Filesets for the MongoDB Plugin. Once the plugin is installed, MongoDB backups are configured like other Bacula Enterprise backup jobs: create a Fileset that invokes the plugin, attach that Fileset to a Job, and schedule it according to the required recovery point objective.

6.1 Basic Job Example

Listing 4: **MongoDB Job configuration**

```
Job {
    Name = "MongoDB-Backup"
    JobDefs = "DefaultJob"
    Client = mongodb-fd
    FileSet = "FS-MongoDB"
    Storage = File
    Pool = Default
    Messages = Standard
    Accurate = yes
}

FileSet {
    Name = "FS-MongoDB"
    Include {
        Options { signature = MD5 }
        Plugin = "mongodb: uri=mongodb://backup:secret@mongo1.example.com:27017/
admin?authSource=admin"
    }
}
```

The following sections document the plugin parameters and provide complete Fileset examples for common production scenarios.

6.2 Plugin Configuration

Enterprise

Bacula Enterprise Only

This solution is **only** available for Bacula Enterprise. For subscription inquiries, please reach out to sales@baculasystems.com.

A MongoDB backup Fileset invokes the plugin with a line similar to:

Listing 5: **Fileset MongoDB**

```
FileSet {
    Name = "FS-MongoDB"
```

(continues on next page)

(continued from previous page)

```
Include {  
  Options { signature = MD5 }  
  Plugin = "mongodb: uri=mongodb://backup:secret@mongo1.example.com:27017/  
→admin?authSource=admin database=appdb"  
  }  
}
```

Parameters are written as key=value pairs after `mongodb:`. Boolean parameters can be written as `yes` or `no`. Use quotes around the entire plugin line when values contain special characters.

Connection Parameters

Enterprise

Bacula Enterprise Only

This solution is **only** available for Bacula Enterprise. For subscription inquiries, please reach out to sales@baculasystems.com.

The plugin supports two connection styles:

- URI mode, using `uri`
- Structured mode, using `host` plus optional authentication, replica-set, and TLS parameters.

Use URI mode for simple configurations. Use structured mode when passwords, TLS files, or BWeb-generated forms should be kept as separate parameters.

Parameter Reference

Table 2: Connection parameters

Option	Re- quire	De- fault	Values	Example	Description
uri	Yes*		MongoDB connection string	mongodb:// mongo1. example. com:27017/ admin	Complete MongoDB URI. Do not combine with struc- tured connection options such as host, username, replica_set, or TLS file parameters.
host	Yes*		host[:port]	mongo1. example. com:27017	MongoDB host or seed host. Repeat for replica-set seeds.
username	No		String	backup	MongoDB username. Re- quired when password-based authentication is used.
password	No		String	secret	MongoDB password. Prefer password_file or password_env in production.
password_file	No		Path	/opt/bacula/ etc/mongodb. pass	File containing the MongoDB password.
password_env	No		Envi- ronment variable name	MONGODB_BACKUP	Environment variable contain- ing the MongoDB password.
authenticatic	No	admin	Database name	admin	Authentication database. Use \$external for MONGODB-X509.
authenticatic	No	Driver de- fault	SCRAM-SHA-256 SCRAM-SHA-256 MONGODB-X509	SCRAM-SHA-256	Authentication mechanism passed to the MongoDB driver and tools.
replica_set	No		Replica-set name	rs0	Expected replica-set name.
direct_connec	No	no	yes or no	yes	Force direct connection to one MongoDB node. Useful in controlled test environments.
tls	No	no	yes or no	yes	Enable TLS.
tls_ca_file	No		Path	/opt/ bacula/etc/ mongodb-ca. pem	PEM file containing trusted CA certificates.
tls_certificate	No		Path	/opt/ bacula/etc/ bacula-mongodb pem	PEM file containing client cer- tificate chain and unencrypted private key.
tls_allow_inva	No	no	yes or no	no	Allow TLS hostname mis- match. Use only in controlled labs or temporary migrations.

A connection definition is required for Bacula backup, restore, LIST, and QUERY operations. Use either

`uri` or structured connection parameters. When structured parameters are used, at least one `host` value is required.

Use only one password source: `password`, `password_file`, or `password_env`.

Connection Rules by Scenario

Use the following combinations as the baseline for each deployment type:

- **Standalone mongod (no replica set):** use URI mode (`uri=...`) or structured mode with one host. Do not set `replica_set`. Keep `direct_connection=no` unless you explicitly need direct node binding.
- **Replica set (recommended):** use URI with multiple seeds or structured mode with multiple host entries plus `replica_set=<name>`. Keep `direct_connection=no` so MongoDB driver discovery can route operations to the writable primary.
- **Replica set (forced single node):** use `direct_connection=yes` only for controlled scenarios (for example diagnostics). For backup/restore operations, this node must be writable primary, otherwise operations can fail by design.
- **Sharded cluster through mongos:** connect to mongos routers (URI or structured host values that point to mongos), do not set `replica_set`, and keep `direct_connection=no`.
- **TLS-enabled deployments:** add `tls=yes` plus trust material (typically `tls_ca_file`). Add `tls_certificate_key_file` when client certificates are required.
- **Authentication source selection:** keep `authentication_database=admin` for most users; use `$external` only for external mechanisms such as X.509.

Practical safety rules:

- Pick one mode only: either `uri` or structured parameters.
- Keep credentials out of plain filesets when possible by using `password_file` or `password_env`.
- For replica sets, provide at least two seed hosts when available, so node changes do not break connectivity.

X.509 Rules

When `authentication_mechanism=MONGODB-X509` is used:

- `tls=yes` is required.
- `tls_certificate_key_file` is required.
- `authentication_database` must be `$external`.
- Do not set `password`, `password_file`, or `password_env`.
- The username must match the client certificate subject exactly as MongoDB expects it.

Example subject:

Listing 6: X.509 username option

```
username="O=Bacula,OU=Backup,CN=bacula-mongodb-client"
```

Backup Parameters

Enterprise

Bacula Enterprise Only

This solution is **only** available for Bacula Enterprise. For subscription inquiries, please reach out to sales@baculasystems.com.

The following parameters control backup scope and behavior.

Table 3: Backup parameters

Option	Re- quired	Default	Values	Example	Description
database	No	All appli- cation databases	Database name	appdb	Application database to include. Repeat or use comma-separated values where supported.
database_excl	No		Database name	analytics_	Database to exclude after include selection.
collection	No	All col- lections	Collection name	widgets	Collection name to include across selected databases.
collection_ex	No		Collection name	temp_event	Collection to exclude after include selection.
include_admin	No	no	yes or no	yes	Include admin database. Intended only for isolated disaster-recovery artifacts.
skip_security	No	no	yes or no	yes	Skip application users and custom roles.
cleanup_on_success	No	no	yes or no	yes	Delete local artifact root after successful backup. Mainly for orchestrated runs.
dry_run	No	no	yes or no	yes	Run backup preflight only.
report_format	No	text	text or json	json	Dry-run report format.

Bacula Level Behavior

The plugin uses Bacula job metadata and local chain state to resolve the previous artifact for Incremental and Differential jobs.

For standalone and sharded deployments, Incremental and Differential jobs run as safe full-style fall-back artifacts. For replica-set primaries, they capture oplog segments and participate in a point-in-time recovery chain.

Security Capture

By default, the plugin captures selected application database users and custom roles when the MongoDB user has sufficient privileges. Use `skip_security=yes` for lower-privilege data-only backups.

Use `include_admin=yes` only when the backup is intended for isolated disaster recovery. Normal in-place restore of `admin` content is intentionally guarded because it can overwrite administrative users and roles.

Restore Parameters

Enterprise

Bacula Enterprise Only

This solution is **only** available for Bacula Enterprise. For subscription inquiries, please reach out to sales@baculasystems.com.

Note

Restore parameters are not Fileset backup parameters. They are entered during a Bacula restore session from the `Plugin Options` prompt, or through a Bacula GUI that exposes plugin restore options.

Restore parameters are entered from the Bacula restore `Plugin Options` prompt or through a Bacula GUI that exposes plugin restore options.

Table 4: Restore parameters

Option	Re- quirec	Default	Values	Example	Description
<code>conflict_str:</code>	No	fail	fail, drop, skip	drop	Behavior when target collections already exist.
<code>collection</code>	No	All col- lections	Collection name	widgets	Restore only matching collection names across selected databases.
<code>map_db</code>	No		source=targ	appdb=appd	Restore a dumped database under a different target database name.
<code>replay_to</code>	No	Chain end	seconds[:or	1778245150	Stop oplog replay at a selected point in a replica-set recovery chain.
<code>isolated_dr</code>	No	no	yes or no	yes	Enable isolated disaster-recovery guardrails.
<code>recover_admin</code>	No	no	yes or no	yes	Explicitly replay dumped <code>admin</code> database during isolated DR where supported.
<code>skip_security</code>	No	no	yes or no	yes	Skip restoring application users and custom roles.
<code>dry_run</code>	No	no	yes or no	yes	Validate restore plan without changing the target.
<code>report_format</code>	No	text	text or json	json	Dry-run report format.

Restore Conflict Strategy

`conflict_strategy` controls how the plugin handles MongoDB objects that already exist in the restore target. It is the MongoDB Plugin overwrite policy. It is not the same as the generic Bacula Replace option shown by Bconsole.

For normal standalone and replica-set application restores, conflict handling is collection-based:

- **fail:** before running `mongorestore`, the plugin checks whether any collection selected for restore already exists in the target database. If a matching target collection exists, restore is rejected before data is written. This is the safest default.
- **drop:** the plugin runs `mongorestore` with `--drop`. For each restored collection, MongoDB drops the matching target collection first and then recreates it from the backup. All documents, indexes, and collection options inside that restored target collection are replaced by the backup content. Collections that are not part of the restore scope are not dropped. The plugin does not use `--drop` for automated admin database replay.
- **skip:** the plugin removes already-existing target collections from the restore plan and restores only selected collections that do not exist in the target. Existing collections are kept exactly as they are. This is a collection-level decision, not a document-level merge.

The plugin does not compare or merge individual documents inside a collection. If a collection is restored, its content comes from the backup. If a collection is skipped, all existing documents in that target collection remain untouched.

For sharded restore artifacts, `conflict_strategy=drop` is broader. Before recreating sharded routing metadata, the plugin drops the target database named in the sharded restore plan, then prepares the sharded collections and runs `mongorestore` through `mongos`. Use this only when the target database can be replaced.

Examples:

Table 5: Conflict strategy examples

Scenario	Target state before restore	Result
<code>conflict_strategy</code>	<code>appdb.widgets</code> already exists	Restore fails before writing data. No MongoDB collection is changed.
<code>conflict_strategy</code>	<code>appdb.widgets</code> exists and is selected for restore; <code>appdb.logs</code> exists but is not selected	<code>appdb.widgets</code> is dropped and recreated from the backup. <code>appdb.logs</code> is left untouched.
<code>conflict_strategy</code>	<code>appdb.widgets</code> exists and <code>appdb.events</code> does not exist; both are selected for restore	<code>appdb.widgets</code> is skipped and left untouched. <code>appdb.events</code> is restored from the backup.
<code>conflict_strategy</code> with sharded artifact	Target sharded database <code>appdb</code> exists	The target <code>appdb</code> database is dropped before sharded routing metadata and selected data are restored through <code>mongos</code> .

Bacula Replace Option

Bconsole also shows the generic Bacula restore Replace option, with values such as `Never`, `Always`, `Ifnewer`, and `Ifolder`. That option is a filesystem restore policy used by Bacula for ordinary file restores.

For MongoDB live plugin restore with `Where=/`, the MongoDB Plugin does not currently use the Bacula Replace value to decide what happens inside MongoDB. The MongoDB target overwrite behavior is controlled by plugin restore options, mainly `conflict_strategy`.

For example, if Bconsole displays `Replace: Always` but the plugin restore options keep `conflict_strategy=fail`, an existing target collection still causes the MongoDB restore to fail. Conversely, `conflict_strategy=drop` can replace selected MongoDB collections even though the Bacula `Replace` option is only a generic restore setting.

When using `Where=/local/fd/path` for file-based artifact restore, the restore is not applied to MongoDB. The selected Bacula files are reconstructed under the chosen File Daemon path for inspection or manual use.

Isolated DR Parameters

Use `isolated_dr=yes` only for a dedicated recovery target. This allows artifacts containing `admin` to be considered for restore. For standalone and replica-set targets, `recover_admin=yes` explicitly replays the dumped `admin` database. For sharded targets, automated `admin` replay is blocked; manual cluster security recovery remains required.

Supported Restore Use Cases

Use restore parameters according to the recovery goal:

- **Standard full restore to an empty target:** keep defaults (`conflict_strategy=fail`) and restore all dumped databases/collections.
- **Overwrite existing target data:** set `conflict_strategy=drop` so existing collections are dropped before replay.
- **Merge into a partially populated target:** set `conflict_strategy=skip` to preserve existing collections and restore only missing ones.
- **Restore only selected application objects:** use `collection` to limit restore scope to matching collection names.
- **Restore into alternate database names:** use one or more `map_db` mappings (`source=target`) to avoid overwriting original names.
- **Point-in-time restore on replica-set incremental chains:** set `replay_to=seconds[:ordinal]` to stop oplog replay at a controlled boundary.
- **Isolated disaster recovery including admin payload:** set `isolated_dr=yes` and add `recover_admin=yes` only when the target architecture supports automated `admin` replay.
- **Schema and plan validation only:** set `dry_run=yes` (optionally `report_format=json` for automation pipelines).

Operational rules:

- `report_format` is evaluated only with `dry_run=yes`.
- `recover_admin=yes` requires `isolated_dr=yes` and an artifact set that includes `admin`.
- On sharded targets, keep `isolated_dr=yes` for `admin`-containing artifacts, but expect manual `admin/security` recovery.
- `collection` filters are applied per selected database in the restore plan.

Replay_To Format Details

`replay_to` uses the same format as MongoDB oplog limits:

- Format: `seconds[:ordinal]`
- `seconds`: Unix epoch time in **UTC seconds** (for example `1778245150`).

- **ordinal**: MongoDB oplog increment within that same second (the *i* component of `Timestamp(t, i)`).

Accepted values:

- **seconds** must be a non-negative integer.
- **ordinal** is optional; when omitted, it defaults to 0.
- **ordinal** must also be a non-negative integer.
- Invalid formats (extra separators, non-numeric values, negatives) are rejected before restore starts.

Meaning:

- 1778245150 means 1778245150:0.
- 1778245150:3 means replay until the oplog position `t=1778245150`, `i=3`.
- Use `:ordinal` when multiple oplog entries share the same second and you need a deterministic cut point.

Planner behavior:

- If **replay_to** is not set, restore automatically replays to the end of the selected chain.
- If **replay_to** is outside the selected Full+Incremental chain boundaries, restore is rejected.
- **replay_to** is relevant only when oplog replay is used (replica-set incremental recovery path).

Examples:

Listing 7: **Replay to exact second with default ordinal 0**

```
replay_to=1778245150
```

Listing 8: **Replay to specific oplog entry within the same second**

```
replay_to=1778245150:3
```

General Examples

Listing 9: **Overwrite existing target collections**

```
conflict_strategy=drop
```

Listing 10: **Preserve existing collections and restore only missing ones**

```
conflict_strategy=skip
```

Listing 11: **Restore only two collection names across selected databases**

```
collection="widgets,events"
```

Listing 12: **Remap databases during restore**

```
map_db="appdb=appdb_restored,logs=logs_recovered"
```

Listing 13: **Replica-set point-in-time restore with oplog cutoff**

```
replay_to=1778245150:3
```

Listing 14: **Isolated DR with explicit admin replay (supported targets)**

```
isolated_dr=yes  
recover_admin=yes
```

Listing 15: **Dry-run preflight report for CI/CD checks**

```
dry_run=yes  
report_format=json
```

Listing 16: **Combined selective restore with overwrite policy**

```
conflict_strategy=drop  
map_db="appdb=appdb_stage"  
collection="widgets"  
skip_security=yes
```

Common Parameters

Enterprise

Bacula Enterprise Only

This solution is **only** available for Bacula Enterprise. For subscription inquiries, please reach out to sales@baculasystems.com.

The following parameters control general behavior of the MongoDB Plugin. They are provided by the Bacula MetaPlugin layer and have the same purpose as in other Bacula Enterprise plugins that use this backend model.

Table 6: Common parameters

Option	Re- quire	De- fault	Values	Example	Description
abort_on_	No	no	yes or no	yes	Abort the Bacula job as soon as the plugin detects a non-recoverable error. Keep this enabled for production jobs where partial artifacts must not be accepted silently.
config_file	No		Path to a readable file	/opt/bacula/etc/mongodb.settings	Read plugin parameters from a file. This is useful for keeping long connection settings, credentials, TLS paths, and database scope outside the FileSet definition.
log	No	Bacula job log only	Writable file path	/var/log/bacula/mongodb-plugin.log	Write additional plugin debug information to a local file. Use this for troubleshooting together with the Bacula job log.
debug	No	0	0 to 9	2	Enable increasing levels of plugin debug output. Higher values are intended for support investigations and may expose operational details in logs.
path	No	Bacula working path	Existing directory	/opt/bacula/working	Directory used for small local control files, temporary metadata, restore planning files, and staged oplog replay payloads. Current normal Bacula restores stream dump/*.archive directly to mongorestore; size this path for metadata and oplog replay, not for full dump archive staging.

Configuration File Example

The `config_file` parameter accepts the same `key=value` options that can be placed directly in the Fileset. Sensitive parameters should normally be kept in a root-readable file instead of the Director configuration.

Listing 17: /opt/bacula/etc/mongodb.settings

```
host=mongo1.example.com:27017
host=mongo2.example.com:27017
replica_set=rs0
username=backup
password_file=/opt/bacula/etc/mongodb.pass
authentication_database=admin
authentication_mechanism=SCRAM-SHA-256
database=appdb
```

(continues on next page)

(continued from previous page)

```
database=analytics
```

Use `config_file` for stable site configuration and keep per-job behavior, such as backup level and restore conflict strategy, in the Bacula job or restore options.

Tuning Parameters

Enterprise

Bacula Enterprise Only

This solution is **only** available for Bacula Enterprise. For subscription inquiries, please reach out to sales@baculasystems.com.

The MongoDB Plugin is intentionally conservative by default. Most performance comes from MongoDB Database Tools and from the Bacula Storage Daemon path.

Table 7: Tuning parameters

Option	Re-quired	Default	Values	Exam-ple	Description
<code>stream_telemetry</code>	No	no	yes or no	yes	Emit stream diagnostics such as bytes, duration, and throughput in job logs.

Operational Tuning Guidance

- Start with one Bacula job per MongoDB deployment.
- Avoid overlapping MongoDB backup jobs against the same source.
- Verify Storage Daemon throughput before assuming MongoDB is the bottleneck.
- Enable `stream_telemetry=yes` during performance investigations, then disable it for normal jobs if the extra log volume is not needed.
- For sharded clusters, schedule backups during a window where the balancer can remain stopped for the full operation.

Fileset Examples

Enterprise

Bacula Enterprise Only

This solution is **only** available for Bacula Enterprise. For subscription inquiries, please reach out to sales@baculasystems.com.

This section provides practical Fileset examples for common and advanced MongoDB deployments.

Getting Started with Filesets

Use this table to choose the best example as a starting point.

Table 8: Which example should I start from?

If your environment is...	Start from this section	Why
Single MongoDB node, simple setup	Standalone with URI	Fastest baseline configuration.
Single MongoDB node, password/TLS handled outside URI	Standalone with Structured Parameters	Cleaner credential and TLS file handling.
Replica set with normal primary discovery	Replica Set with URI Seed List or Replica Set with Structured Parameters and TLS	Driver can route to writable primary automatically.
Replica set in controlled lab with one fixed node	Replica Set Direct Connection (Lab / Controlled Cases)	Explicit single-node targeting.
Sharded cluster through mongos	Sharded Cluster through mongos (URI) or Sharded Cluster through mongos (Structured Hosts)	Uses mongos entry-point instead of direct shard members.
Lower-privilege data-only backup	Data-Only Lower-Privilege Backup	Disables users/roles capture with <code>skip_security=yes</code> .
Isolated disaster-recovery artifact	Isolated Disaster-Recovery Artifact	Includes admin meta-data intentionally.
Scope tuning (db/collection include-exclude)	Backup Selection Examples	Shows realistic include/exclude combinations.
Operational diagnostics and performance investigations	Operational Debug and Telemetry	Uses debug, log, and <code>stream_telemetry</code> .
Centralized reusable configuration	Configuration File as Primary Source	Keeps long parameter sets in <code>config_file</code> .

Connection Examples

Standalone with URI

Listing 18: Standalone backup with URI mode

```
FileSet {
  Name = "FS-MongoDB-Standalone-URI"
  Include {
    Options { signature = MD5 }
    Plugin = "mongodb: uri=mongodb://backup:secret@mongo1.example.com:27017/
    ↪admin?authSource=admin database=appdb"
  }
}
```

Standalone with Structured Parameters

Listing 19: Standalone backup with password_env

```
FileSet {
  Name = "FS-MongoDB-Standalone-Structured"
  Include {
    Options { signature = MD5 }
    Plugin = "mongodb: host=mongo1.example.com:27017 username=backup_
    ↪password_env=MONGODB_BACKUP_PASSWORD authentication_database=admin_
    ↪database=appdb"
  }
}
```

Replica Set with URI Seed List

Listing 20: Replica-set backup (driver elects writable primary)

```
FileSet {
  Name = "FS-MongoDB-RS-URI"
  Include {
    Options { signature = MD5 }
    Plugin = "mongodb: uri=mongodb://mongo1.example.com:27017,mongo2.
    ↪example.com:27017,mongo3.example.com:27017/admin?replicaSet=rs0&
    ↪authSource=admin database=appdb"
  }
}
```

Replica Set with Structured Parameters and TLS

Listing 21: Replica-set backup with TLS and password file

```
FileSet {
  Name = "FS-MongoDB-RS-TLS"
  Include {
    Options { signature = MD5 }
    Plugin = "mongodb: host=mongo1.example.com:27017 host=mongo2.example.
    ↪com:27017 host=mongo3.example.com:27017 replica_set=rs0 username=backup_
    ↪password_file=/opt/bacula/etc/mongodb.pass authentication_database=admin_
    ↪authentication_mechanism=SCRAM-SHA-256 tls=yes tls_ca_file=/opt/bacula/etc/
```

(continues on next page)

(continued from previous page)

```
↪mongodb-ca.pem database=appdb"
}
}
```

Replica Set Direct Connection (Lab / Controlled Cases)

Listing 22: Direct connection to one replica-set node

```
FileSet {
  Name = "FS-MongoDB-RS-Direct"
  Include {
    Options { signature = MD5 }
    Plugin = "mongodb: host=mongo-primary.example.com:27017 replica_set=rs0
↪direct_connection=yes username=backup password_file=/opt/bacula/etc/mongodb.
↪pass authentication_database=admin database=appdb"
  }
}
```

Use `direct_connection=yes` only when you intentionally want a single target node. Normal production jobs should prefer seed-list discovery.

Sharded Cluster through mongos (URI)

Listing 23: Sharded backup through mongos URI

```
FileSet {
  Name = "FS-MongoDB-Sharded-URI"
  Include {
    Options { signature = MD5 }
    Plugin = "mongodb: uri=mongodb://backup:secret@mongos1.example.
↪com:27017/admin?authSource=admin database=appdb"
  }
}
```

Sharded Cluster through mongos (Structured Hosts)

Listing 24: Sharded backup through mongos with structured options

```
FileSet {
  Name = "FS-MongoDB-Sharded-Structured"
  Include {
    Options { signature = MD5 }
    Plugin = "mongodb: host=mongos1.example.com:27017 host=mongos2.example.
↪com:27017 username=backup password_file=/opt/bacula/etc/mongodb.pass
↪authentication_database=admin database=appdb"
  }
}
```

Before running sharded jobs, stop the balancer and confirm it is not active.

TLS with MONGODB-X509

Listing 25: X.509 backup

```
FileSet {
  Name = "FS-MongoDB-X509"
  Include {
    Options { signature = MD5 }
    Plugin = "mongodb: host=mongo1.example.com:27017 username=\"O=Bacula,
↪OU=Backup,CN=bacula-mongodb-client\" authentication_database=$external_
↪authentication_mechanism=MONGODB-X509 tls=yes tls_ca_file=/opt/bacula/etc/
↪mongodb-ca.pem tls_certificate_key_file=/opt/bacula/etc/bacula-mongodb-
↪client.pem database=appdb"
  }
}
```

The escaped quotes around username keep the X.509 subject together as a single plugin option value inside the Bacula FileSet string.

Backup Selection Examples

Multiple Included Databases

Listing 26: Include two application databases

```
FileSet {
  Name = "FS-MongoDB-Select-MultiDB"
  Include {
    Options { signature = MD5 }
    Plugin = "mongodb: uri=mongodb://backup:secret@mongo1.example.com:27017/
↪admin?authSource=admin database=appdb database=analytics"
  }
}
```

Include then Exclude Databases

Listing 27: Include appdb and analytics, then exclude analytics_tmp

```
FileSet {
  Name = "FS-MongoDB-Select-DBExclude"
  Include {
    Options { signature = MD5 }
    Plugin = "mongodb: uri=mongodb://backup:secret@mongo1.example.com:27017/
↪admin?authSource=admin database=appdb database=analytics database_
↪exclude=analytics_tmp"
  }
}
```

Collection Subset across Selected Databases

Listing 28: Capture only selected collections

```
FileSet {
  Name = "FS-MongoDB-Select-Collections"
  Include {
    Options { signature = MD5 }
    Plugin = "mongodb: uri=mongodb://backup:secret@mongo1.example.com:27017/
↪admin?authSource=admin database=appdb collection=widgets collection=orders"
  }
}
```

Include then Exclude Collections

Listing 29: Keep app data, exclude temporary collections

```
FileSet {
  Name = "FS-MongoDB-Select-CollectionExclude"
  Include {
    Options { signature = MD5 }
    Plugin = "mongodb: uri=mongodb://backup:secret@mongo1.example.com:27017/
↪admin?authSource=admin database=appdb collection=widgets collection=audits_
↪collection_exclude=audits_tmp collection_exclude=widgets_staging"
  }
}
```

Data-Only Lower-Privilege Backup

Listing 30: Skip user/role capture

```
FileSet {
  Name = "FS-MongoDB-DataOnly"
  Include {
    Options { signature = MD5 }
    Plugin = "mongodb: host=mongo1.example.com:27017 username=appbackup_
↪password_file=/opt/bacula/etc/appbackup.pass authentication_database=appdb_
↪database=appdb skip_security=yes"
  }
}
```

(continues on next page)

(continued from previous page)

```
}
```

Isolated Disaster-Recovery Artifact

Listing 31: **Include admin only for isolated DR artifacts**

```
FileSet {  
  Name = "FS-MongoDB-Isolated-DR"  
  Include {  
    Options { signature = MD5 }  
    Plugin = "mongodb: uri=mongodb://backup:secret@mongo1.example.com:27017/  
↪admin?authSource=admin include_admin=yes"  
  }  
}
```

Use this Fileset only for dedicated disaster-recovery planning. Restoring admin content into a live production target can overwrite administrative state.

Dry-Run Planning in JSON

Listing 32: **Preflight-only planning output**

```
FileSet {  
  Name = "FS-MongoDB-DryRun"  
  Include {  
    Options { signature = MD5 }  
    Plugin = "mongodb: uri=mongodb://backup:secret@mongo1.example.com:27017/  
↪admin?authSource=admin database=appdb dry_run=yes report_format=json"  
  }  
}
```

Common and Tuning Examples

Operational Debug and Telemetry

Listing 33: **Enable extra diagnostics during investigations**

```
FileSet {
  Name = "FS-MongoDB-Tuning-Debug"
  Include {
    Options { signature = MD5 }
    Plugin = "mongodb: uri=mongodb://backup:secret@mongo1.example.com:27017/
↪admin?authSource=admin database=appdb path=/opt/bacula/working log=/var/log/
↪bacula/mongodb-plugin.log debug=2 stream_telemetry=yes abort_on_error=yes"
  }
}
```

Configuration File as Primary Source

Listing 34: **Fileset line using config_file**

```
FileSet {
  Name = "FS-MongoDB-ConfigFile"
  Include {
    Options { signature = MD5 }
    Plugin = "mongodb: config_file=/opt/bacula/etc/mongodb.settings"
  }
}
```

Listing 35: **/opt/bacula/etc/mongodb.settings**

```
host=mongo1.example.com:27017
host=mongo2.example.com:27017
host=mongo3.example.com:27017
replica_set=rs0
username=backup
password_file=/opt/bacula/etc/mongodb.pass
authentication_database=admin
authentication_mechanism=SCRAM-SHA-256
tls=yes
tls_ca_file=/opt/bacula/etc/mongodb-ca.pem
database=appdb
database=analytics
collection_exclude=analytics_tmp
skip_security=no
debug=0
stream_telemetry=no
```

Configuration File plus Inline Overrides

Listing 36: **Keep base connection in file, override job behavior inline**

```
FileSet {
  Name = "FS-MongoDB-ConfigFile-Override"
  Include {
    Options { signature = MD5 }
```

(continues on next page)

(continued from previous page)

```
Plugin = "mongodb: config_file=/opt/bacula/etc/mongodb.settings_
↪ database=appdb collection=widgets dry_run=yes report_format=json"
    }
}
```

The `config_file` content can include any MongoDB plugin parameter, including connection, backup selection, restore, common, and tuning options.

7 MongoDB Plugin: Operations

Enterprise

Bacula Enterprise Only

This solution is **only** available for Bacula Enterprise. For subscription inquiries, please reach out to sales@baculasystems.com.

This chapter explains the main operational workflows for the MongoDB Plugin.

7.1 Backup

Enterprise

Bacula Enterprise Only

This solution is **only** available for Bacula Enterprise. For subscription inquiries, please reach out to sales@baculasystems.com.

Backup Procedure

A MongoDB backup job is driven by the Bacula job level and the plugin parameters in the Fileset. Bacula scheduling, job level, Fileset scope, and plugin options define the backup plan and the operation to perform.

At the start of the job, the plugin performs these checks and preparation steps:

1. Read the Bacula job level, Fileset parameters, and optional `config_file`.
2. Build the MongoDB connection from `uri` or structured connection options.
3. Validate Java backend availability, MongoDB Java Driver access, and MongoDB Database Tools such as `mongodump`.
4. Inspect deployment topology: standalone, replica set, or sharded cluster through `mongos`.
5. Read MongoDB server version, feature compatibility version, source identity, and selected databases or collections.
6. Validate authentication, TLS, backup privileges, and optional security-capture privileges.

7. Prepare the local working path for small control files such as manifests and recovery-chain metadata.
8. For sharded clusters, validate the balancer guardrail before dumping data.

After preflight succeeds, the plugin produces logical MongoDB artifacts and sends them to Bacula under the `/@mongodb` virtual path. The Bacula Storage Daemon stores those artifacts according to the configured Pool, Volume, and Storage resources.

What Is Protected

The plugin protects MongoDB data as logical artifacts:

- Selected databases and collections are dumped with `mongodump` archive output.
- Collection metadata, indexes, and options are included in the dump archive generated by MongoDB Database Tools.
- `manifest.json` records source topology, MongoDB version, selected scope, checksums, dump files, and plugin decisions.
- Replica-set recovery-chain backups include `chain-index.json` and oplog BSON segments.
- Sharded-cluster backups include cluster metadata used for restore planning and operator validation.
- Users and custom roles are captured unless `skip_security=yes` is configured.

The plugin does not perform filesystem-level copies of MongoDB data files. The backup is a logical MongoDB backup designed for restore through MongoDB tools and driver operations.

If filesystem-level copies of MongoDB configuration files or data files are required, protect them with a separate standard File Daemon job. Keep MongoDB Plugin jobs focused on logical database data.

Backup File Structure

A typical Bacula backup contains files similar to:

Listing 37: MongoDB files protected by Bacula

```
/@mongodb/manifest.json
/@mongodb/control/stream-plan/appdb.json
/@mongodb/dump/appdb.archive
/@mongodb/control/stream-plan/analytics.json
/@mongodb/dump/analytics.archive
/@mongodb/chain-index.json
/@mongodb/oplog/local/oplog.rs.bson
/@mongodb/cluster-metadata.json
```

`manifest.json` is always expected. `control/stream-plan/<database>.json` is a small restore planning record written immediately before the corresponding `dump/<database>.archive` stream. Restore uses that FileIndex order to stream large dump archives directly into `mongorestore` instead of staging complete dump files in the File Daemon working path. `chain-index.json` and `oplog/local/oplog.rs.bson` are used by replica-set recovery chains. `cluster-metadata.json` is used by sharded-cluster restore planning.

Full Backups

A Full backup creates a complete logical artifact for the selected scope. For a standalone deployment, this is the standard consistency point available from `mongodump` for the selected databases and collections.

For a replica set, the Full backup also records the initial chain boundary. This boundary allows later Incremental and Differential jobs to capture and validate oplog ranges for point-in-time recovery inside the available oplog window.

For a sharded cluster, the plugin connects through `mongos` and captures the selected data plus routing metadata. The sharded backup is coordinated by the plugin guardrails, including the balancer check, but it is still a logical MongoDB backup rather than a config-server disaster-recovery image.

Incremental Backups

For replica sets, an Incremental backup captures oplog changes since the previous accepted chain boundary. The plugin resolves the previous Bacula artifact metadata, validates the source identity and chain digest, and confirms that the required starting timestamp is still present in the MongoDB oplog. If the oplog has rolled past the required boundary, the job must be treated as a new Full backup instead of producing an unsafe Incremental artifact.

For standalone deployments, Incremental jobs produce a full-style logical artifact because there is no replica-set oplog guarantee to use.

For sharded clusters, Incremental jobs also produce full-style logical artifacts because a single replica-set oplog chain does not represent a complete sharded cluster point-in-time state.

Incremental Artifact Lifecycle (What Is Stored, Where, and Why)

Replica-set Incremental chains are managed in two layers:

1. **Bacula catalog/storage artifacts** (authoritative recovery payload)
2. **Local incremental-state snapshots** (lightweight auto-resume cache)

The backup payload stored by Bacula remains under `/@mongodb` and includes the normal Full/Incremental chain files, such as:

- `manifest.json`
- `chain-index.json` (when chain metadata applies)
- `oplog/local/oplog.rs.bson` (for replica-set delta segments)
- small stream-plan records under `control/stream-plan/`
- dump archive files under `dump/`

The dump archives are stored in Bacula catalog/storage as normal plugin files, but current Bacula restore streams them directly from Bacula into `mongorestore --archive` for normal live restore paths. They are not used to size the File Daemon working directory.

However, the plugin also keeps a local resume cache to resolve Incremental and Differential continuity without requiring operators to pass a manual `resume_from` path:

- Base path: `path` parameter (or Bacula working path when `path` is not set)
- MongoDB workspace root: `<path>/mongodb/`
- Per-job family directory: `<path>/mongodb/<JobName>/`
- Incremental cache root: `<path>/mongodb/<JobName>/.bacula-incremental-state/`
- Snapshot directory format: `<createdEpoch>-job-<JobId>/`

- Snapshot content: `lightweight manifest.json` only

The local cache exists to answer one question quickly and safely at job start: which previously accepted artifact boundary should this Bacula Incremental or Differential job resume from?

MongoDB incremental safety needs specific metadata from the previous accepted artifact, so this cache stores:

- source fingerprint / replica set identity
- exact `segmentEndInclusive` oplog position
- optional boundary digest (`segmentEndDigest`) to detect rollbacks/divergence
- scope compatibility (databases/selection)
- security mode compatibility

What Happens If Local Incremental-State Is Missing

If no compatible local snapshot is found, the plugin cannot safely compute a resume boundary from Bacula job metadata alone. In that case:

- The requested Incremental/Differential run is downgraded to a Full-style logical backup for that execution.
- The job can still succeed, but the replica-set oplog chain is not extended by that run.
- Bacula logs include an explicit warning that no compatible resume snapshot was found.

This is a safety behavior, not silent data loss. The backup still produces a usable Full-style artifact.

Retention and Pruning of Local Snapshot Cache

The plugin manages the local resume cache automatically. It keeps only the state that can still be useful for future Incremental or Differential jobs and prunes obsolete entries when they are no longer needed, typically after a new Full backup establishes a fresh recovery-chain boundary.

This automatic pruning controls local metadata growth while preserving enough chain history for normal scheduling patterns. It affects only the lightweight `.bacula-incremental-state` cache and does not remove Bacula catalog/storage artifacts.

Architectures That Support True Incremental Chains

True oplog-chain Incremental/Differential behavior is supported only when the source is a writable **replica-set primary**.

- **Replica set primary:** true Incremental/Differential chain with oplog boundaries and digest continuity checks.
- **Standalone:** Incremental/Differential requests run as Full-style logical backups.
- **Sharded through mongos:** Incremental/Differential requests run as Full-style logical backups (single replica-set oplog is not a complete cluster-wide point-in-time source).

Differential Backups

For replica sets, a Differential backup captures oplog changes since the latest Full backup boundary. This gives Bacula administrators a shorter restore chain: the Full artifact plus the selected Differential artifact.

For standalone and sharded deployments, Differential jobs produce full-style logical artifacts for the same consistency reasons described for Incremental jobs.

Differential jobs in Bacula mode use the same local resume-cache model described in *Incremental Artifact Lifecycle (What Is Stored, Where, and Why)*. When that cache is missing or incompatible, Differential also falls back to a Full-style logical artifact for that run.

Sharded Backup Procedure

A sharded backup must be performed through mongos. The plugin records routing information to support restore planning and validates that the balancer is not moving chunks while the backup is running.

The operational sequence is:

1. Configure the Fileset to connect to mongos.
2. Stop or disable the balancer according to the local MongoDB operating procedure. This can be a manual runbook step or a Bacula RunScript hook; see MongoDBShardedBalancerRunScripts.
3. Start the Bacula backup job.
4. The plugin confirms the deployment is sharded and checks the balancer state.
5. mongodump reads the selected application databases through mongos.
6. The plugin records cluster metadata, manifest details, and checksums.
7. Re-enable the balancer after the Bacula job completes and local validation is finished.

The plugin does not claim to rebuild an entire sharded cluster config-server state automatically. The sharded artifact is intended for application-data restore through mongos with explicit metadata checks and operator control.

Security Capture

By default, the plugin captures MongoDB users and custom roles that belong to the selected restore scope. This is useful for isolated disaster recovery, test restores, and complete application recovery.

Use `skip_security=yes` when user and role management is handled by a separate identity or configuration process. Use `include_admin=yes` only when the Bacula backup must include administrative objects for a controlled isolated-DR workflow.

Failure Behavior

The plugin writes enough metadata for restore validation before accepting an artifact as usable. Checksums, manifest entries, source identity, topology, and recovery-chain boundaries are recorded so that restore can reject incomplete, corrupt, or inconsistent selections.

When a command fails, a required artifact is missing, the oplog boundary is no longer available, or a sharded guardrail is violated, the plugin reports the condition in the Bacula job log and fails the job according to `abort_on_error`. Production jobs should keep `abort_on_error=yes`.

Backup Example

Listing 38: Backup JobLog example

```
| 127.0.0.1-dir JobId 1: Start Backup JobId 1, Job=pluginTest.2026-05-12_13.  
↪ 23.57_03 |  
| 127.0.0.1-dir JobId 1: Connected to Storage "File" at 127.0.0.1:8103 with_  
↪ TLS |  
| 127.0.0.1-dir JobId 1: Created new Volume="TEST-2026-05-12:0", Pool="Default  
↪ ", MediaType="File" in catalog. |
```

(continues on next page)

(continued from previous page)

```
| 127.0.0.1-dir JobId 1: Using Device "FileStorage" to write.
|
| 127.0.0.1-dir JobId 1: Connected to Client "127.0.0.1-fd" at 127.0.0.1:8102
| with TLS
| 127.0.0.1-fd JobId 1: Connected to Storage at 127.0.0.1:8103 with TLS
|
| 127.0.0.1-sd JobId 1: Labeled new Volume "TEST-2026-05-12:0" on File device
| "FileStorage" (/tmp/regress/tmp).
| 127.0.0.1-sd JobId 1: Wrote label to prelabeled Volume "TEST-2026-05-12:0"
| on File device "FileStorage" (/tmp/regress/tmp)
| 127.0.0.1-fd JobId 1: mongodbg: Jar Version: 0.1.0 | Java version: 21.0.10 |
| Java Pid: 297566
| 127.0.0.1-fd JobId 1: mongodbg: Preparing backup artifact workspace at /tmp/
| regress/working/mongodbg/pluginTest.2026-05-12_13.23.57_03/mongodbg-bacula-
| backup-1-17338533992950670230 using hosts=127.0.0.1:27043, replicaSet=rs0,
| directConnection=true, tls=false
| 127.0.0.1-fd JobId 1: mongodbg: Connected to MongoDB source with topology
| replica-set primary (connected member localhost:27017)
| 127.0.0.1-fd JobId 1: mongodbg: Doing backup of a selection of databases:
| appdb
| 127.0.0.1-fd JobId 1: mongodbg: Doing backup of all collections within each
| selected database.
| 127.0.0.1-fd JobId 1: mongodbg: Creating dump for database appdb
|
| 127.0.0.1-fd JobId 1: mongodbg: Emitted Bacula restore stream plan control/
| stream-plan/appdb.json before archive dump/appdb.archive (security: false,
| collection scoped: false, sharded: false).
| 127.0.0.1-fd JobId 1: mongodbg: Streamed database appdb directly to Bacula
| as appdb.archive
| 127.0.0.1-fd JobId 1: mongodbg: Backup manifest written
|
| 127.0.0.1-fd JobId 1: mongodbg: Backup artifact creation completed for 1
| databases
| 127.0.0.1-fd JobId 1: mongodbg: Persisted Bacula incremental resume snapshot
| at /tmp/regress/working/mongodbg/pluginTest.2026-05-12_13.23.57_03/.bacula-
| incremental-state/1778585042-job-1
| 127.0.0.1-sd JobId 1: Elapsed time=00:00:03, Transfer rate=2.846 K Bytes/
| second
| 127.0.0.1-sd JobId 1: Sending spooled attrs to the Director. Despooling 1,
| 964 bytes ...
| 127.0.0.1-dir JobId 1: Bacula 127.0.0.1-dir 18.2.4 (10Mar26):
| Build OS: x86_64-pc-linux-gnu ubuntu 24.04
| JobId: 1
| Job: pluginTest.2026-05-12_13.23.57_03
| Backup Level: Full
| Client: "127.0.0.1-fd" 18.2.4 (10Mar26) x86_64-pc-linux-gnu,
| ubuntu,24.04
| FileSet: "FS_MONGODB" 2026-05-12 13:23:56
| Pool: "Default" (From Job resource)
| Catalog: "MyCatalog" (From Client resource)
| Storage: "File" (From Job resource)
| Scheduled time: 12-may-2026 13:23:57
```

(continues on next page)

(continued from previous page)

```
Start time:          12-may-2026 13:23:59
End time:            12-may-2026 13:24:03
Elapsed time:        4 secs
Priority:             10
FD Files Written:     5
SD Files Written:     5
FD Bytes Written:     6,380 (6.380 KB)
SD Bytes Written:     8,538 (8.538 KB)
Rate:                1.6 KB/s
Software Compression: 27.9% 1.4:1
Comm Line Compression: 0.5% 1.0:1
Snapshot/VSS:        no
Encryption:          no
Accurate:            yes
Volume name(s):      TEST-2026-05-12:0
Volume Session Id:    1
Volume Session Time:  1778585036
Last Volume Bytes:    9,487 (9.487 KB)
Non-fatal FD errors:  0
SD Errors:           0
FD termination status: OK
SD termination status: OK
Termination:         Backup OK |
| 127.0.0.1-dir JobId 1: Begin pruning Jobs older than 6 months .
|
| 127.0.0.1-dir JobId 1: No Jobs found to prune.
|
| 127.0.0.1-dir JobId 1: Begin pruning Files.
|
| 127.0.0.1-dir JobId 1: No Files found to prune.
|
| 127.0.0.1-dir JobId 1: End auto prune.
```

7.2 Restore

Enterprise

Bacula Enterprise Only

This solution is **only** available for Bacula Enterprise. For subscription inquiries, please reach out to sales@baculasystems.com.

Restore Procedure

The MongoDB Plugin restores Bacula MongoDB artifacts into a running MongoDB target. The target may be the original deployment, a replacement deployment, or a dedicated isolated disaster-recovery environment.

The Bacula restore workflow is:

1. Start a Bacula restore session.
2. Select the required MongoDB files under `/@mongodb`.
3. Use `Where=/` so plugin restore handling receives the original virtual paths.
4. Set plugin restore options such as `conflict_strategy`, `map_db`, `replay_to`, or `isolated_dr`.
5. Run `dry_run=yes` first for important production recoveries.
6. Run the restore job.
7. Validate application data, indexes, users, roles, and application behavior after restore.

Local Path File-based Restore

Use `Where=/local/fd/path` to perform a file-based restore over the File Daemon host in the selected path, instead of running the restore over a running MongoDB target.

Restore Example Session

Listing 39: **Restore Bconsole session**

```
*restore jobid=1,2 where=/
You have selected the following JobIds: 1,2

Building directory tree for JobId(s) 1,2 ...
8 nodes inserted into the tree.

You are now entering file selection mode where you add (mark) and
remove (unmark) files to be restored. No files are initially added, unless
you used the "all" keyword on the command line.
Enter "done" to leave this mode.

cwd is: /
$ cd @mongodb
cwd is: /@mongodb/
$ dir
----- 0 root      root              0  1970-01-01 01:00:00  /@mongodb/.
↪ bacula-incremental-state/
-rw-r----- 1 nobody  nogroup          1546  2026-05-12 13:24:09  /@mongodb/
↪ chain-index.json
----- 0 root      root              0  1970-01-01 01:00:00  /@mongodb/
↪ dump/
-rw-r----- 1 nobody  nogroup          3792  2026-05-12 13:24:09  /@mongodb/
↪ manifest.json
----- 0 root      root              0  1970-01-01 01:00:00  /@mongodb/
↪ oplog/
$ mark *
8 files marked.
$ done
Bootstrap records written to /tmp/regress/working/127.0.0.1-dir.restore.4.bsr

The Job will require the following (*=>InChanger):
  Volume(s)                Storage(s)                SD Device(s)
=====
```

(continues on next page)

(continued from previous page)

```
TEST-2026-05-12:0          File          FileStorage

Volumes marked with "*" are in the Autochanger.

8 BSR records selected to be restored.

Automatically selected Client: 127.0.0.1-fd
Using Catalog "MyCatalog"
Run Restore job
JobName:          RestoreFiles
Bootstrap:        /tmp/regress/working/127.0.0.1-dir.restore.4.bsr
Where:            /
Replace:          Always
FileSet:          Full Set
Backup Client:    127.0.0.1-fd
Restore Client:   127.0.0.1-fd
Storage:          File
When:             2026-05-12 13:42:29
Catalog:          MyCatalog
Priority:          10
Plugin Options:   *None*
OK to run? (Yes/mod/no): mod
Parameters to modify:
  1: Level
  2: Storage
  3: Job
  4: FileSet
  5: Restore Client
  6: When
  7: Priority
  8: Bootstrap
  9: Where
 10: File Relocation
 11: Replace
 12: JobId
 13: Plugin Options
Select parameter to modify (1-13): 13
Automatically selected : mongodbg uri="mongodbg://127.0.0.1:27043/admin?
↳replicaSet=rs0&directConnection=true" database="appdb" skip_security=yes_
↳debug=0
Plugin Restore Options
Option          Current Value          Default Value
uri:            *None*                (*none*)
host:           *None*                (*none*)
username:       *None*                (*none*)
password:       *None*                (*none*)
password_env:   *None*                (*none*)
password_file:  *None*                (*none*)
authentication_database: *None*                (*none*)
authentication_mechanism: *None*                (*none*)
```

(continues on next page)

(continued from previous page)

```
replica_set:          *None*          (*none*)
direct_connection:    *None*          (*none*)
tls:                  *None*          (*none*)
tls_ca_file:          *None*          (*none*)
tls_certificate_key_file: *None*      (*none*)
tls_allow_invalid_hostnames: *None*    (*none*)
conflict_strategy:    *None*          (*none*)
map_db:               *None*          (*none*)
collection:           *None*          (*none*)
replay_to:            *None*          (*none*)
isolated_dr:          *None*          (*none*)
recover_admin:        *None*          (*none*)
skip_security:        *None*          (*none*)
report_format:        *None*          (*none*)
dry_run:              *None*          (*none*)
stream_telemetry:     *None*          (*none*)
debug:                *None*          (*none*)
```

Use above plugin configuration? (Yes/mod/no): mod

You have the following choices:

- 1: uri (MongoDB connection string with administrative credentials for the ↵
↵restore target)
- 2: host (MongoDB host[:port] for a structured restore connection. For ↵
↵multiple hosts, use a comma-separated list)
- 3: username (MongoDB username for structured restore connections)
- 4: password (MongoDB password for structured restore connections)
- 5: password_env (Environment variable containing the MongoDB password)
- 6: password_file (File containing the MongoDB password)
- 7: authentication_database (MongoDB authentication database)
- 8: authentication_mechanism (MongoDB authentication mechanism)
- 9: replica_set (Replica set name expected by the target connection)
- 10: direct_connection (Force a direct connection to a single MongoDB node)
- 11: tls (Enable TLS for restore connections and MongoDB tools)
- 12: tls_ca_file (PEM file containing trusted CA certificates for MongoDB ↵
↵TLS)
- 13: tls_certificate_key_file (PEM file containing the client certificate ↵
↵chain and private key)
- 14: tls_allow_invalid_hostnames (Allow TLS when the certificate hostname ↵
↵does not match)
- 15: conflict_strategy (Restore conflict strategy: fail, drop, or skip)
- 16: map_db (Database remapping in source=target form. For multiple ↵
↵mappings, use a comma-separated list)
- 17: collection (Collection name to restore. For multiple collections, use ↵
↵a comma-separated list)
- 18: replay_to (Replay limit for oplog-based restore in seconds[:ordinal] ↵
↵form)
- 19: isolated_dr (Enable isolated disaster-recovery restore rules)
- 20: recover_admin (Replay the dumped admin database during isolated DR ↵
↵when supported)
- 21: skip_security (Skip users and roles during restore)
- 22: report_format (Dry-run report format: text or json)
- 23: dry_run (Run restore preflight without changing the target)
- 24: stream_telemetry (Enable verbose stream telemetry logs (bytes, ↵

(continues on next page)

(continued from previous page)

```
↪throughput, duration))
    25: debug (Change debug level)
Select parameter to modify (1-25): 1
Please enter a value for uri: mongodb://127.0.0.1:27044/admin?replicaSet=rs0&
↪directConnection=true
Plugin Restore Options
Option                                Current Value                        Default Value
uri:                                mongodb://127.0.0.1:27044/admin?replicaSet=rs0&
↪directConnection=true (*none*)
host:                                *None*                              (*none*)
username:                            *None*                              (*none*)
password:                            *None*                              (*none*)
password_env:                        *None*                              (*none*)
password_file:                       *None*                              (*none*)
authentication_database:             *None*                              (*none*)
authentication_mechanism:            *None*                              (*none*)
replica_set:                         *None*                              (*none*)
direct_connection:                   *None*                              (*none*)
tls:                                 *None*                              (*none*)
tls_ca_file:                         *None*                              (*none*)
tls_certificate_key_file:            *None*                              (*none*)
tls_allow_invalid_hostnames:         *None*                              (*none*)
conflict_strategy:                   *None*                              (*none*)
map_db:                              *None*                              (*none*)
collection:                          *None*                              (*none*)
replay_to:                           *None*                              (*none*)
isolated_dr:                         *None*                              (*none*)
recover_admin:                       *None*                              (*none*)
skip_security:                       *None*                              (*none*)
report_format:                       *None*                              (*none*)
dry_run:                             *None*                              (*none*)
stream_telemetry:                    *None*                              (*none*)
debug:                              *None*                              (*none*)
Use above plugin configuration? (Yes/mod/no): yes
Run Restore job
JobName:                             RestoreFiles
Bootstrap:                           /tmp/regress/working/127.0.0.1-dir.restore.4.bsr
Where:                               /
Replace:                             Always
FileSet:                             Full Set
Backup Client:                       127.0.0.1-fd
Restore Client:                      127.0.0.1-fd
Storage:                             File
When:                                2026-05-12 13:42:29
Catalog:                             MyCatalog
Priority:                             10
Plugin Options:                      User specified
OK to run? (Yes/mod/no): yes
Job queued. JobId=7
```

Listing 40: Restore example joblog

```
| 127.0.0.1-dir JobId 7: Start Restore Job RestoreFiles.2026-05-12_13.43.24_
↪26 |
| 127.0.0.1-dir JobId 7: Restoring files from JobId(s) 1,2
↪ |
| 127.0.0.1-dir JobId 7: Connected to Storage "File" at 127.0.0.1:8103 with
↪TLS |
| 127.0.0.1-dir JobId 7: Using Device "FileStorage" to read.
↪ |
| 127.0.0.1-dir JobId 7: Connected to Client "127.0.0.1-fd" at 127.0.0.1:8102
↪with TLS |
| 127.0.0.1-fd JobId 7: Connected to Storage at 127.0.0.1:8103 with TLS
↪ |
| 127.0.0.1-sd JobId 7: Ready to read from volume "TEST-2026-05-12:0" on File
↪device "FileStorage" (/tmp/regress/tmp). |
| 127.0.0.1-sd JobId 7: Forward spacing Volume "TEST-2026-05-12:0" to
↪addr=249 |
| 127.0.0.1-sd JobId 7: Elapsed time=00:00:01, Transfer rate=11.23 K Bytes/
↪second |
| 127.0.0.1-fd JobId 7: mongodbg: Jar Version: 0.1.0 | Java version: 21.0.10 |
↪Java Pid: 316898 |
| 127.0.0.1-fd JobId 7: mongodbg: MongoDB Bacula restore target connection:
↪hosts=127.0.0.1:27044, replicaSet=rs0, directConnection=true, tls=false |
| 127.0.0.1-fd JobId 7: mongodbg: Using Bacula restore stream plan to stream
↪archive dump/appdb.archive without staging dump payload (security: false,
↪collection scoped: false). |
| 127.0.0.1-fd JobId 7: mongodbg: Restoring database appdb into appdb
↪ |
| 127.0.0.1-fd JobId 7: mongodbg: Direct streaming restore for dump/appdb.
↪archive into appdb |
| 127.0.0.1-fd JobId 7: mongodbg: Staged Bacula restore control metadata file .
↪bacula-incremental-state/1778585042-job-1/manifest.json under /tmp/regress/
↪working/mongodbg/RestoreFiles.2026-05-12_13.43.24_26/mongodbg-bacula-restore-
↪7-16970689073729387253. |
| 127.0.0.1-fd JobId 7: mongodbg: Staged Bacula restore control metadata file
↪chain-index.json under /tmp/regress/working/mongodbg/RestoreFiles.2026-05-12_
↪13.43.24_26/mongodbg-bacula-restore-7-16970689073729387253. |
| 127.0.0.1-fd JobId 7: mongodbg: Staged Bacula restore oplog recovery payload
↪file oplog/local/oplog.rs.bson under /tmp/regress/working/mongodbg/
↪RestoreFiles.2026-05-12_13.43.24_26/mongodbg-bacula-restore-7-
↪16970689073729387253. |
| 127.0.0.1-fd JobId 7: mongodbg: Staged Bacula restore oplog recovery payload
↪file oplog/local/oplog.rs.metadata.json under /tmp/regress/working/mongodbg/
↪RestoreFiles.2026-05-12_13.43.24_26/mongodbg-bacula-restore-7-
↪16970689073729387253. |
| 127.0.0.1-fd JobId 7: mongodbg: Staged Bacula restore oplog recovery payload
↪file oplog/local/prelude.json under /tmp/regress/working/mongodbg/
↪RestoreFiles.2026-05-12_13.43.24_26/mongodbg-bacula-restore-7-
↪16970689073729387253. |
| 127.0.0.1-fd JobId 7: mongodbg: Staged Bacula restore control metadata file .
↪bacula-incremental-state/1778585049-job-2/manifest.json under /tmp/regress/
```

(continues on next page)

(continued from previous page)

```
↪working/mongodb/RestoreFiles.2026-05-12_13.43.24_26/mongodb-bacula-restore-
↪7-16970689073729387253. |
| 127.0.0.1-fd JobId 7: mongodb: Auto-selected replay boundary from included_
↪recovery chain metadata: 1778585044:3 |
| 127.0.0.1-fd JobId 7: mongodb: Replaying oplog segment 1778585036:5 ->_
↪1778585044:3 from streamed restore metadata at /tmp/regress/working/mongodb/
↪RestoreFiles.2026-05-12_13.43.24_26/mongodb-bacula-restore-7-
↪16970689073729387253/.stream-replay-artifacts/1778585049-job-2 |
| 127.0.0.1-fd JobId 7: mongodb: MongoDB Bacula restore finished in direct_
↪archive stream mode. |
| 127.0.0.1-dir JobId 7: Bacula 127.0.0.1-dir 18.2.4 (10Mar26):
Build OS:                x86_64-pc-linux-gnu ubuntu 24.04
JobId:                   7
Job:                     RestoreFiles.2026-05-12_13.43.24_26
Restore Client:          "127.0.0.1-fd" 18.2.4 (10Mar26) x86_64-pc-linux-gnu,
↪ubuntu,24.04
Where:                   /
Replace:                 Always
Start time:              12-may-2026 13:43:26
End time:                12-may-2026 13:43:29
Elapsed time:            3 secs
Files Expected:          8
Files Restored:          8
Bytes Restored:          15,921 (15.92 KB)
Rate:                    5.3 KB/s
FD Errors:               0
FD termination status:   OK
SD termination status:   OK
Termination:             Restore OK |
| 127.0.0.1-dir JobId 7: Begin pruning Jobs older than 6 months .
↪|
| 127.0.0.1-dir JobId 7: No Jobs found to prune.
↪|
| 127.0.0.1-dir JobId 7: Begin pruning Files.
↪|
| 127.0.0.1-dir JobId 7: No Files found to prune.
↪|
| 127.0.0.1-dir JobId 7: End auto prune.
↪|
+-----+
↪-----+
```

At the plugin options prompt, set values such as `conflict_strategy=drop` or `dry_run=yes` when required. See `MongoDBRestoreParameters` for the full restore option reference.

For the complete restore-specific configuration and examples, see `MongoDBRestoreParametersPlugin-Configuration`.

What Happens During Restore

Before changing the target, the plugin builds a restore plan from the selected Bacula files and the restore options. The plan includes these steps:

1. Receive restore packets from the Bacula File Daemon for files under `/@mongodb`.
2. Stage small control files such as `manifest.json`, `chain-index.json`, `cluster-metadata.json`, and `control/stream-plan/*.json` in the configured working path.
3. Use the stream-plan record, or a compatible manifest when it arrives first, to identify each `dump/*.archive` payload before the archive bytes are consumed.
4. Validate sharded direct-stream restores against a `mongos` target and stop if the target balancer is active.
5. Validate replica-set chain continuity when Full plus Incremental or Differential artifacts are selected.
6. Apply the selected restore mode with `mongorestore` and driver operations.
7. Restore users and custom roles when the artifact contains security metadata and the restore options allow it.
8. Remove empty temporary restore directories created during planning.

For current Bacula backups, large database archives are streamed from Bacula directly into `mongorestore --archive` during compatible live restores. They are not written as complete temporary dump files in the File Daemon working path. Restore planning still stages metadata because the plugin must inspect it locally before applying changes. Replica-set chain restores also stage oplog payload files under `oplog/` so they can be replayed in validated order after the streamed Full archive has been restored.

Full Application Restore

A Full application restore loads the selected databases and collections from a Full artifact. The plugin uses `mongorestore` archive input to recreate collection data, collection options, and indexes recorded by MongoDB Database Tools.

The safest production default is `conflict_strategy=fail` against an empty or prepared target. This prevents accidental overwrite when a restore is pointed at the wrong MongoDB deployment. Use `conflict_strategy=drop` only when the operator intentionally wants the plugin to drop existing target collections before restore. Use `conflict_strategy=skip` for controlled partial restores where existing collections must remain untouched.

Replica-Set Chain Restore

A replica-set chain restore uses a Full artifact plus the selected Incremental or Differential artifact set. The plugin validates the chain before modifying the target:

- The Full artifact must belong to the same source identity as the chain artifacts.
- Chain digests and boundaries must be continuous.
- Oplog files must be present and pass checksum validation.
- `replay_to` must fall inside the selected chain when it is configured.

The plugin restores the Full artifact first. It then replays oplog entries in order until the end of the selected chain or until `replay_to` is reached. This is the restore mode used for replica-set point-in-time recovery within the oplog ranges protected by Bacula.

Database Rename Restore

Use `map_db=source=target` to restore one dumped database under a different name. This is useful for test restores, application validation, and selected migration workflows.

Database rename restores are application-data restores. Security metadata is not renamed automatically. Review users, roles, and application connection strings after the restore.

Selective Collection Restore

Use `collection` to restrict restore to one or more collection names. This is useful for accidental collection deletion or controlled data extraction.

Selective collection restore does not guarantee application consistency by itself. Applications often require related collections, indexes, reference data, or user-defined metadata to be restored together.

Sharded Restore

A sharded restore is performed through `mongos`. The plugin uses the sharded metadata in the artifact to validate the restore plan and to make operator intent explicit.

For sharded application-data restore:

1. Prepare the target sharded cluster according to the MongoDB operating procedure.
2. Connect the plugin to `mongos`.
3. Stop or disable the balancer for the restore window. This can be a manual runbook step or a Bacula RunScript hook; see `MongoDBShardedBalancerRunScripts`.
4. Restore selected application databases and collections.
5. Validate chunk distribution, indexes, application behavior, and balancer state before returning the cluster to service.

Automated `admin` or complete `config` database replay is not performed for sharded clusters. Recover sharded cluster security and config-server state manually according to the MongoDB disaster-recovery runbook for that environment.

Isolated Disaster Recovery

Use `isolated_dr=yes` for a dedicated recovery target that is separate from production. This option allows restore behavior that is appropriate for a clean DR environment but risky on an active production deployment.

Set `recover_admin=yes` only when administrative users and roles from the backup must be recovered. This is normally limited to isolated standalone or replica-set recovery targets.

Dry-Run Restore

Set `dry_run=yes` to validate the restore plan without modifying the target. The plugin checks metadata, checksums, version compatibility, chain continuity, conflicts, and isolated-DR guardrails. Use `report_format=json` when the plan is consumed by BWeb, automation, or support tooling.

Restore Staging and Cleanup

Operators may see log messages about staged restore metadata resources. These messages refer to small control files required for planning and validation, such as manifests, chain indexes, checksum inventories, and sharded metadata.

Temporary directories created for restore planning are removed when they are no longer needed. Empty Bacula restore working directories, such as `RestoreFiles.<timestamp>`, are cleaned up at the end of the operation.

In current Bacula restore flows, data and metadata are handled differently by design:

- Control metadata is staged locally for validation and planning.
- Full dump archives may be streamed directly to `mongorestore` when the restore mode is compatible with streaming.
- Oplog payloads are staged when chain replay is required.

Use-case Behavior

The following summarizes where the plugin streams versus stages payloads:

Full restore (single Full artifact, standard scope):

- `dump/*.archive` is streamed directly into `mongorestore --archive`.
- Database users and roles can be restored from the same archive when the artifact contains security metadata and `skip_security` is not enabled.
- Control metadata, for example `manifest.json` and stream-plan records, is staged.

Full + Incremental or Differential replay (`full-plus-oplog`):

- The base Full `dump/*.archive` can be streamed directly.
- Oplog payload files under `oplog/` are staged for ordered replay.
- Control metadata is staged.

Standalone or sharded Incremental/Differential fallback jobs:

- These jobs are full-style logical artifacts. Their `dump/*.archive` payloads can be streamed directly during normal live restore.

Collection-scoped restore (`collection=...`):

- Compatible with archive streaming in supported Full and Full+oplog modes.
- Namespace filters are applied during restore.

Sharded restore workflows through mongos:

- The plugin stages sharded planning metadata and validates the target balancer state before applying data through `mongorestore`.

`dry_run=yes`:

- Metadata validation only; no live data apply.

Special restore modes such as `explicit_replay_to`, `map_db`, `isolated_dr=yes`, or `recover_admin=yes` may require the conservative local artifact path for additional validation and safety checks.

Compatibility and Guardrails

Compatible with direct archive streaming:

- Simple Full restore scenarios that do not require remapping or DR-only behaviors.
- Simple Full restore with or without application user/role replay, when the archive format and selected scope support streaming.

- Full plus oplog replay when the replay boundary is selected from staged chain metadata and oplog files.

Not compatible with direct archive streaming (fallback to staged flow):

- Explicit `replay_to`, `map_db`, `isolated_dr`, `recover_admin`, `dry_run`, and sharded restore orchestration paths.
- Older artifacts or selected file sets that do not provide enough metadata before the dump archive arrives.

Operational Notes

- Seeing staged-file log lines during a streamed restore is expected. A restore can stream dump archives and still stage metadata and oplog resources.
- Security replay for simple Full restores can use the same streamed archive; DR-specific admin recovery remains guarded by isolated-DR restore rules.

7.3 List and Query

Enterprise

Bacula Enterprise Only

This solution is **only** available for Bacula Enterprise. For subscription inquiries, please reach out to sales@baculasystems.com.

The MongoDB Plugin supports Bacula LIST and QUERY operations for inventory and configuration assistance. These operations connect to MongoDB and return information about the server, databases, and collections.

Typical uses include:

- Confirming that the File Daemon can connect to MongoDB.
- Listing databases before creating Filesets.
- Listing collections for selective backup planning.
- Feeding BWeb scan-plugin/Automation workflows.

The exact command syntax depends on the Bacula interface used. In Bconsole, `.query` uses a `parameter` selector and `.ls` uses a `path` selector. The MongoDB plugin line contains the connection parameters only.

Listing 41: Database inventory query

```
*.query client=mongodb-fd plugin="mongodb: uri=mongodb://backup:secret@mongo1.
example.com:27017/admin?authSource=admin" parameter=database
```

Listing 42: Collection listing

```
*.ls client=mongodb-fd plugin="mongodb: uri=mongodb://backup:secret@mongo1.  
example.com:27017/admin?authSource=admin" path=/collections/appdb
```

Useful selectors include `all`, `server`, `database`, `collection`, `/db/<database>`, `/collections/<database>`, and `/collection/<database>/<collection>`.

For large deployments, restrict inventory queries to the databases or collections that are relevant to backup configuration. This avoids unnecessary output and reduces the permissions required for discovery.

7.4 Sharded Balancer RunScript Hooks

Enterprise

Bacula Enterprise Only

This solution is **only** available for Bacula Enterprise. For subscription inquiries, please reach out to sales@baculasystems.com.

For sharded MongoDB backup and restore operations, the Bacula MongoDB Plugin connects through `mongos` and validates that the balancer is not moving chunks during the protected operation. The plugin does not permanently stop, start, or reconfigure the balancer. Balancer control remains an operator-side procedure because each MongoDB environment can have different maintenance windows, automation, change-control requirements, and Atlas or Kubernetes operator policies.

For this reason, stopping or disabling the balancer is normally handled by one of the following:

- A manual MongoDB operating runbook before and after the Bacula job.
- A site-specific automation script launched outside Bacula.
- Bacula RunScript hooks configured on the MongoDB backup or restore job.

The MongoDB Plugin guardrail still applies: if the plugin detects that the balancer is active when a sharded backup or restore requires it to be stopped, the operation fails instead of continuing with an unsafe logical artifact.

Bacula Job Example

The following example runs a site-local script before the backup job to stop the balancer and another script after the job to restore the normal balancer state. Use `FailJobOnError = yes` for the pre-job hook so a failed balancer stop does not allow the backup to continue.

Listing 43: Bacula Job with sharded balancer hooks

```
Job {  
  Name = "MongoDB-Sharded-Backup"  
  Type = Backup  
  Level = Full  
  Client = mongodb-fd  
  FileSet = "FS-MongoDB-Sharded"  
  Schedule = "Nightly"
```

(continues on next page)

(continued from previous page)

```
Storage = File1
Pool = Default
Messages = Standard

RunScript {
    RunWhen = Before
    RunOnClient = yes
    FailJobOnError = yes
    Command = "/opt/bacula/scripts/mongodb-stop-balancer.sh"
}

RunScript {
    RunWhen = After
    RunOnClient = yes
    RunOnFailure = yes
    Command = "/opt/bacula/scripts/mongodb-start-balancer.sh"
}
}
```

The restore job can use the same pattern when restoring into a sharded target. In that case, the scripts must point to the restore target mongos endpoint, not necessarily the original backup source.

Example Stop Script

The following script is a basic example for self-managed MongoDB deployments. It connects to mongos with mongosh, stops the balancer, and waits until no balancing round is active. Adjust the URI, credential source, TLS options, timeout, logging, and approval controls to match the local MongoDB operating procedure.

Listing 44: `/opt/bacula/scripts/mongodb-stop-balancer.sh`

```
#!/bin/sh
set -eu

MONGODB_URI="${MONGODB_URI:-mongodb://backup:secret@mongos1.example.com:27017/
↪admin?authSource=admin}"
TIMEOUT_SECONDS="${TIMEOUT_SECONDS:-300}"

mongosh "${MONGODB_URI}" --quiet --eval "
const timeoutMs = ${TIMEOUT_SECONDS} * 1000;
const start = Date.now();

sh.stopBalancer();

while (sh.isBalancerRunning()) {
    if (Date.now() - start > timeoutMs) {
        throw new Error('Timed out waiting for MongoDB balancer to stop');
    }
    sleep(1000);
}

print('MongoDB balancer is stopped');
"
```

Example Start Script

The post-job script can re-enable the balancer after the Bacula job completes and after any site-required validation has run. Some environments prefer not to re-enable the balancer automatically, for example when a manual validation gate or a scheduled balancer window is required. In those cases, replace this script with the approved local procedure.

Listing 45: `/opt/bacula/scripts/mongodb-start-balancer.sh`

```
#!/bin/sh
set -eu

MONGODB_URI="${MONGODB_URI:-mongodb://backup:secret@mongos1.example.com:27017/
↪admin?authSource=admin}"

mongosh "${MONGODB_URI}" --quiet --eval "
sh.startBalancer();
print('MongoDB balancer start requested');
"
```

Operational Notes

- `mongos` is the remote MongoDB query router used as the sharded cluster endpoint. It is not a local executable run by the plugin.
- `mongosh` is shown only in these operator-side example scripts. The MongoDB Plugin itself does not require `mongosh` for backup or restore.
- Store script credentials according to the local security policy. Prefer protected files, environment injection, or a secret-management system over clear-text command lines.
- Keep the balancer stopped for the whole backup or restore window. The plugin validates this state as a safety guardrail.
- In managed environments, such as Atlas or Kubernetes operator deployments, use the vendor-supported method for pausing and resuming balancing instead of copying these scripts unchanged.

8 MongoDB Plugin: Limitations

Enterprise

Bacula Enterprise Only

This solution is **only** available for Bacula Enterprise. For subscription inquiries, please reach out to sales@baculasystems.com.

The MongoDB Plugin is designed for Bacula-managed logical backup and restore workflows. The following operating boundaries help keep backup artifacts consistent and restore behavior predictable.

- The plugin runs from a Bacula File Daemon host and is currently focused on Linux deployments.
- MongoDB Database Tools must be installed and executable by the File Daemon environment, as described in `MongoDBToolsInstallation`.

- Replica set operations should connect through a replica set URI or directly to the writable primary. Direct secondary backup is rejected.
- Sharded cluster operations should connect through mongos. The plugin does not back up shard members or config servers directly, and config database recovery remains a manual administrative operation.
- Sharded Incremental and Differential jobs are stored as full-style logical artifacts, not oplog deltas. Sharded oplog point-in-time replay is not currently exposed.
- Sharded admin replay is intentionally blocked by automated restore guardrails; recover sharded users and roles manually when required.
- Standalone Incremental and Differential jobs are also stored as full-style logical artifacts because standalone servers do not provide an oplog chain.
- Restore to an older MongoDB major version is rejected. Restore to the same or a newer compatible major version is the expected path.
- MONGODB-X509 requires TLS, a client certificate/key file, and authentication_database=\$external.
- tls_allow_invalid_hostnames=yes reduces TLS identity validation and should be limited to controlled test or migration scenarios.

9 MongoDB Plugin: Best Practices

Enterprise

Bacula Enterprise Only

This solution is **only** available for Bacula Enterprise. For subscription inquiries, please reach out to sales@baculasystems.com.

The following practices are recommended for production MongoDB backups with Bacula Enterprise.

9.1 General Practices

- Use one Bacula job per MongoDB deployment or per clear recovery scope.
- Prefer structured connection parameters with password_file over embedding passwords in URIs.
- Run dry_run=yes before enabling a new Fileset in production.
- Test restore into a non-production MongoDB target regularly.
- Keep MongoDB Database Tools aligned with the protected MongoDB server family.
- Monitor Bacula job logs for topology, compatibility, and chain warnings.
- Keep the File Daemon host clock synchronized; recovery-chain metadata uses timestamps.

9.2 Replica-Set Practices

- Connect to a writable primary, not to a secondary.
- Size the oplog so the oldest available oplog entry cannot roll past the backup interval.
- Schedule regular Full backups to limit chain length and recovery time.
- Keep all required Incremental or Differential jobs available until the next Full chain is validated.
- Test point-in-time restore with `replay_to` before relying on it during an incident.

9.3 Sharded-Cluster Practices

- Connect only through mongos.
- Stop the balancer before backup and restore.
- Confirm that the balancer is not active before starting the Bacula job.
- Treat sharded artifacts as application-data recovery artifacts, not full config-server disaster-recovery images.
- Maintain a separate MongoDB operational runbook for config-server and cluster-wide security recovery.

9.4 Security Practices

- Use a dedicated MongoDB backup user.
- Grant only the privileges required for the selected backup mode.
- Use `skip_security=yes` for data-only lower-privilege backups.
- Use `include_admin=yes` only for isolated disaster-recovery artifacts.
- Protect password files and TLS private keys with filesystem permissions readable only by the File Daemon service user.
- For MONGODB-X509, document the exact certificate subject used as the MongoDB username.

9.5 Restore Practices

- Prefer restoring to an empty target with `conflict_strategy=fail` for validation restores.
- Use `dry_run=yes` before production restore.
- Use `conflict_strategy=drop` only when the target data can be overwritten.
- Record the selected JobIds and restore options in the incident ticket or recovery log.
- Run application-level validation after MongoDB restore completes.

10 MongoDB Plugin: Troubleshooting

Enterprise

Bacula Enterprise Only

This solution is **only** available for Bacula Enterprise. For subscription inquiries, please reach out to sales@baculasystems.com.

This section lists common problems and corrective actions for the MongoDB Plugin.

10.1 Common Issues

- **Plugin not visible in status client:** verify package installation, Plugin Directory in the File Daemon configuration, and File Daemon restart.
- **mongodump or mongorestore not found:** install MongoDB Database Tools as described in `MongoDBToolsInstallation`, or make the commands available in the File Daemon service PATH. The Bacula job log reports `Required MongoDB Database Tool not available: mongodump` or `Required MongoDB Database Tool not available: mongorestore` when this pre-flight check fails.
- **Authentication failed:** verify username, password source, `authentication_database`, and `authentication_mechanism`.
- **X.509 authentication failed:** verify `tls=yes`, `authentication_database=$external`, client certificate/key file readability, and the exact subject string used as username.
- **TLS certificate error:** verify `tls_ca_file` and server certificate hostnames. Use `tls_allow_invalid_hostnames=yes` only as a temporary diagnostic or lab setting.
- **Backup rejects secondary:** connect to the replica-set primary or to a seed list that resolves to the primary.
- **Oplog rollover detected:** increase oplog size, run backups more frequently, or take a new Full backup.
- **Incremental restore chain is incomplete:** select the required Full and all needed Incremental or Differential artifacts.
- **Sharded backup reports balancer problem:** stop the balancer, wait until balancing is inactive, and rerun the job.
- **Restore conflict on existing collection:** use an empty target, or set `conflict_strategy=drop` or `conflict_strategy=skip` according to the recovery objective.
- **Admin restore rejected:** use `isolated_dr=yes` and, only for standalone or replica-set isolated DR, `recover_admin=yes`.

10.2 Diagnostic Checklist

1. Confirm the Bacula job log shows the expected plugin line and job level.
2. Confirm the File Daemon host can run `mongodump --version` and `mongorestore --version`.
3. Test MongoDB connectivity from the File Daemon host using equivalent connection parameters.
4. Run a backup or restore with `dry_run=yes`.
5. Review the plugin debug log configured by `log` or the default plugin working path.
6. For replica-set chains, verify that all required JobIds are available in Bacula.
7. For sharded clusters, confirm the target is mongos and the balancer is stopped.

10.3 Useful Bacula Checks

Listing 46: **Bacula status and restore checks**

```
*status client=mongodb-fd
*list jobs name=MongoDB-Backup
*restore jobid=123 client=mongodb-fd where="/"
```

When escalating a support case, include the Bacula job log, plugin debug log, MongoDB server version, MongoDB Database Tools version, topology type, and the Fileset plugin line with secrets redacted.