Release Notes

Product	Volt Active Data
Version	V14.3.0
Release Date	June 30, 2025

This document provides information about known issues and limitations to the current release of VoltDB. If you encounter any problems not listed below, please be sure to report them to support@voltactivedata.com. Thank you.

Upgrading From Older Versions

The process for upgrading from the recent versions of VoltDB is as follows:

1. Shutdown the database, creating a final snapshot (using **voltadmin shutdown --save**).
2. Upgrade the VoltDB software.
3. Restart the database (using **voltdb start**).

For Kubernetes, see the section on "Upgrading the VoltDB Software and Helm Charts" in the *VoltDB Kubernetes Administrator's Guide*. For DR clusters, see the section on "Upgrading VoltDB Software" in the *VoltDB Administrator's Guide* for special considerations related to DR upgrades. If you are upgrading from versions before V6.8, see the section on "Upgrading Older Versions of VoltDB Manually" in the same manual.

Finally, for all customers upgrading from earlier versions of VoltDB, please be sure to read the *Volt Upgrade Guide*, paying special attention to the notes for your current and subsequent releases, including *V8*, *V10*, *V11*, *V12*, and *V13*.

Important Notes Concerning the Upcoming Major Release

This release, 14.3.0, is the initial Long-Term Support (LTS) release for Volt Active Data V14. There will be ongoing point releases to ensure the stability and reliability of the product throughout its support period. But there will be no new features until the next major release, V15, due later this year. In preparation for that release, customers expecting to upgrade should be aware of the following important notes concerning the next version:

- To perform an in-service upgrade (ISU) on Kubernetes from V14.3.x to V15, the V14 cluster must be using Volt Operator version 3.6 or later.
- With the release of V15, the migration of configuration options from XML to YAML will be complete. The final step involves matching the YAML configuration properties for bare metal to the existing Kubernetes properties. This means, wherever possible, the bare metal properties will be identical to the Kubernetes properties starting with `config.cluster.deployment...`

  As a consequence, there will be changes to a limited number of existing YAML properties on bare metal. To simplify the upgrade process, the new version will accept both the V14 and V15 syntax as input to the **voltdb init** command. However, the **voltadmin set** command will only accept the new YAML syntax. Similarly, for YAML, both the

voltadmin get and **voltdb get** commands will return only the new V15 syntax, no matter what was specified during the initialization of the cluster.

- Due to the change in YAML syntax, the Volt Management Console (VMC) for V15 will not support interaction with database servers running previous versions of the server software.
- SHA-1 support for encrypted data was dropped from the Volt servers in version 7.1. However, the client APIs continued to support it for connection to older servers. As of version 15, that support will be removed and the VoltDB client API will no longer support SHA-1 encryption for passwords.

Changes Since the Last Release

Users of previous versions of VoltDB should take note of the following changes that might impact their existing applications. See the *Volt Operator Release Notes* for changes specific to the use of VoltDB on the Kubernetes platform.

1. Release V14.3.0 (June 30, 2025)

1.1. Recent Changes

- The use of Java Key Store (JKS) files for managing TLS/SSL encryption is now deprecated in favor of Privacy Enhanced Mail (PEM) format. Although use of JKS is still possible in some cases, the third-party module required to support JKS in Python no longer installs successfully under pip 23.1 and later. And for Java, JKS was deprecated in Java 9. PEM, on the other hand, is supported in the current releases of both languages.
- Command logging uses a pool of segment files for managing logs prior to their being written to a snapshot. The segment files are reused to avoid the overhead of deleting and recreating them. However, it is possible for a spike in logging to create additional files that, under normal conditions, are not needed. Now, to avoid unnecessary disk usage, Volt deletes excess segments files if they remain unused for over 24 hours.
- A new system procedure, @AdjustLogging, dynamically changes the level of a specified logger in the running cluster. The "voltadmin log4j" command also supports this as an alternative to providing the entire logging configuration in a file.
- VoltDB assigns each client an ID internally. That ID is now accessible by the client application through a new method on the Java client API, clientId(), which returns the ID as an integer.
- The @QueryStats system procedure and associated querystats directive in sqlcmd are now deprecated and will be removed in a future major release.
- When an initialized cluster joins XDCR for the first time, it receives a synchronization snapshot containing the current contents of the XDCR environment. Previously, the restoration of this snapshot was recorded in the command logs. However, this is not necessary, since the joining cluster is not operational until the snapshot is finished. Now, to avoid unnecessary work, command logging does not begin until the snapshot is fully restored.
- A new column, PARTITIONS_MATCH, has been added to the DRCONSUMER and DRROLE selectors for the @Statistics system procedure. This column indicates whether the two clusters have the same number of unique partitions or not.
- VoltDB now supports mutual TLS (mTLS) and certificate revocation lists (CRLs). See the sections on configuring TLS/SSL encryption in the Using VoltDB and Kubernetes Administrator's Guide for details.

1.2. Security Updates

The following high and critical CVEs were resolved:

CVE-2024-13009
CVE-2025-27363

There are three outstanding CVEs for which there is currently no fix. These vulnerabilities will be corrected as soon as a patch is available for the base image:

CVE-2025-49794
CVE-2025-49795
CVE-2025-49796

1.3. Additional Improvements

The following limitations in previous versions have been resolved:

- There was an issue with the Meshmonitor tool that prevented Prometheus from scraping metrics data. This issue has been resolved.
- Previously, if a function that returns a string value (such as SUBSTRING()) was included as part of an index, the index could fail to find all matching tuples, under certain conditions. The issue has been resolved.
- There was a race condition in the Volt Management Console (VMC) where, if security was enabled and a user logged in, clicking on the menu item to log out could give the appearance that the user was logged out, but occasionally still provide access to the database. This issue has been resolved and logging out consistently clears the account credentials from the browser cache.
- There was an issue where, in certain unique cases, an index containing a function could result in the database crashing when the index is accessed at runtime. This issue has been resolved.
- When using XDCR between heterogeneous clusters (that is, clusters with a differing number of unique partitions), it was possible for the joining cluster to run out of heap memory while processing the initial synchronizing snapshot. This issue has been resolved.

2. Release V14.2.0 (March 27, 2025)

2.1. Recent Changes

- Previously, enabling TLS/SSL encryption required a combination of JKS and PEM files. Starting with VoltDB V14.2.0 and Volt Operator 3.7.0 it is now possible to set up TLS/SSL using only PEM files. To create the necessary PEM files for a self-signed certificate, you can use the openssl command. Then everywhere where you specify a certificate or key in the server configuration, operator configuration, client, or command line utilities, you can use the corresponding PEM file. (On Kubernetes, the resulting configuration will save the PEM file with a ".jks" extension, but the actual contents are in PEM format. This inconsistency will be corrected in a future release.)

  These features will be described in more detail in an upcoming update to the documentation.

- A new feature has been added to manage the performance of database snapshots. Autotuning allows the system to dynamically adjust the execution of snapshot tasks based on the current workload. Snapshot autotuning is enabled by setting the deployment.systemsettings.snapshot.autotune.enabled configuration property to true. See the chapter on saving and restoring the database in the Using VoltDB guide for more information.

  In addition, the @Statistics system procedure has a new selector, SNAPSHOTAUTOTUNING, that provides feedback on the effect of autotuning.

- The resource monitoring configuration is now reported as part of the daily information included when the log file rolls over.

- The Volt Kubernetes Diagnostics Pod now includes a network monitoring tool, Mesh Monitor. See the appendix on diagnostic tools in the Volt Kubernetes Administrator's Guide for more information.
- The VoltDB Enterprise V14 container for Kubernetes now uses a Java 21 runtime.
- The Kinesis importer is being deprecated and has been removed from the VoltDB docker image on Kubernetes because the version of the Kinesis library it uses is no longer supported.
- The online help for the Volt Management Console (VMC) has been rewritten to provide better guidance on the use of VMC with TLS/SSL encryption.
- Openshift 4.17 has been tested and qualified as a Kubernetes base platform for VoltDB.
- When nodes leave or join the cluster, it is possible for command log snapshot files to be left behind. VoltDB now periodically scans the command log snapshot directories and deletes old snapshots that are no longer needed.

2.2. Security Updates

The following high and critical CVEs were resolved:

CVE-2022-49043
CVE-2024-7254
CVE-2025-24970

2.3. Additional Improvements

The following limitations in previous versions have been resolved:

- Some functions of the Volt Management Console (VMC) failed when running under Java 17 or later. This issue has been resolved.
- Under certain circumstances, believed to be related to network instability, the Volt Java clients may start flooding the server with requests for topology information. These requests end up consuming large amounts of heap memory, ultimately causing the system to become unresponsive. This issue has been resolved by limiting both the number of statistics requests the server will accept and the number the clients will send. This means that to receive both parts of the fix, you must update both the server software and java client library.
- Sometimes, if the DR_OVERFLOW device ran out of space, all I/O processing was affected, blocking stored procedure execution and in some cases possibly generating multi-partition deadlocks. This issue has been resolved.
- In rare situations for XDCR with three or more clusters, issuing the voltadmin DR RESET command without the --force qualifier could leave XDCR replication in an unstable state. One symptom of this situation is files in the dr_overflow directory continue to accumulate and re-issuing the command with the --force qualifier does not clear the condition. This issue has now been resolved. Customers who are not able to upgrade immediately should always use ‘--force’ with the DR RESET command to avoid this issue.
- In previous releases, the @SnapshotDelete system procedure returned an empty table, even if snapshots had been deleted. This issue has been resolved.
- In rare cases, if there was insufficient memory when Volt attempted to start a snapshot, the database could crash with a SIGSEGV error. This issue has been resolved.
- There was an issue where using the command log statistics in a call to the @QueryStats system procedure or querystats directive within sqlcmd could result in a null pointer exception. This was due to the handling

of a null value for the last back pressure. This issue has been resolved. Now, if there is no previous back pressure, the value is reported as the lowest possible UNIX time, January 1, 1970.

- Under normal operation, it is not possible to convert a non-XDCR cluster into an XDCR cluster without reinitializing the cluster. However, if you manually edit the on-disk configuration file or change the DR role while the database is running in Kubernetes, the next time the cluster restarts, it will try to start as an XDCR cluster. In previous releases, the cluster would fail because the export connectors for XDCR conflict logs are not created. The cluster failure has been resolved and now the cluster starts with a warning. However, the conflict logs are still not set up correctly. The proper corrective action, to ensure the cluster is configured as desired, is to save a snapshot, reinitialize the cluster, and restore the snapshot to the freshly configured cluster.

- There was a situation specific to one unique cluster configuration where, after several nodes failed and rejoined a four-node cluster with K=2, partition placement would fail to select the optimal arrangement. This issue has been resolved.

- It should be possible to dynamically update the configuration property deployment.dr.exportconficts. However, in V14.1.0, updating this property on a running database had no effect. This issue has been resolved.

- There was a race condition in XDCR that could cause a consumer cluster to report that it was "ahead" of the producer or that it received a transaction ID "twice", causing replication to stop. This issue has been resolved.

- A recent release fixed an issue to improve a performance regression in certain indexes. Unfortunately, this fix could mistakenly result in a fatal error, “Failed to remove tuple” from an index. This issue has been resolved.

- Under certain rare situations, it was possible for a third-party package (Snappy) to fail with a buffer overflow while decompressing data. This issue has been resolved.

- Under certain circumstances, if a node starts to rejoin the cluster but then fails, the snapshot used to load data for the rejoining node also fails, blocking any subsequent snapshots. This issue has been resolved.

3. Release V14.1.0 (December 23, 2024)

3.1. Recent Changes

- Several new fields have been added to the DRCONSUMER and DRPRODUCER selectors for the @Statistics system procedure. These fields provide information on both the number of items in the execution queue and the specific tasks at the head of the queue.

- SNMP update. The snmp4j library used by Volt for SNMP messaging has been updated to version 3.8.2.

- It is now possible to update the license on a running cluster to a new license that adds features, such as XDCR.

- The 'engine id' of the SNMP agent is now written to the log file on startup. The id is always 8000137004766f6c746462.

- The documentation and examples have been rewritten to emphasize use of Client2, a modern and improved Java API for accessing VoltDB programmatically. Documentation of the original Client API is available in the Upgrade Guide and the javadoc.

- It is now possible to disable XDCR conflict logging. Normally, XDCR records any data conflicts to disk. If you do not design your application to avoid conflicts, this can result in excessive disk I/O and storage use. If you do not need the conflict logs, you can turn them off by setting the property deployment.dr.exportconflicts to false. Note, however, if you do disable conflict logging, you lose information about the specific conflicts that do occur. This setting does not affect the counts of conflicts shown in the DRCONFLICTS Statistics.

- New YAML configuration available for network settings. There is a new argument to the volt start command, --network, that lets you specify alternate network addresses and ports, such as for the client and admin ports, as a single YAML file. See the description of the VoltDB command in the Using VoltDB manual for details.

3.2. Security Updates

The following high and critical CVEs were resolved:

CVE-2024-10963
CVE-2024-47535
CVE-2024-47554
CVE-2024-47561

3.3. Additional Improvements

The following limitations in previous versions have been resolved:

- An issue was introduced in the V11.4 Java client API that could result in client calls reporting back pressure prematurely. This issue has been resolved. Note this is a client, not a server issue. So client applications must be run against the updated client JAR files to correct the problem.

- Under certain circumstances, notifications to DR producer for committed transactions could be missed, which could prevent DR producer from being drained. The issue has been resolved.

- If the command logs are corrupted such that certain files are missing, Volt would throw an exception during recovery. This issue has been resolved and Volt now reports a meaningful error if this occurs.

- Previously, if during a cluster recovery from command logs the server cannot resolve the path to the command log segments, the process would stop with a null pointer exception. This issue has been resolved. Now the recovery will continue, but because the command log segments cannot be accessed, the completed recovery will be missing any final transactions not in the last command log snapshot.

- Previously, the voltdb check command did not report that Java 8 is no longer supported. This issue has been resolved.

- There was an issue where attempting to assign a decimal fraction (with a scale larger than the Volt default of 12) to a Volt BIGDECIMAL variable would cause a rounding exception. This issue has been resolved.

- When using YAML to configure VoltDB, some properties of the DR connection were incorrectly parsed. The issue is now fixed.

- It was possible, in rare cases during an in-service upgrade, that if a node in the cluster failed while another node was performing the upgrade, the upgrading node would fail as well. This issue has been resolved.

- Previously, configuring a nodeport for a Kubernetes service definition would use the node IP address if no external address was already defined. In recent releases, the fall back to the node IP address stopped working on OpenShift. This issue has been resolved.

- Previously, calling a user-defined function that aggregates on a column that is not part of the GROUP BY clause could result in the server failing with an "unsupported internal operation" exception. This issue has been resolved.

- There was a degradation in performance starting in VoltDB V9, specifically for SQL queries using an index where the requested data did not exist within the database. This issue has been resolved.

- Under certain situations, when a Volt cluster was shut down, CPU usage of the management console (VMC) connected to that cluster would grow, potentially using all available CPU. This issue has been resolved.
- Improvements have been made to the network partition detection for clusters with a K-safety factor of two or more.
- There was a rare race condition in XDCR clusters where, if someone performs a schema change while the initial synchronization snapshot is being sent to one of the clusters, it was possible for XDCR communication to fail and replication stop. This issue has been resolved.

4. Release V14.0.1 (October 11, 2024)

4.1. Security updates

The following CVEs were resolved:

CVE-2024-47175
CVE-2024-47554

The following outstanding CVE in the Universal Base Image (UBI) will be addressed as soon as a fix becomes available from Red Hat:

CVE-2024-7254

4.2. Additional improvements

The following limitations in previous versions have been resolved:

- Several minor issues related to the new YAML configuration parsing have been corrected, including:
  - Incorrect parsing of certain XDCR properties
  - Modifying user definitions using YAML would always report the `expires` property as having changed
  - If no disk limits were set in `systemsettings`, using **voltadmin set** with dot notation to set a disk limit would fail, whereas using a YAML file would succeed
- Previously, configuring a nodeport for a Kubernetes service definition would use the node IP address if no external address was already defined. In recent releases, the fall back to the node IP address stopped working on OpenShift. This issue has been resolved.

5. Release V14.0.0 (September 30, 2024)

The following changes and improvements have been made since the last release.

5.1. Redesign of Database Configuration

VoltDB now uses YAML properties for defining the configuration.See the section on understanding YAML syntax in the *Using VoltDB* manual for more information.

5.2. New voltadmin get and set Commands

There are two new **voltadmin** commands: **get** and **set**. The voltadmin get command lets you retrieve part or all of the current configuration settings in YAML. If you specify "deployment" as the argument, you get all of the settings. Or you can specify a single property or group of properties by specifying the desired settings in dot notation. For example, you can get just the current K-safety factor with the **voltadmin get deployment.cluster.kfactor** command or all export settings with **voltadmin get deployment.export**.

Similarly, the **voltadmin set** command lets you modify individual properties. You can either specify an individual property using dot notion (such as **voltadmin set deployment.snapshot.enabled=true**) or a YAML file for setting multiple properties at once (for example, **voltadmin set --file=newusers.yaml**)

5.3. Updated results for @SystemInformation OVERVIEW

The return results for the @SystemInformation system procedure OVERVIEW selector have been updated and a new field added to make it clearer when a cluster is at full K-safety or not. Originally, the field CLUSTERSAFETY could be misleading because it only reported on whether a hash mismatch had forced the cluster into reduced K-safety. Its value did not change if one or more nodes had failed for other reasons. To make it less misleading, CLUSTERSAFETY now reports FULL or REDUCED depending on whether the cluster is fully functional or nodes are missing for any reason. A new field, REDUCEDSAFETY, reports on whether K-safety has been intentionally reduced due to a hash mismatch.

5.4. Updated Platform Support

Volt Active Data now supports Kubernetes up through version 1.30, Ubuntu version 24.04, and Java versions 17 and 21.

5.5. Removing Obsolete Functionality

Over the life cycle of version 13, a number of older features were deprecated and replaced by improved and enhanced implementations. WIth the release of V14, these deprecated items are being removed from the product. These include:

- Embedded Volt Management Center (VMC) and HTTP JSON API
- Standalone Prometheus agent

5.6. New @Statistics selector DEPLOYMENTUPDATE

There is a new selector for the @Statistics system procedure and corresponding metrics records. The DEPLOYMENTUPDATE selector returns performance information concerning the execution of configuration updates to the database including the length of time required to perform the update and how long other transactions are blocked during the update. See the description of @Statistics in the Using VoltDB manual for details.

5.7. Passwords in the configuration automatically masked on input

The **voltdb mask** command allows you to mask passwords in the configuration file before submitting them to initialize or update the database. Now, as an additional security measure, passwords entered in plain text are automatically masked on input guaranteeing that no plain text passwords are stored in the system.

5.8. Security updates

The following outstanding CVEs in the Universal Base Image (UBI) will be addressed as soon as a fix becomes available from Red Hat:

CVE-2024-7254
CVE-2024-47175

5.9. Additional improvements

The following limitations in previous versions have been resolved:

- By default, the Client2 API manages connections to the cluster; for example, reconnecting to failed hosts after they rejoin. The configuration option `disableConnectionMgmt` was provided to disable this behavior,

but a bug in previous releases prevented automatic management from actually being disabled. This issue has been resolved.

- There was an issue with memory compaction on replica clusters in passive DR. Compaction was not being scheduled on the replica, so memory usage could continually increase, potentially leading to an out of memory error. This issue has been resolved.

- Previously, the IS_READY column of the @Statistics system procedure XDCR selector output could give the wrong result and report "false" for *all* hosts if any of the hosts had a partition that was not active. This issue has been resolved and it now reports false for only those hosts that are affected.

Known Limitations

The following are known limitations to the current release of VoltDB. Workarounds are suggested where applicable. However, it is important to note that these limitations are considered temporary and are likely to be corrected in future releases of the product.

1. Command Logging

1.1. Do not use the subfolder name "segments" for the command log snapshot directory.

VoltDB reserves the subfolder "segments" under the command log directory for storing the actual command log files. Do not add, remove, or modify any files in this directory. In particular, do not set the command log snapshot directory to a subfolder "segments" of the command log directory, or else the server will hang on startup.

2. Database Replication

2.1. Avoid bulk data operations within a single transaction when using database replication

Bulk operations, such as large deletes, inserts, or updates are possible within a single stored procedure. However, if the binary logs generated for DR are larger than 45MB, the operation will fail. To avoid this situation, it is best to break up large bulk operations into multiple, smaller transactions. A general rule of thumb is to multiply the size of the table schema by the number of affected rows. For deletes and inserts, this value should be under 45MB to avoid exceeding the DR binary log size limit. For updates, this number should be under 22.5MB (because the binary log contains both the starting and ending row values for updates).

2.2. Database replication ignores resource limits

There are a number of VoltDB features that help manage the database by constraining memory size and resource utilization. These features are extremely useful in avoiding crashes as a result of unexpected or unconstrained growth. However, these features could interfere with the normal operation of DR when passing data from one cluster to another, especially if the two clusters are different sizes. Therefore, as a general rule of thumb, DR overrides these features in favor of maintaining synchronization between the two clusters.

Specifically, DR ignores any resource monitor limits defined in the deployment file when applying binary logs on the consumer cluster. This means, for example, if the replica database in passive DR has less memory or fewer unique partitions than the master, it is possible that applying binary logs of transactions that succeeded on the master could cause the replica to run out of memory. Note that these resource monitor limits *are* applied on any original transactions local to the cluster (for example, transactions on the master database in passive DR).

2.3. Different cluster sizes can require additional Java heap

Database Replication (DR) now supports replication across clusters of different sizes. However, if the replica cluster is smaller than the master cluster, it may require a significantly larger Java heap setting. Specifically, if

the replica has fewer unique partitions than the master, each partition on the replica must manage the incoming binary logs from more partitions on the master, which places additional pressure on the Java heap.

A simple rule of thumb is that the worst case scenario could require an additional P * R * 20MB space in the Java heap , where P is the number of sites per host on the replica server and R is the ratio of unique partitions on the master to partitions on the replica. For example, if the master cluster is 5 nodes with 10 sites per host and a K factor of 1 (i.e. 25 unique partitions) and the replica cluster is 3 nodes with 8 sites per host and a K factor of 1 (12 unique partitions), the Java heap on the replica cluster may require approximately 320MB of additional space in the heap:

Sites-per-host * master/replace ratio * 20MB
8 * 25/12 * 20 = ~ 320MB

An alternative is to reduce the size of the DR buffers on the master cluster by setting the DR_MEM_LIMIT Java property. For example, you can reduce the DR buffer size from the default 10MB to 5MB using the VOLTDB_OPTS environment variable before starting the master cluster.

```
$ export VOLTDB_OPTS="-DDR_MEM_LIMIT=5"

$ voltdb start
```

Changing the DR buffer limit on the master from 10MB to 5MB proportionally reduces the additional heap size needed. So in the previous example, the additional heap on the replica is reduced from 320MB to 160MB.

2.4. The **voltadmin status --dr** command does not work if clusters use different client ports

The **voltadmin status --dr** command provides real-time status on the state of database replication (DR). Normally, this includes the status of the current cluster as well as other clusters in the DR environment. (For example, both the master and replica in passive DR or all clusters in XDCR.) However, if the clusters are configured to use different port numbers for the client port, VoltDB cannot reach the other clusters and the command hangs until it times out waiting for a response from the other clusters.

3. Cross Datacenter Replication (XDCR)

3.1. Avoid replicating tables without a unique index.

Part of the replication process for XDCR is to verify that the record's starting and ending states match on both clusters, otherwise known as *conflict resolution*. To do that, XDCR must find the record first. Finding uniquely indexed records is efficient; finding non-unique records is not and can impact overall database performance.

To make you aware of possible performance impact, VoltDB issues a warning if you declare a table as a DR table and it does not have a unique index.

3.2. When starting XDCR for the first time, only one database can contain data.

You cannot start XDCR if both databases already have data in the DR tables. Only one of the two participating databases can have preexisting data when DR starts for the first time.

3.3. During the initial synchronization of existing data, the receiving database is paused.

When starting XDCR for the first time, where one database already contains data, a snapshot of that data is sent to the other database. While receiving and processing that snapshot, the receiving database is paused. That is, it is in read-only mode. Once the snapshot is completed and the two database are synchronized, the receiving database is automatically unpaused, resuming normal read/write operations.

3.4. A large number of multi-partition write transactions may interfere with the ability to restart XDCR after a cluster stops and recovers.

Normally, XDCR will automatically restart where it left off after one of the clusters stops and recovers from its command logs (using the **voltdb recover** command). However, if the workload is predominantly multi-partition write transactions, a failed cluster may not be able to restart XDCR after it recovers. In this case, XDCR must be restarted from scratch, using the content from one of the clusters as the source for synchronizing and recreating the other cluster (using the **voltdb create --force** command) without any content in the DR tables.

3.5. Avoid using TRUNCATE TABLE in XDCR environments.

TRUNCATE TABLE is optimized to delete all data from a table rather than deleting tuples row by row. This means that the binary log does not identify which rows are deleted. As a consequence, a TRUNCATE TABLE statement and a simultaneous write operation to the same table can produce a conflict that the XDCR clusters cannot detect or report in the conflict log.

Therefore, do not use TRUNCATE TABLE with XDCR. Instead, explicitly delete all rows with a DELETE statement and a filter. For example, `DELETE * FROM table WHERE column=column` ensures all deleted rows are identified in the binary log and any conflicts are accurately reported. Note that `DELETE FROM table` without a WHERE clause is *not* sufficient, since its execution plan is optimized to equate to TRUNCATE TABLE.

3.6. Use of the VoltProcedure.getUniqueId method is unique to a cluster, not across clusters.

VoltDB provides a way to generate a deterministically unique ID within a stored procedure using the getUniqueId method. This method guarantees uniqueness *within the current cluster*. However, the method could generate the same ID on two distinct database clusters. Consequently, when using XDCR, you should combine the return values of VoltProcedure.getUniqueId with VoltProcedure.getClusterId, which returns the current cluster's unique DR ID, to generate IDs that are unique across all clusters in your environment.

3.7. Multi-cluster XDCR environments require command logging.

In an XDCR environment involving three or more clusters, command logging is used to ensure the durability of the XDCR "conversations" between clusters. If not, when a cluster stops, the remaining clusters can be at different stages of their conversation with the downed cluster, resulting in divergence.

For example, assume there are three clusters (A, B, and C) and cluster B is processing binary logs faster than cluster C. If cluster A stops, cluster B will have more binary logs from A than C has. You can think of B being "ahead" of C. With command logging enabled, when cluster A restarts, it will continue its XDCR conversations and cluster C will catch up with the missing binary logs. However, without command logging, when A stops, it must restart from scratch. There is no mechanism for resolving the difference in binary logs processed by clusters B and C before the failure.

This is why command logging is required to ensure the durability of XDCR conversations in a multi-cluster (that is , three or more) XDCR environment. The alternative, if not using command logging, is to restart all but one of the remaining clusters to ensure they are starting from the same base point.

3.8. Do not use **voltadmin dr reset** to remove an XDCR cluster that is still running

There are two ways to remove an cluster from an XDCR relationship: you can use **voltadmin drop** on a running cluster to remove it from the XDCR network, or you can use **voltadmin dr reset** from the remaining clusters to remove a cluster that is no longer available. But you *should not* use **voltadmin dr reset** to remove a cluster that is still running and connected to the network. Resetting a running cluster will break DR for that cluster, but will result in errors on the remaining clusters and leave their DR queues for the reset cluster in an ambiguous state. Ultimately, this can result in the removed cluster not being able to rejoin the XDCR network at a later time.

4. TTL

4.1. Use of TTL (time to live) with replicated tables and Database Replication (DR) can result in increased DR activity.

TTL, or time to live, is a feature that automatically deletes old records based on a timestamp or integer column. For replicated tables, the process of checking whether records need to be deleted is performed as a write transaction — even if no rows are deleted. As a consequence, any replicated DR table with TTL defined will generate frequent DR log entries, whether there are any changes or not, significantly increasing DR traffic.

Because of the possible performance impact this behavior can have on the database, use of TTL with replicated tables and DR is not recommended at this time.

5. Export

5.1. Synchronous export in Kafka can use up all available file descriptors and crash the database.

A bug in the Apache Kafka client can result in file descriptors being allocated but not released if the producer.type attribute is set to "sync" (which is the default). The consequence is that the system eventually runs out of file descriptors and the VoltDB server process will crash.

Until this bug is fixed, use of synchronous Kafka export is not recommended. The workaround is to set the Kafka producer.type attribute to "async" using the VoltDB export properties.

6. Import

6.1. Data may be lost if a Kafka broker stops during import.

If, while Kafka import is enabled, the Kafka broker that VoltDB is connected to stops (for example, if the server crashes or is taken down for maintenance), some messages may be lost between Kafka and VoltDB. To ensure no data is lost, we recommend you disable VoltDB import before taking down the associated Kafka broker. You can then re-enable import after the Kafka broker comes back online.

6.2. Kafka import can lose data if multiple nodes stop in succession.

There is an issue with the Kafka importer where, if multiple nodes in the cluster fail and restart, the importer can lose track of some of the data that was being processed when the nodes failed. Normally, these pending imports are replayed properly on restart. But if multiple nodes fail, it is possible for some in-flight imports to get lost. This issue will be addressed in an upcoming release.

7. SQL and Stored Procedures

7.1. Comments containing unmatched single quotes in multi-line statements can produce unexpected results.

When entering a multi-line statement at the sqlcmd prompt, if a line ends in a comment (indicated by two hyphens) and the comment contains an unmatched single quote character, the following lines of input are not interpreted correctly. Specifically, the comment is incorrectly interpreted as continuing until the next single quote character or a closing semi-colon is read. This is most likely to happen when reading in a schema file containing comments. This issue is specific to the sqlcmd utility.

A fix for this condition is planned for an upcoming point release

7.2. Do not use assertions in VoltDB stored procedures.

VoltDB currently intercepts assertions as part of its handling of stored procedures. Attempts to use assertions in stored procedures for debugging or to find programmatic errors will not work as expected.

7.3. The UPPER() and LOWER() functions currently convert ASCII characters only.

The UPPER() and LOWER() functions return a string converted to all uppercase or all lowercase letters, respectively. However, for the initial release, these functions only operate on characters in the ASCII character set. Other case-sensitive UTF-8 characters in the string are returned unchanged. Support for all case-sensitive UTF-8 characters will be included in a future release.

8. Client Interfaces

8.1. Avoid using decimal datatypes with the C++ client interface on 32-bit platforms.

There is a problem with how the math library used to build the C++ client library handles large decimal values on 32-bit operating systems. As a result, the C++ library cannot serialize and pass Decimal datatypes reliably on these systems.

Note that the C++ client interface *can* send and receive Decimal values properly on 64-bit platforms.

9. SNMP

9.1. Enabling SNMP traps can slow down database startup.

Enabling SNMP can take up to 2 minutes to complete. This delay does not always occur and can vary in length. If SNMP is enabled when the database server starts, the delay occurs after the server logs the message "Initializing SNMP" and before it attempts to connect to the cluster. If you enable SNMP while the database is running, the delay can occur when you issue the **voltadmin update** command or modify the setting in the VoltDB Management Center Admin tab. This issue results from a Java constraint related to secure random numbers used by the SNMP library.

10. TLS/SSL

10.1. Using Commercial TLS/SSL certificates with the command line utilities.

The command line utilities, such as **sqlcmd**, **voltadmin**, and the data loaders, have an `--ssl` flag to specify the truststore for user-created certificates. However, if you use commercial certificates, there is no truststore. In that case, you need to specify an empty string to the `--ssl` command qualifier to access the database. For example:

```
$ sqlcmd --ssl=""
```

10.2. Bypassing TLS/SSL verification with **voltadmin**.

When TLS/SSL is enabled, the command line utility **voltadmin** assumes you want to verify the authenticity of the database server. If you are using user-created certificates, you must provide the associated truststore on the command line using the `--ssl` flag. However, if you trust the server and do not need to verify the certificate, you can use the syntax `--ssl=nocheck` to bypass the verification. For example:

```
$ voltadmin --ssl=nocheck
```

This is particularly important if you exec into the shell of a Kubernetes pod running Volt with TLS/SSL enabled, since the Volt Docker image includes a slimmed-down set of command line tools not designed for accessing databases outside of the pod.

11. Kubernetes

11.1. Shutting down a VoltDB cluster by setting `cluster.clusterSpec.replicas` to zero might not stop the associated pods.

Shutting down a VoltDB cluster by specifying a replica count of zero should shut down the cluster and remove the pods on which it ran. However, on very rare occasions Kubernetes does not delete the pods. As a result, the cluster cannot be restarted. This is an issue with Kubernetes. The workaround is to manually delete the pods before restarting the cluster.

11.2. Specifying invalid or misconfigured volumes in `cluster.clusterSpec.additionalVolumes` interferes with Kubernetes starting the VoltDB cluster.

The property `cluster.clusterSpec.additionalVolumes` lets you specify additional resources to include in the server classpath. However, if you specify an invalid or misconfigured volume, Helm will not be able to start the cluster and the process will stall.

11.3. Using binary data with the Helm --set-file argument can cause problems when later upgrading the cluster.

The Helm `--set-file` argument lets you set the value of a property as the contents of a file. However, if the contents of the file are binary, they can become corrupted if you try to resize the cluster with the **helm upgrade** command, using the `--reuse-values` argument. For example, this can happen if you use `--set-file` to assign a JAR file of stored procedure classes to the `cluster.config.classes` property.

This is a known issue for Kubernetes and Helm. The workaround is either to explicitly include the `--set-file` argument again on the **helm upgrade** command. Or you can include the content through a different mechanism. For example, you can include class files by mounting them on a separate volume that you then include with the `cluster.clusterSpec.additionalVolumes` property.

12. User-Defined Functions (UDF)

12.1. Certain UDF aggregate functions can result in SQL run-time error

Under certain circumstances, if a user-defined aggregate function (UDF) uses both numeric and character data, attempts to execute the UDF can result in a run-time SQL error. The workaround is to rewrite the UDF as a stored procedure.

Implementation Notes

The following notes provide details concerning how certain VoltDB features operate. The behavior is not considered incorrect. However, this information can be important when using specific components of the VoltDB product.

1. Networking

1.1. Support for IPv6 addresses

VoltDB works in IPv4, IPv6, and mixed network environments. Although the examples in the documentation use IPv4 addresses, you can use IPv6 when configuring your database, making connections through applications, or using the VoltDB command line utilities, such as **voltdb** and **voltadmin**. When specifying IPv6 addresses on the command line or in the configuration file, be sure to enclose the address in square brackets. If you are specifying both an IPv6 address and port number, put the colon and port number *after* the square brackets. For example:

```
voltadmin status --host=[2001:db8:85a3::8a2e:370:7334]:21211
```

1.2. Issues with Jumbo Frames

There have been reports that VoltDB does not operate correctly when the networking environment is configured to use *jumbo frames*. The symptoms are that TCP packets with MTU>9000 are dropped. However, this is not a Volt-specific issue. When configuring the network to use jumbo frames, it is crucial to thoroughly test the network to guarantee that TCP packets are not fragmented or have their segments reordered during reassembly. If not, there is a danger of lost packets in the network layer, unrelated to the specific application involved.

2. XDCR

2.1. Upgrading existing XDCR clusters for Dynamic Schema Change

Volt Active Data V12.3 introduces a new feature, dynamic schema change for XDCR clusters. To use this feature, clusters must be configured to enable the feature and must be using the latest DR protocol. However,

existing clusters that upgrade from earlier versions will not automatically use the new protocol. Instead, you must explicitly upgrade the DR protocol using the **voltadmin dr protocol --update** command.

First, to use dynamic schema change you must enable the feature in the configuration file. This must be done when you initialize the cluster which, for existing XDCR clusters, is easiest to when performing the version upgrade. To upgrade XDCR clusters, you must drop the cluster from the XDCR environment, upgrade the software, then reinitialize and restart the cluster. While reinitializing the cluster, add the `<schemachange enabled="true"/>` element to the configuration file. For example:

```
<deployment>
  <dr id="1" role="xcdr">
    <schemachange enabled="true"/>
    <connection source="paris.mycompany.com,rome.mycompany.com"/>
  </dr>
</deployment>
```

Similarly, for Kubernetes, the YAML would be:

```
cluster:
  config:
    deployment:
      dr:
        id: 1
        role: xdcr
        schemachange:
          enabled: true
        connection:
            source: paris.mycompany.com,rome.mycompany.com
```

Once all of the clusters are upgraded to the appropriate software version, you can upgrade the DR protocol. When used by itself, the **voltadmin dr protocol** command displays information about what versions of the DR protocol the XDCR clusters support and which version they are using. When the XDCR relationship starts, the clusters use the highest supported protocol. However, when an existing XDCR group is upgraded, they do not automatically upgrade the originally agreed-upon protocol. In this case, you must go to each cluster and issue the **voltadmin dr protocol --update** command to use the highest protocol that all the clusters can support. Once you issue the **voltadmin dr protocol --update** command on all of the clusters, you are then ready to perform dynamic schema changes on your XDCR environment.

3. Volt Management Center

3.1. Schema updates clear the stored procedure data table in the Management Center Monitor section

Any time the database schema or stored procedures are changed, the data table showing stored procedure statistics at the bottom of the Monitor section of the VoltDB Management Center get reset. As soon as new invocations of the stored procedures occur, the statistics table will show new values based on performance after the schema update. Until invocations occur, the procedure table is blank.

3.2. TLS/SSL for the HTTPD port on Kubernetes

Prior to VoltDB V12.0, encryption for the httpd port which VMC uses was automatically enabled on Kubernetes when you enabled TLS/SSL for the server using the `cluster.config.deployment.ssl.enabled` property. You could then selectively enable SSL for other ports using the `.dr`, `.internal`, and `.external` subproperties of `cluster.config.deployment.ssl`.

Starting with V12.0, VMC on Kubernetes is run as a separate service by default and you can enable or disable TLS/SSL encryption for VMC independently using the `vmc.service.ssl...` properties. However, if

you choose not to run VMC as a separate service, by setting the `vmc.enabled` property to "false", VMC encryption is managed the same way as in earlier releases using the `cluster.config.deployment.ssl...` properties as described above.

4. SQL

4.1. You cannot partition a table on a column defined as ASSUMEUNIQUE.

The ASSUMEUNIQUE attribute is designed for identifying columns in partitioned tables where the column values are known to be unique but the table is not partitioned on that column, so VoltDB cannot verify complete uniqueness across the database. Using interactive DDL, you can create a table with a column marked as ASSUMEUNIQUE, but if you try to partition the table on the ASSUMEUNIQUE column, you receive an error. The solution is to drop and add the column using the UNIQUE attribute instead of ASSUMEUNIQUE.

4.2. Adding or dropping column constraints (UNIQUE or ASSUMEUNIQUE) is not supported by the ALTER TABLE ALTER COLUMN statement.

You cannot add or remove a column constraint such as UNIQUE or ASSUMEUNIQUE using the ALTER TABLE ALTER COLUMN statement. Instead to add or remove such constraints, you must first drop then add the modified column. For example:

```
ALTER TABLE employee DROP COLUMN empID;
ALTER TABLE employee ADD COLUMN empID INTEGER UNIQUE;
```

4.3. Do not use UPDATE to change the value of a partitioning column

For partitioned tables, the value of the column used to partition the table determines what partition the row belongs to. If you use UPDATE to change this value and the new value belongs in a different partition, the UPDATE request will fail and the stored procedure will be rolled back.

Updating the partition column value may or may not cause the record to be repartitioned (depending on the old and new values). However, since you cannot determine if the update will succeed or fail, you should not use UPDATE to change the value of partitioning columns.

The workaround, if you must change the value of the partitioning column, is to use both a DELETE and an INSERT statement to explicitly remove and then re-insert the desired rows.

4.4. Ambiguous column references no longer allowed.

Starting with VoltDB 6.0, ambiguous column references are no longer allowed. For example, if both the *Customer* and *Placedorder* tables have a column named *Address*, the reference to *Address* in the following SELECT statement is ambiguous:

```
SELECT OrderNumber, Address FROM Customer, Placedorder
   . . .
```

Previously, VoltDB would select the column from the leftmost table (Customer, in this case). Ambiguous column references are no longer allowed and you must use table prefixes to disambiguate identical column names. For example, specifying the column in the preceding statement as *Customer.Address*.

A corollary to this change is that a column declared in a USING clause can now be referenced using a prefix. For example, the following statement uses the prefix *Customer.Address* to disambiguate the column selection from a possibly similarly named column belonging to the *Supplier* table:

```
SELECT OrderNumber, Vendor, Customer.Address
   FROM Customer, Placedorder Using (Address), Supplier
    . . .
```

5. Runtime

5.1. File Descriptor Limits

VoltDB opens a file descriptor for every client connection to the database. In normal operation, this use of file descriptors is transparent to the user. However, if there are an inordinate number of concurrent client connections, or clients open and close many connections in rapid succession, it is possible for VoltDB to exceed the process limit on file descriptors. When this happens, new connections may be rejected or other disk-based activities (such as snapshotting) may be disrupted.

In environments where there are likely to be an extremely large number of connections, you should consider increasing the operating system's per-process limit on file descriptors.

5.2. Use of Resources in JAR Files

There are two ways to access additional resources in a VoltDB database. You can place the resources in the `/lib` folder where VoltDB is installed on each server in the cluster or you can include the resource in a subfolder of a JAR file you add using the sqlcmd **LOAD CLASSES** directive. Adding resources via the `/lib` directory is useful for stable resources (such as third-party software libraries) that do not require updating. Including resources (such as XML files) in the JAR file is useful for resources that may need to be updated, as a single transaction, while the database is running.

LOAD CLASSES is used primarily to load classes associated with stored procedures and user-defined functions. However, it will also load any additional resource files included in subfolders of the JAR file. You can remove classes that are no longer needed using the **REMOVE CLASSES** directive. However, there is no explicit command for removing other resources.

Consequently, if you rename resources or move them to a different location and reload the JAR file, the database will end up having multiple copies. Over time, this could result in more and more unnecessary memory being used by the database. To remove obsolete resources, you must first reinitialize the database root directory, start a fresh database, reload the schema (including the new JAR files with only the needed resources) and then restore the data from a snapshot.

5.3. Servers with Multiple Network Interfaces

If a server has multiple network interfaces (and therefore multiple IP addresses) VoltDB will, by default, open ports on all available interfaces. You can limit the ports to an single interface in two ways:

- Specify which interface to use for internal and external ports, respectively, using the **--internalinterface** and **--externalinterface** arguments when starting the database process with the **voltdb start** command.
- For an individual port, specify the interface and port on the command line. For example **voltdb start --client=32.31.30.29:21212**.

Also, when using an IP address to reference a server with multiple interfaces in command line utilities (such as **voltadmin stop node)**, use the @SystemInformation system procedure to determine which IP address VoltDB has selected to identify the server. Otherwise, if you choose the wrong IP address, the command might fail.

5.4. Using VoltDB where the /tmp directory is noexec

On startup, VoltDB extracts certain native libraries into the /tmp directory before loading them. This works in all standard operating environments. However, in custom installations where the /tmp storage is mounted with the "noexec" option, VoltDB cannot extract the libraries and, in the past, refused to start.

For those cases where the /tmp directory is assigned on storage mounted with the ‘noexec’ option, you can assign an alternative storage for VoltDB to use for extracting and executing temporary files. This is now done automatically on Kubernetes and does not require any user intervention.

On non-Kubernetes environments, you can identify an alternative location by assigning it to `volt.tmpdir`, `org.xerial.snappy.tmpdir`, and `jna.tmpdir` in the VOLTDB_OPTS environment variable before starting the server process. The specified location must exist, must be an absolute path, and cannot be on storage mounted with the "noexec" option. For example, the following command assigns an alternate temporary directory called `/volttemp`:

```
export VOLTDB_OPTS="-Dvolt.tmpdir=/volttemp \
                    -Dorg.xerial.snappy.tempdir=/volttemp \
                    -Djna.tmpdir=/volttemp"
```

When using an alternate temporary directory, files can accumulate over time since the directory is not automatically purged on startup. So you should make sure you periodically delete old files from the directory.

5.5. Text Data and Character Sets

Volt Active Data processes and stores text data as UTF-8 encoded strings. When using the Volt Java API, the Java String datatype always stores text in Unicode. So there is no conversion necessary when passing string data to Volt stored procedures. If the application must handle data in alternate character encodings you can use existing Java functionality to convert the data to Unicode on input. For example, using the second argument to specify the character encoding when instantiating a buffered reader:

```
BufferedReader myfile = Files.newBufferedReader(filename,charset);
```

On the other hand, when users enter data interactively using an alternate character encoding, Volt automates the conversion of the character set of the current session to UTF-8 on input and output. In other words, if the user's session is localized to use a specific character set, when entering data at the **sqlcmd** prompt (for example, when executing an INSERT statement) the data is automatically converted to UTF-8 before processing. Similarly, on output (such as displaying the results of a SELECT statement) the UTF-8 data is converted to the user's localized character set.

When processing text files, such as CSV files or **sqlcmd** scripts, the command line utilities provide a `--charset` qualifier that lets you specify the character set of the input file. The `--charset` qualifier affects the FILE directive and `--file` and `--output` qualifiers for **sqlcmd** as well as the input files for the **csvloader**. Note that the user's localized session character set does *not* affect file input. If the `--charset` qualifier is not used the data is assumed to be UTF-8 by default.

Finally, which character sets are supported depends on which Java virtual machine (JVM) release you are using on your servers (for export) or client machines (for **sqlcmd** and **csvloader**). For established character sets, such as Shift_JIS or ISO-8859-1, all supported JVM releases provide support. For newer character sets, you may need a more recent release of the JVM. For example, the recent Simplified Chinese character set GB18030-2022 requires a JVM released in 2023 or later. For OpenJDK this includes the following releases:

- Java 8 — release 8u32-b05
- Java 11 — release 11.0.20+8
- Java 17 — release 17.0.8+7

5.6. Using Java 11 in Client Applications

When running a client application linked against the VoltDB Java API and using the Java 11 runtime, the application may issue warnings about an " illegal reflective access operation." The message does not impact the actual operation of the client or the database itself. You can suppress this warning using the --add-opens qualifier on the **java** command. For example:

```
$ java --add-opens=java.base/java.lang=ALL-UNNAMED  \
```

```
--add-opens=java.base/sun.nio.ch=ALL-UNNAMED \
--add-opens=java.base/java.net=ALL-UNNAMED   \
--add-opens=java.base/java.nio=ALL-UNNAMED   \
myclientapp
```

6. Kubernetes

6.1. Do not change the UID on the Kubernetes account used to run VoltDB

In the security context section of the Helm chart for VoltDB, the user account UID is set to 1001. This value is required. Do not change or override any of the following properties when configuring your database:

cluster.clusterSpec.podSecurityContext.runAsUser
cluster.clusterSpec.podSecurityContext.fsGroup
cluster.clusterSpec.securityContext.runAsUser
cluster.clusterSpec.securityContext.runAsGroup

6.2. OpenShift and Transparent Huge Pages (THP)

For production, VoltDB requires that Transparent Huge Pages (THP) are disabled because they interfere with memory-intensive applications. However, THP may be enabled on OpenShift containers and the containers themselves not have permission to disable them. To overcome this situation, you must run the Helm chart for disabling THP from a privileged container:

```
$ helm -n kube-system install thp voltdb/transparent-hugepages \
       --set thp.securityContext.privileged=true
```

6.3. Kubernetes Compatibility

See the *Volt Kubernetes Compatibility Chart* for information on which versions of the Volt Operator and Helm charts support which version of VoltDB. See the *VoltDB Operator Release Notes* for additional information about individual releases of the VoltDB Operator.

7. Java Client API

7.1. The BulkLoader and Success and Failure Responses

The Volt data loader command line utilities (csvloader, jdbcloader, and kafkaloader) all use the generic bulkloader interface available from the Java Client API. The bulkloader lets you create your own custom bulkloader, batching and optimizing multiple insert operations by grouping single partition statements according to partition and applying them as a single transaction.

When writing a custom application using the bulkloader interface it is important to understand how the callbacks work. When the batch transaction succeeds, the success callback is called for every insert in the batch. If the batch transaction fails, the bulkloader API switches to trying each insert as a separate transaction. In this second mode of operation, the failure callback is called for any inserts that fail. However, the success callback is *not* invoked for any of the inserts that succeed individually.