mariaDB

GUIDE

Bridging the gap: MySQL to MariaDB migration guide



CONTENTS

Table of contents

01	**Introduction**	3
02	**The evolution of MariaDB: compatibility vs. feature divergence**	4
03	**The strategy for success: dedicated infrastructure**	5
04	**Key technical deviations**	6
	1. JSON data type handling	6
	2. Authentication differences	7
	3. Data type defaults for virtual columns	7
	4. Metadata (system tables) management — architectural differences	7
	5. User privilege differences and mapping the security model	8
	6. Data-at-rest encryption and compression	9
	7. Replication and advanced clustering considerations	9
05	**MySQL Galera cluster migration (MySQL 8.0 and higher to MariaDB 11.4/11.8.1)**	10
06	**Summary**	11
07	**About MariaDB**	12

01

Introduction

MariaDB and MySQL share a foundational, common lineage. However, over time, the technologies have diverged in critical areas — particularly concerning performance optimization, specific storage engines, and key configuration settings. Despite these differences, MariaDB maintains high compatibility with MySQL, which means that most applications can transition without code changes, significantly streamlining the migration process.

This comprehensive guide is your essential blueprint. We outline actionable, step-by-step measures to ensure your move to MariaDB is successful and stable.

02

The evolution of MariaDB: compatibility vs. feature divergence

It is essential to recognize that MariaDB has evolved far beyond being a direct fork of MySQL. While older releases (such as MariaDB 5.5 and MySQL 5.5) were near drop-in replacements, modern release series (MySQL 8.0 and beyond) are now best defined as **application-compatible database systems**.

This shift means that achieving application compatibility requires acknowledging the divergence in critical areas like replication, specific engine behaviors, and configuration syntax.

03

The strategy for success: dedicated infrastructure

For this reason, a high-assurance migration strategy mandates transitioning to a new, dedicated MariaDB infrastructure rather than performing an in-place upgrade. This methodology is the recommended way to:

- **Minimize risk:** Decouple the transition from the existing production environment.
- **Ensure stability:** Allow for comprehensive performance baselining and rigorous functional testing on the new platform.
- **Maximize safety:** Establish a clear and immediate rollback path (disaster recovery).

This approach makes the migration more gradual and predictable, making it a controlled and staged transition.

SERVER VERSION	END OF STANDARD SUPPORT	END OF LIFE (EOL) DATE	COMPATIBILITY WITH MYSQL 8+
11.8	October 22, 2030	October 22, 2033	Highest compatibility and recommended target. caching_sha2_password is available. Offers the longest Enterprise support cycle and best functional alignment to MySQL 8.0.
11.4	January 16, 2030	January 16, 2033	Very high compatibility. The caching_sha2_password is available in the latest releases, significantly reducing authentication hurdles.
10.6	August 23, 2027	August 23, 2029	High compatibility, but requires checks. caching_sha2_password is available in the latest releases, significantly reducing authentication hurdles.

Details about the support lifecycle for all products is available under the MariaDB Engineering Policy.

Legacy systems: For MySQL v5.x installations, which are past EOL, it is strongly recommended to migrate directly to the latest MariaDB Enterprise Server release to gain access to security patches, features, and the longest support life cycle.

04

Key technical deviations

Before initiating any migration, it is absolutely crucial to prepare your environment and codebase:

- **Review incompatibilities:** Always verify the specific feature differences and incompatibilities between your current MySQL version and your target MariaDB version. MariaDB maintains high compatibility, but checking for differences helps you to select the most suitable version.
- **Backup:** Before making any changes, always back up your data to ensure recovery.
- **Test applications:** While the syntax for application code and database scripts is often highly compatible, thorough testing and validation are required due to potential functional differences.

Most differences often appear in data storage formats, user authentication, and encryption methods.

1. JSON data type handling

MariaDB and MySQL handle the storage of JSON data differently. MariaDB implements the JSON data type as LONGTEXT while MySQL uses a binary format for JSON.

Migration considerations:

- **Replication/Galera:** If you are migrating a cluster using replication or Galera, you must convert JSON columns to TEXT in the MySQL source to ensure consistent data storage between systems.
- **SQL changes/Casting:** If application SQL contains direct `CAST(… AS JSON)`, these will fail on MariaDB because JSON is an alias and not a native type. The remediation would be to cast it to LONGTEXT.
- **JSON functional differences (MySQL 8.4 to MariaDB 11.8):** Identify and replace the MySQL-specific shorthand operators (`->` , `->>`) and any unavailable functions being used (e.g., `JSON_STORAGE_SIZE()`).
- Due to the implementation differences, thorough testing and validations are required for complex JSON functions to catch any subtle differences.

Remediation — tools.

- **Automatic conversion:** MariaDB versions 10.6.15 or above, or 10.11.6 (and newer), will automatically convert the MySQL JSON type to a MariaDB TEXT column during the upgrade process using the mariadb-upgrade tool.
- **Migration tools:** MariaDB's migration assessment tool, SQLines, identifies incompatibilities and converts MySQL schemas into a supported MariaDB schema format.

Remediation — manually.

- **Identify JSON columns:** Use `SELECT table_schema, table_name FROM information_schema.COLUMNS WHERE data_type='JSON'`
- **Convert JSON to TEXT:** Execute `ALTER TABLE table_name MODIFY json_column LONGTEXT`

2. Authentication differences

The implementation of user authentication has historically been a major point of incompatibility; however, the caching_sha2_password authentication plugin is now available to address this.

- **Newer releases (compatible):** The caching_sha2_password authentication plugin has been added as a compatibility feature for MySQL. Starting with the MariaDB Community Server 11.4, 11.8 and MariaDB Enterprise Server 11.8.3-1 release, this is no longer a challenge.
- **Older releases (incompatible):** For releases prior to the versions mentioned above, the caching_sha2_password authentication plugin is not supported. It may be backported to 10.6 in the future.

3. Data type defaults for virtual columns

Incompatibilities arise from how MySQL 8.0 handles default values on large text fields and the syntax for defining constraints on generated columns.

- **Virtual column NULL/NOT NULL syntax:** MariaDB currently does not support NULL/NOT NULL syntax for virtual (computed) columns unlike MySQL. In MariaDB, using NOT NULL on a virtual column is semantically equivalent to applying a CHECK constraint (e.g., `CONSTRAINT col_not_null CHECK (col IS NOT NULL)`).
- **Identification (virtual columns):** To check if your tables use virtual columns in MySQL with NULL/NOT NULL constraints, query the column metadata and update the constraints.

```sql
SELECT TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME, COLUMN_TYPE, EXTRA, IS_NULLABLE
FROM INFORMATION_SCHEMA.COLUMNS
WHERE EXTRA LIKE '%VIRTUAL%' OR EXTRA LIKE '%STORED%'
ORDER BY TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME;
```

MariaDB's Migration Assessment Tool can identify these and convert them automatically into a supported MariaDB schema.

4. Metadata (system tables) management — architectural differences

- **MariaDB's metadata management** is a major difference from MySQL 8.0's approach. MariaDB uses .frm files and system tables; MySQL 8.0+ uses a unified Global Data Dictionary (DD).
- **Migration path:** Due to this architectural divergence, avoiding in-place upgrades and an export/import build migration strategy is recommended with pre/post data migration remediation tasks.

REMEDIATION TASK	DETAILS AND REQUIRED ACTION
User and privileges	The mysql.user table structure is incompatible and was not restored. You must manually re-create all necessary users and grants on the new MariaDB server.
Authentication (SHA-2)	The required feature is available in the latest releases of MariaDB 10.6 and 11.8.3-1.

5. User privilege differences and mapping the security model

While MariaDB maintains high syntactic compatibility with MySQL, the core difference lies in how MySQL and MariaDB classify and manage permissions, impacting your migration strategy, particularly for administrative users.

The two systems diverge in their approach to granting administrative control:

- **MariaDB:** Relies on Static Privileges (hardcoded, built-in permissions like SELECT) and a Role system for granular administrative control.
- **MySQL 8.0+:** Uses Static Privileges plus Dynamic Privileges (permissions defined and managed by components/plugins at runtime, e.g., BACKUP_ADMIN).

Remediation. Use Show Privileges to identify any MySQL user relying on a Dynamic Privilege. You must then map these functions to MariaDB's granular Static Privileges or its Role structure.

MariaDB introduced an architectural change by dividing the monolithic SUPER privilege into multiple, distinct, granular privileges. This allows administrators to adopt a security model based on the principle of least privilege.

If MySQL users relied on the single SUPER privilege, you must now explicitly grant several distinct MariaDB privileges to achieve the same administrative scope.

MARIADB GRANULAR PRIVILEGE	FUNCTION/PURPOSE
CONNECTION ADMIN	Administers connection resource limits and allows ignoring the max_user_connections variable.
REPLICA MONITOR	Required to execute SHOW REPLICA STATUS and SHOW RELAYLOG EVENTS.
SET USER	Enables setting the DEFINER when creating routines or triggers.
READ_ONLY ADMIN	Allows a user to perform write operations even when the database is protected by the read_only system variable.
BINLOG MONITOR	The functional equivalent of the older REPLICATION CLIENT privilege (though the old name remains an alias).

This means you must explicitly grant multiple privileges (e.g., BINLOG ADMIN, REPLICATION MASTER ADMIN, REPLICATION SLAVE ADMIN) instead of a single SUPER grant.

6. Data-at-rest encryption and compression

The implementation of table-level encryption and compression features differs significantly between MySQL and MariaDB.

- **Migration requirement:** Encrypted or compressed tables must be de-encrypted/decompressed before migration using a logical dump, and then re-encrypted/re-compressed after the tables are loaded into MariaDB.

Detect encrypted tables:

```sql
SELECT table_schema, table_name, create_options
FROM information_schema.TABLES
WHERE create_options LIKE '%ENCRYPT%';
```

Detect compressed tables:

```sql
SELECT table_schema, table_name, create_options
FROM information_schema.TABLES
WHERE create_options LIKE '%COMP%';
```

7. Replication and advanced clustering considerations

When migrating highly available setups (like Galera clusters) or replication environments, specific configuration changes are mandatory.

Replication from MySQL 8.0 to MariaDB. MariaDB versions MariaDB 11.4/11.8 and newer are capable of replicating from a MySQL 8.0 and higher primary, but require configuration adjustments on the MySQL side.

- **Disable incompatible events:** You must disable binary log compression and incompatible row update events in your MySQL 8.0 configuration file: set `binlog-row-value-options=''` (disables PARTIAL_UPDATE_ROWS_EVENT); set `binlog_transaction_compression=0` (disables TRANSACTION_PAYLOAD_EVENT). It is also recommended to set `binlog_format=row` for cross-replication compatibility.
- **GTID handling:** MySQL implements GTID differently from MariaDB, hence MariaDB replica must use the binary log file and position by explicitly disabling GTID usage for the master connection.

```
COMMAND ON REPLICA

CHANGE MASTER ... MASTER_LOG_FILE = file_name MASTER_LOG_POS = # MASTER_USE_GTID = no;
```

05

MySQL Galera cluster migration (MySQL 8.0 and higher to MariaDB 11.4/11.8.1)

If performing a node-by-node in-place migration from a MySQL Galera Cluster, specific parameters are required on the MariaDB configuration, including setting gcs.check_appl_proto to 0 to prevent application protocol mismatch messages, and setting wsrep_sst_method to mysqldump for the first MariaDB node joining the MySQL cluster.

COMPONENT	STATUS/CONTEXT	REMEDIATION STRATEGY
mysql.user & privilege tables	Underlying system tables are structured differently and cannot be migrated directly.	Manual re-creation: administrators must manually recreate necessary users and grants on the MariaDB target server.
Data backup workaround	To avoid corruption during cluster migration methods (like node-by-node), the table must be excluded.	During the backup process (e.g., using mysqldump), the mysql.user table should be skipped using the --skip-table=mysql.user option.
General privilege update	Even after a clean install on the target server, MariaDB requires post-installation steps to integrate its unique fields.	Best practice: run mariadb-upgrade to ensure system tables (like privilege and event tables) are fully consistent and utilize all fields unique to the MariaDB version.

06

Summary

MariaDB Enterprise Platform provides a clear path for organizations looking to move beyond the limitations of legacy MySQL and reclaim control over their data roadmap. This guide serves as your technical blueprint for that transition, ensuring a stable migration that preserves your existing application logic.

Moving to MariaDB establishes a multimodal foundation for your data. By integrating OLTP, OLAP, JSON and AI workflows, you ensure your architecture is ready for the next decade of scale.

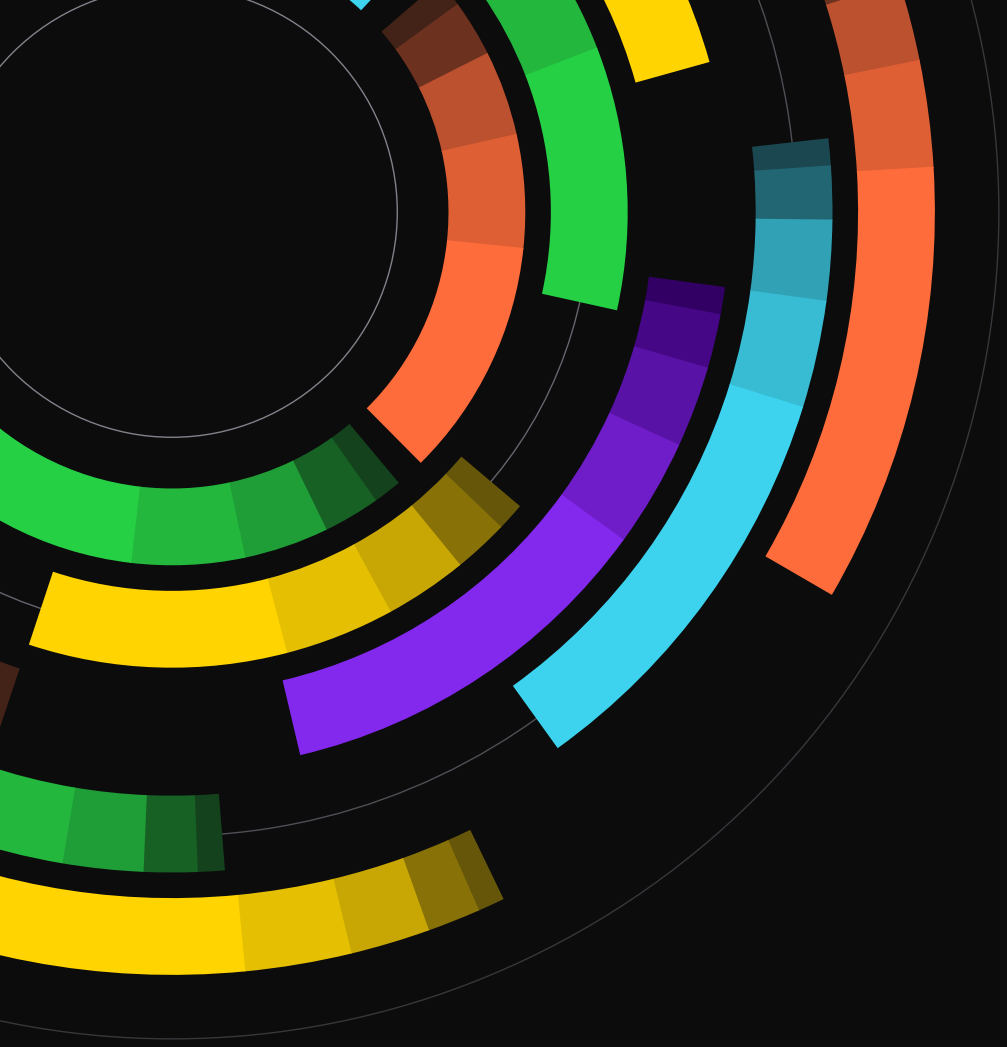
07

About MariaDB

MariaDB seeks to eliminate the constraints and complexity of proprietary databases, enabling organizations to reinvest in what matters most – rapidly developing innovative, customer-facing applications. Enterprises can depend on a single complete database for all their needs, that can be deployed in minutes for transactional, analytical and hybrid use cases. Trusted by organizations such as Deutsche Bank, DBS Bank and Samsung – MariaDB delivers customer value without the financial burden of legacy database providers. For more information, please visit mariadb.com.

Contact us to learn more about MariaDB

mariadb.com/contact/



mariaDB

Contact us to learn more about MariaDB

mariadb.com/contact/