MySQL NDB Cluster 8.3 Release Notes

Abstract

This document contains release notes for the changes in MySQL NDB Cluster version 8.3.

NDB Cluster 8.3 is based on MySQL Server 8.3 and uses version 8.3 of the `NDB` storage engine, as shown in the version string returned by executing `SELECT VERSION()` in the `mysql` client, or by executing the `ndb_mgm` client `SHOW` or `STATUS` command; for more information, see MySQL NDB Cluster 8.3.

For general information about features added in NDB Cluster 8.3, see What is New in MySQL NDB Cluster 8.3. For a complete list of all bug fixes and feature changes in MySQL NDB Cluster, please refer to the changelog section for each individual NDB Cluster release.

For additional MySQL 8.3 documentation, see the MySQL 8.3 Reference Manual, which includes an overview of features added in MySQL 8.3 that are not specific to NDB Cluster (What Is New in MySQL 8.3), and discussion of upgrade issues that you may encounter for upgrades from MySQL 8.2 to MySQL 8.3 (Changes in MySQL 8.3). For a complete list of all bug fixes and feature changes made in MySQL 8.3 that are not specific to `NDB`, see Changes in MySQL 8.3.0 (2024-01-16, Innovation Release).

Updates to these notes occur as new product features are added, so that everybody can follow the development process. If a recent version is listed here that you cannot find on the download page (https://dev.mysql.com/downloads/), the version has not yet been released.

The documentation included in source and binary distributions may not be fully up to date with respect to release note entries because integration of the documentation occurs at release build time. For the most up-to-date release notes, please refer to the online documentation instead.

For legal information, see the Legal Notices.

For help with using MySQL, please visit the MySQL Forums, where you can discuss your issues with other MySQL users.

Document generated on: 2024-04-30 (revision: 28294)

Table of Contents

Preface and Legal Notices

This document contains release notes for MySQL NDB Cluster version 8.3.

Legal Notices

Documentation Accessibility

Access to Oracle Support for Accessibility

Changes in MySQL NDB Cluster 8.3.0 (2024-01-16, Innovation Release)

MySQL NDB Cluster 8.3 is an Innovation release of NDB 8.3, based on MySQL Server 8.3 and including features in version 8.3 of the `NDB` storage engine, as well as fixing recently discovered bugs in previous NDB Cluster releases.

Obtaining NDB Cluster 8.3. NDB Cluster 8.3 source code and binaries can be obtained from https://dev.mysql.com/downloads/cluster/.

For an overview of changes made in NDB Cluster 8.3, see What is New in MySQL NDB Cluster 8.3.

This release also incorporates all bug fixes and changes made in previous NDB Cluster releases, as well as all bug fixes and feature changes made in mainline MySQL 8.3 (see Changes in MySQL 8.3.0 (2024-01-16, Innovation Release)).

- Compilation Notes
- Functionality Added or Changed
- Bugs Fixed

Compilation Notes

- **NDB Cluster APIs:** In MySQL 8.0 and later, it was necessary to build MGM API applications using a C++ compiler. In addition, the compiler requirements for both NDB API and MGM API applications were not consistent between NDB Cluster releases. This fix addresses both issues as follows:

  - MGM API applications now require a C compiler that supports C99 or later.
  - NDB API applications now require a compiler that supports C++11 or later.

  Pre-release testing has also been improved to ensure that future versions of the APIs continue to meet these requirements.

  For more detailed information about language support and compiler requirements for building NDB Cluster API applications, including those for previous versions of NDB, see General Requirements. (WL #15908)

- NDB Cluster did not compile correctly on Ubuntu 23.10. (Bug #35847193)
- It is now possible to build NDB Cluster for the s390x platform.

  Our thanks to Namrata Bhave for the contribution. (Bug #110807, Bug #35330936)

Functionality Added or Changed

- This release implements support for network communications between NDB nodes secured by Transport Layer Security (TLS) and Internet Public Key Infrastructure (PKI) to authenticate and encrypt connections, and between the NDB management server and its clients. TLS is applied both to the NDB Transporter Protocol, and to the NDB Management Protocol. In both cases, this is done using TLS mutual authentication.

  (Connections that use the MySQL client protocol employ MySQL user authentication which can use TLS; see Using Encrypted Connections, for more information.)

  A new tool `ndb_sign_keys` can be used to create and manage CA, certificate files, and keys. You can generate a set of keys and certificates for all nodes in a cluster using `ndb_sign_keys --create-key`.

  Private keys are created in place, so that copying of files containing private keys is minimized. Both private keys and certificates are labeled as either active or pending; `ndb_sign_keys` also provides help with rotating keys to allow for pending keys to replace active keys before the active keys expire.

  You can test node TLS connections with `ndb_mgm --test-tls`, or from within the `ndb_mgm` client using the `TLS INFO` command. You can also obtain information about certificates used by cluster nodes by checking the `ndbinfo certificates` table.

  You can enforce a requirement for TLS on the cluster, by setting the appropriate client options and node configuration parameters. See Using TLS Connections, for details.

  Use of TLS connections is also now supported in NDB Cluster API applications. For information about MGM API support, see TLS Functions. The NDB API now provides `configure_tls()` `get_tls_certificate_path()` methods of `Ndb_cluster_connection` for setting up TLS connections by clients.

  For more information, see TLS Link Encryption for NDB Cluster, and ndb_sign_keys — Create, Sign, and Manage TLS Keys and Certificates for NDB Cluster. (WL #15135, WL #15154, WL #15166, WL #15521)

Bugs Fixed

- **NDB Replication:** An internal thread memory usage self-check was too strict, invoking unnecessary file rotation and possibly increased memory usage. (Bug #35657932)

- **NDB Replication:** `CREATE USER` on a source cluster caused SQL nodes attached to the replica clusters to exit. (Bug #34551954)

  References: See also: Bug #112775, Bug #33172887, Bug #33542052, Bug #35928350.

- **NDB Replication:** Replicating a `GRANT NDB_STORED_USER` statement with replication filters enabled caused the SQL node to exit. This occured since the replication filter caused all non-updating queries to return an error, with the assumption that only changes needed to be replicated.

  Our thanks to Mikael Ronström for the contribution. (Bug #112775, Bug #35928350)

  References: See also: Bug #34551954, Bug #33172887, Bug #33542052.

- **NDB Replication:** On an NDB Replication setup where an SQL node in a replica cluster had `read_only=ON`, a `DROP DATABASE` statement on the source cluster caused the SQL thread on the replica server to hang with `Waiting for schema metadata lock`.

- **NDB Cluster APIs:** An event buffer overflow in the NDB API could cause a timeout while waiting for `DROP TABLE`. (Bug #35655162)

  References: See also: Bug #35662083.

- **ndbinfo Information Database:** An assumption made in the implementation of `ndbinfo` is that the data nodes always use the same table ID for a given table at any point in time. This requires that a given table ID is not moved between different tables in different versions of NDB Cluster, as this would expose an inconsistency during a rolling upgrade. This constraint is fairly easily maintained when `ndbinfo` tables are added only in the latest release, and never backported to a previous release series, but could be problematic in the case of a backport.

  Now we ensure that, if a given `ndbinfo` table added in a newer release series is later backported to an older one, the table uses the same ID as in the newer release. (Bug #28533342)

- When a node failure is detected, transaction coordinator (TC) instances check their own transactions to determine whether they need handling to ensure completion, implemented by checking whether each transaction involves the failed node, and if so, marking it for immediate timeout handling. This causes the transaction to be either rolled forward (commit) or back (abort), depending on whether it had started committing, using the serial commit protocol. When the TC was in the process of getting permission to commit (`CS_PREPARE_TO_COMMIT`), sending commit requests (`CS_COMMITTING`), or sending completion requests (`CS_COMPLETING`), timeout handling waited until the transaction was in a stable state before commencing the serial commit protocol.

  Prior to the fix for Bug#22602898, all timeouts during `CS_COMPLETING` or `CS_COMMITTING` resulted in switching to the serial commit-complete protocol, so skipping the handling in any of the three states cited previously did not stop the prompt handling of the node failure. It was found later that this fix removed the blanket use of the serial commit-complete protocol for commit-complete timeouts, so that when handling for these states was skipped, no node failure handling action was taken, with the result that such transactions hung in a commit or complete phase, blocking checkpoints.

  The fix for Bug#22602898 removed this stable state handling to avoid it accidentally triggering, but this change also stopped it from triggering when needed in this case where node failure handling found a transaction in a transient state. We solve this problem by modifying `CS_COMMIT_SENT` and `CS_COMPLETE_SENT` stable state handling to perform node failure processing if a timeout has occurred for a transaction with a failure number different from the current latest failure number, ensuring that all transactions involving the failed node are in fact eventually handled. (Bug #36028828)

  References: See also: Bug #22602898.

- The `QMGR` block's `GSN_ISOLATE_ORD` signal handling was modified by the fix for a previous issue to handle the larger node bitmap size necessary for supporting up to 144 data nodes. It was observed afterwards that it was possible that the original sender was already shut down when `ISOLATE_ORD` was processed, in which case its node version might have been reset to zero, causing the inline bitmap path to be taken, resulting in incorrect processing.

  The signal handler now checks to decide whether the incoming signal uses a long section to represent nodes to isolate, and to act accordingly. (Bug #36002814)

  References: See also: Bug #30529132.

- Messages like `Metadata: Failed to submit table 'mysql.ndb_apply_status' for synchronization` were submitted to the error log each minute, which filled up the log unnecessarily, since `mysql.ndb_apply_status` is a utility table managed by the binary logging thread, with no need to be checked for changes. (Bug #35925503)

- The `DBSPJ` function `releaseGlobal()` is responsible for releasing excess pages maintained in m_free_page_list; this function iterates over the list, releases the objects, and after 16 iterations takes a realtime break. In parallel with the realtime break, `DBSPJ` spawned a new invocation of `releaseGlobal()` by sending a `CONTINUEB` signal to itself with a delay, which could lead to an overflow of the Long-Time Queue since there is no control over the number of signals being sent.

  We fix this by not sending the extra delayed `CONTINUEB` signal when a realtime break is taken. (Bug #35919302)

- API node failure handling during a data node restart left its subscriptions behind. (Bug #35899768)
- Removed the file `storage/ndb/tools/restore/consumer_restorem.cpp`, which was unused. (Bug #35894084)
- Removed unnecessary output printed by `ndb_print_backup_file`. (Bug #35869988)
- Removed a possible accidental read or write on a reused file descriptor in the transporter code. (Bug #35860854)
- When a timed read function such as `read_socket()`, `readln_socket()`, `NdbSocket::read()`, or `NdbSocket::readln()` was called using an invalid socket it returned `0`, indicating a timeout, rather than the expected `-1`, indicating an unrecoverable failure. This was especially apparent when using the `poll()` function, which, as a result of this issue, did not treat an invalid socket appropriately, but rather simply never fired any event for that socket. (Bug #35860646)
- It was possible for the `readln_socket()` function in `storage/ndb/src/common/util/socket_io.cpp` to read one character too many from the buffer passed to it as an argument. (Bug #35857936)
- It was possible for `ssl_write()` to receive a smaller send buffer on retries than expected due to `consolidate()` calculating how many full buffers could fit into it. Now we pre-pack these buffers prior to consolidation. (Bug #35846435)
- During online table reorganization, rows that are moved to new fragments are tagged for later deletion in the copy phase. This tagging involves setting the `REORG_MOVED` bit in the tuple header; this affects the tuple header checksum which must therefore be recalculated after it is modified. In some cases this is calculated before `REORG_MOVED` is set, which can result in later access to the same tuple failing with a tuple header checksum mismatch. This issue was observed when executing `ALTER TABLE REORGANIZE PARTITION` concurrently with a table insert of blob values, and appears to have been a side effect of the introduction of configurable query threads in MySQL 8.0.23.

  Now we make sure in such cases that `REORG_MOVED` is set before the checksum is calculated. (Bug #35783683)
- Following a node connection failure, the transporter registry's error state was not cleared before initiating a reconnect, which meant that the error causing the connection to be disconnected originally might still be set; this was interpreted as a failure to reconnect. (Bug #35774109)
- When encountering an **`ENOMEM`** (end of memory) error, the TCP transporter continued trying to send subsequent buffers which could result in corrupted data or checksum failures.

  We fix this by removing the **`ENOMEM`** handling from the TCP transporter, and waiting for sufficient memory to become available instead. (Bug #35700332)
- Setup of the binary log injector sometimes deadlocked with concurrent DDL. (Bug #35673915)
- The slow disconnection of a data node while a management server was unavailable could sometimes interfere with the rolling restart process. This became especially apparent when the cluster was hosted by NDB Operator, and the old `mgmd` pod did not recognize the IP address change of the restarted data node pod; this was visible as discrepancies in the output of `SHOW STATUS` on different management nodes.

  We fix this by making sure to clear any cached address when connecting to a data node so that the data node's new address (if any) is used instead. (Bug #35667611)

- The maximum permissible value for the oldest restorable global checkpoint ID is `MAX_INT32` (4294967295). Such an ID greater than this value causes the data node to shut down, requiring a backup and restore on a cluster started with `--initial`.

  Now, approximately 90 days before this limit is reached under normal usage, an appropriate warning is issued, allowing time to plan the required corrective action. (Bug #35641420)

  References: See also: Bug #35749589.

- Transactions whose size exceeded `binlog_cache_size` caused duplicate warnings. (Bug #35441583)

- NDB Cluster installation packages contained two copies of the `INFO_SRC` file. (Bug #35400142)

- Table map entries for some tables were written in the binary log, even though `log_replica_updates` was set to `OFF`. (Bug #35199996)

- The `NDB` source code is now formatted according to the rules used by `clang-format`, which it aligns it in this regard with the rest of the MySQL sources. (Bug #33517923)

- Subscription reports were sent out too early by `SUMA` during a node restart, which could lead to schema inconsistencies between cluster SQL nodes. In addition, an issue with the `ndbinfo restart_info` table meant that restart phases for nodes that did not belong to any node group were not always reported correctly. (Bug #30930132)

- Online table reorganization inserts rows from existing table fragments into new table fragments; then, after committing the inserted rows, it deletes the original rows. It was found that the inserts caused `SUMA` triggers to fire, and binary logging to occur, which led to the following issues:

  - Inconsistent behavior, since DDL is generally logged as one or more statements, if at all, rather than by row-level effect.

  - It was incorrect, since only writes were logged, but not deletes.

  - It was unsafe since tables with blobs did not receive associated the row changes required to form valid binary log events.

  - It used CPU and other resources needlessly.

  For tables with no blob columns, this was primarily a performance issue; for tables having blob columns, it was possible for this behavior to result in unplanned shutdowns of mysqld processes performing binary logging and perhaps even data corruption downstream. (Bug #19912988)

  References: See also: Bug #16028096, Bug #34843617.

- NDB API events are buffered to match the rates of production and consumption by user code. When the maximum size set to avoid unbounded memory usage when the rate is mismatched for an extended time was reached, event buffering stopped until the buffer usage dropped below a lower threshold; this manifested as an inability to find the container for latest epoch in when handling `NODE_FAILREP` events. To fix this problem, we add a `TE_OUT_OF_MEMORY` event to the buffer to inform the consumer that there may be missing events.