Release Notes

Product	VoltDB
Version	10.2.6 VoltDB Operator 1.3.5 VoltDB Helm Chart 1.3.5
Release Date	September 3, 2021 (Updated October 7, 2021)

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@voltdb.com. Thank you.

Important

Starting with the next feature release, version 11.0, VoltDB will switch from using Python 2 to Python 3. This means Python 3 will be required by all VoltDB command line utilities and the VoltDB Python API.

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 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.

For customers upgrading from V8.x or earlier releases of VoltDB, please see the *V8.0 Upgrade Notes*.

For customers upgrading from V7.x or earlier releases of VoltDB, please see the *V7.0 Upgrade Notes*.

For customers upgrading from V6.x or earlier releases of VoltDB, please see the *V6.0 Upgrade Notes*.

Changes Since the Last Release

Users of previous versions of VoltDB should take note of the following changes that might impact their existing applications.

1. Release V10.2.6 (September 3, 2021)

1.1. New --credentials argument added to Prometheus agent

The Prometheus agent for VoltDB has a new argument available when starting the agent from the shell command. The `--credentials` argument lets you specify a text file containing the authentication credentials

for accessing the database when security is enabled. The file must define two properties, `username` and `password`. For example:

```
$ cat $HOME/mycreds.txt
username: admin
password: mySpecialPassword
$ ./voltdb/tools/monitoring/prometheus/voltdb-prometheus \
   --credentials $HOME/mycreds.txt
```

Using the `--credentials` argument instead of `--username` and `--password` avoids exposing your credentials on the command line to the **ps** command. Note that the file path must be specified as a full pathname, not a relative path.

1.2. Additional improvements

The following limitations in previous versions have been resolved:

- Previously, it was possible when using IPv6, for the @SystemInformation system procedure to return the string "localhost" as the server's IP address, which also interferes with the server's ability to join a cluster. This problem has been resolved.
- There was an issue with the VoltDB Management Center where, if security was enabled, the user could not log in through the web browser. This problem has been resolved.

2. Release V10.2.5 (June 16, 2021)

2.1. IMPORTANT: Limit partition row feature to be removed in VoltDB V11.0

The LIMIT PARTITION ROWS feature was deprecated in Version 9. It will be removed in Version 11. This is a change to the VoltDB schema syntax that is not forward compatible.

This means that if your database schema still contains the LIMIT PARTITION ROWS syntax, you need to remove the offending clause before upgrading to the upcoming major release. Fortunately, there is a simple process for doing this. You can use the `ALTER TABLE {table-name} DROP LIMIT PARTITION ROWS` statement to correct the table schema while the database is running and with no impact to the database contents.

2.2. Improved Java client handles both topology awareness and reconnections

The VoltDB Java client has two separate features that let you enable topology awareness (`setTopology-ChangeAware`) and reconnection for lost connections (`setReconnectOnConnectionLosss`). Topology awareness uses existing connections to determine if there has been any changes to the servers or ports available and creates connections to all servers in the cluster. Reconnection periodically attempts to reconnect to a specific server and port if a connection is lost.

Previously, these features were mutually exclusive. However, there are times when you might require both. For example, topology awareness fails if there are no remaining connections (such as a cluster reboot). Whereas, reconnection can only reconnect to addresses it already knows; it cannot detect if a failed server restarts with a new address. To cover this situation, the client has been improved to allow both features to be enabled at the same time.

2.3. The VoltDB Kubernetes Operator now logs all interaction with the individual VoltDB processes

To aid in debugging, the VoltDB Operator now logs all of the commands it issues to the VoltDB processes running on the Kubernetes pods.

2.4. The Java client autotune feature is deprecated

The VoltDB Java client has an autotune feature (with methods in the ClientConfig class) that was originally designed to assist in developing demo applications. This feature is now deprecated and will be removed in a future release.

2.5. The Java client send-reads-to-replicas feature is deprecated

Previously, VoltDB had a feature to enable complete read consistency (to protect against various failure scenarios). The clientConfig method setSendReadsToReplicasByDefault was associated with that feature. However, read consistency is now always enabled, so this method is obsolete and has been deprecated. It will be removed in a future release.

2.6. Additional improvements

The following limitations in previous versions have been resolved:

- Previously, if the snapshot rate limit was set (using the Java property SNAPSHOT_RATELIMIT_MEGABYTES), requesting a CSV formatted snapshot could raise an illegal argument exception stating that "requested permits must be positive" and the resulting snapshot files would be empty. This only affected CSV formatted snapshots. This problem has been resolved.
- In Kubernetes, if you set the property `cluster.clusterSpec.deletePVC` to false then uninstalled and reinstalled a release with the same name, some of the characteristics of the previous instance of the release would be reused, creating problems for the new instance. This problem has been resolved.
- In previous releases, there was an issue when using XDCR in Kubernetes, where repetitive health checks on the DR port could flood the logs with warnings and interfere with regular client connections. A similar condition could occur when enabling SSL on the VoltDB cluster. These problems have been resolved.
- When using the VoltDB Java client with `setTopologyChangeAware` enabled, the service could generate two calls to the client status listener callback when a connection was created, rather than one. This problem has been resolved.
- Previously, if both `setTopologyChangeAware` and `setReconnectOnConnectionLosss` were enabled, and the last connection was lost long enough to trigger backpressure and the query times out, the procedure callbacks are called repeatedly, causing unnecessary thrashing and CPU consumption. The new, improved client now supports use of these features together and this problem has been resolved.
- The VoltDB Management Center lets you use a web browser to perform administrative functions. However, in previous releases, if you attempted to connect to two database instances with security enabled from separate browser tabs, logging on to one database would log you out of the other and vice versa. This problem has been resolved.

3. Release V10.2.4 (May 7, 2021)

3.1. New license improvements

This release includes a number of improvements to the licensing and management of VoltDB software. These improvements include:

- A new **voltadmin license** command, which updates the license on a running VoltDB cluster
- A new **voltadmin inspect** command used by VoltDB product support to display summary information about the cluster operating environment, including the current license

The new **voltadmin license** command is the most important of these changes for users, since it allows you to update the license for a cluster without having to restart. Note that the cluster must be complete — with no missing nodes — when you update the license. For example:

```
$ voltadmin license new-license-file.xml
```

3.2. Beta utility voltsql deprecated

There is a beta utility, voltsql, that extends the standard **sqlcmd** utility adding command completion and other interactive aids. The added functionality never fully met its goals and maintaining two separate utilities is both impractical from a product perspective and confusing from a customer perspective. For that reason, **voltsql** is being deprecated and will be removed in the next major release.

3.3. Improved connectivity for XDCR in Kubernetes

In environments such as Kubernetes where IP addresses are transient, XDCR could take an extended period of time to reconnect a server on a remote cluster if the server restarted with a different address. The connection logic has been rewritten to accommodate these environments, eliminating the delay.

3.4. Additional improvements

The following limitations in previous versions have been resolved:

- The **snapshotconverter** utility lets you generate CSV files from VoltDB snapshot files. These files can be used to recover and reload data from individual tables through the **csvloader** utility. However, for certain data — such as XDCR tables, tables defined with MIGRATE, or views with no `COUNT(*)` column — the **snapshotconverter** utility includes hidden columns in its output, which can be confusing. A new command flag has been added, `--filter-hidden`, that lets you exclude these hidden columns from the utility's output.
- The Java method `TaskHelper.getTaskScepe` has been replaced by the method `getTaskScope`. The older method is now deprecated and will be removed in a future release.
- Previously, if a cluster with command logging enabled stopped and restarted multiple times, with the `--missing` argument used during at least one of the restarts, it was possible for the recovery of the command logs to fail with an index out of bounds error. The problem was that the database could not identify the original topology of the cluster. This issue has been resolved. If the same situation occurs now, the cluster assigns a new arrangement to the partitions during recovery.
- There was an issue regarding tasks and directed procedures, where modifying the class (with LOAD CLASSES) for a directed procedure associated with a task that was already running could cause the database to fail with an error stating that active transactions were "moving backwards". This issue has been resolved.
- There was an issue where Prometheus was randomly reporting additional database replication (DR) producer statistics with an invalid timestamp. This problem has been resolved.
- Previously, a problem could occur if a node becomes detached from the cluster (for example, due to network issues) and does not immediately fail but times out. The result was that the remote cluster might stop replication, reporting a "replica ahead of master" error. This issue has also been resolved.
- There was an issue in the export subsystem where, it was possible that releasing an export queue with missing records could result in more records being deleted from the queue than necessary. Normally releasing an export queue with a gap means the export connector "jumps" to the next record after the missing data. However, if — after the queue pauses at a gap — the database schema was updated before the release command is issued, it was possible for additional records unaffected by the gap to be deleted from the queue. This issue has been resolved.
- There was a potential situation where, if a cluster used for cross datacenter replication (XDCR) suffered one or more node failures, then was shutdown and restarted using command logs to recover, replication might later fail with a "replica ahead of master" error. This underlying issue was related to recovery using

the failed node's command logs which did not match the current state of the remote cluster. This problem has been resolved.

- Previously, integer columns (such as INTEGER and BIGINT) were allowed as TTL columns. However, they did not produce the correct results. TTL columns are now constrained to TIMESTAMP columns only.
- In recent releases (since 10.2.2), the command **voltdb get license** failed to run, returning a Java error message instead. This problem has been resolved.
- Recent improvements to VoltDB allow clusters to continue running in a "reduced" K-safety mode after a hash mismatch occurs, rather than shutting down. In reduced mode the extra partition copies are stopped to avoid any data divergence. However, in certain cases when this happened, CPU usage could eventually spike on individual nodes in the cluster. This problem has been resolved.
- There was a race condition where, when using database replication (either passive DR or XDCR), applying multiple schema changes to the consumer cluster could cause the cluster to crash with a SIGSEGV error. This problem has been resolved. However, it is still strongly recommended when applying schema changes on DR clusters to process the DDL statements in batch mode using the **sqlcmd** `file -batch` directive. Batch processing can greatly reduce the possibility of divergence occurring between the clusters.

4. Release V10.2.3 (March 25, 2021)

4.1. Support for Kubernetes 1.19

VoltDB and the VoltDB Operator now support Kubernetes 1.19.

4.2. Recent improvements

The following limitations in previous versions have been resolved:

- There was an issue regarding tasks and directed procedures, where modifying the class (with LOAD CLASSES) for a directed procedure associated with a task that was already running could cause the database to fail with an error stating that active transactions were "moving backwards". This issue has been resolved.
- In certain situations, if an XDCR cluster stopped and recovered using command logs, some partitions on the restarted cluster would not resume consuming data from the other clusters in the XDCR relationship. A possible workaround was to perform a rolling restart of the cluster nodes. However, this issue has now been resolved.
- There was a problem in the Prometheus agent for VoltDB, where database replication (DR) statistics for the DR consumer were not being reported correctly. This issue has been resolved.

5. Release V10.2.2 (March 2, 2021)

5.1. Support for including additional content through Kubernetes persistent volumes

You can now identify additional content — such as schema files, stored procedure classes, and third-party JAR files — to be included when initializing a VoltDB database on Kubernetes by specifying their location in the `additionalVolumes` and `additionalVolumeMounts` properties. Mounting persistent volume claims to */etc/voltdb/schema*, */etc/voltdb/classes*, and */etc/voltdb/extension* are equivalent to using the `voltdb init --schema` argument, the `--classes` argument, or including JAR files in the `/lib/extension` folder where VoltDB is installed on non-Kubernetes servers.

5.2. Remove requirement for Python 2.7.13 inadvertently added in an earlier release

Improvements associated with SSL/TLS and IPv6 inadvertently added a requirement for Python version 2.7.13 in VoltDB versions 10.2 and 10.1.1. This constraint has been corrected and VoltDB now accepts Python version 2.7.5 and later.

5.3. Additional improvements

The following limitations in previous versions have been resolved:

- Previously, it was possible for a final shutdown snapshot to stall due to "unacknowledged transactions" in export. This could happen if an export stream was declared, but the associated export connector was set to `enabled="false"` in the configuration. If data was then written into the stream and a final shutdown snapshot requested (using the `voltadmin shutdown --save` command), the shutdown could not finish due to the pending data in the queue. This issue has been resolved and pending data in disabled queues is ignored.
- There was a rare condition where, if a node in a K-safe cluster failed while a snapshot was being initiated, the cluster did not properly cleanup the aborted snapshot. As a result, no subsequent snapshots could be started, including the snapshot needed to transfer data to the failed node when it tried to rejoin. This issue has now been resolved.
- There was an issue in the cron scheduler for user-defined tasks (that is, tasks defined using CREATE TASK ON SCHEDULE CRON...). As a consequence of the error, the tasks were always scheduled for immediate execution. This issue has now been resolved.

6. Release V10.2.1 (January 21, 2021)

6.1. Initial Kubernetes release corrected

The initial release of 10.2 on Kubernetes included the wrong Docker image. This issue is resolved by the 10.2.1 point release. Do ***not*** use the initial application and helm chart versions (10.2.0 and 1.3.0). Please be sure to use the latest releases, which are 10.2.1 and 1.3.1 respectively.

This change affects the Kubernetes release of VoltDB only.

7. Release V10.2 (January 19, 2021)

7.1. Configuration updates available in Kubernetes

The VoltDB Operator for Kubernetes now supports changes to cluster and database configuration properties while the database is running. For properties that can be changed dynamically, the change occurs immediately. For other properties, the Operator orchestrates a cluster restart or rolling upgrade, as needed. See the chapter on updates and upgrades in the *VoltDB Kubernetes Administrator's Guide* for details.

7.2. DR initialization snapshots changed to asynchronous processing

At the beginning of database replication (DR), a snapshot of the database is created and sent to the joining cluster. Previously, the initialization snapshot was created as a synchronous snapshot — blocking transactions on the existing database until initialization is complete. However, depending on the size of the database, the snapshot could take a significant amount of time to complete, stalling ongoing database transactions until the snapshot is complete.

This release changes the processing of DR initialization snapshots from synchronous to asynchronous. The asynchronous snapshot eliminates the interruption to ongoing work on the active cluster. The one drawback to this change is, when using cross datacenter replication (XDCR) with more than two clusters, if a node fails on the active cluster during the initialization snapshot, existing XDCR connections to other clusters may be lost and need to be reset.

7.3. DR binary log handling improved for multi-cluster XDCR

Database replication (DR) is managed by passing binary logs between the participating clusters. The DR consumer acknowledges packets after they have been applied. If the consumer falls behind and has no room in

its queue, it throws away additional packets and waits to request them again when it is ready. However, for multi-cluster XDCR environments, this means all clusters are constrained by the latency of the slowest cluster.

Starting with VoltDB V10.0, the management of binary logs was enhanced to track the queuing and acknowledgement of packets for each cluster separately. This means that each DR consumer can process packets at an optimal speed. To help understand the impact of this change, extra fields have been added to the return results of the DRCONSUMER and DRPRODUCER selectors for the @Statistics system procedure. See the description of @Statistics in the *Using VoltDB* manual for more information.

7.4. Additional improvements

The following limitations in previous versions have been resolved:

- There was an issue where a stream could stop writing data to its export target after having more than two billion rows inserted into any one partition. The problem surfaced only after the necessary number of records (approximately 2.15 billion) were written to the export connector and the database was saved, shutdown, restarted, and restored. After the snapshot was restored, no further records were written to the target by the export connector.

  This issue has now been resolved. In fact, upgrading to this release using the standard **voltadmin shutdown --save** command, installing 10.2, and then restarting the database will automatically circumvent the issue.

- There was a rare condition where using the CAST function to convert a VARCHAR column to an integer for numeric comparison (for example, `CAST(IQ AS INT) > 140` where *IQ* is a VARCHAR column) could produce an incorrect result. This would only occur if the table containing the column had an index and that index was selected to optimize the query. This issue has been resolved.

- The New Relic latency graph data has been adjusted to improve accuracy.

- The VoltDB Prometheus agent supports monitoring a subset of available statistics, using the `--stats` and `--skipstats` options. However, in earlier VoltDB v10.1 releases, use of these options could cause the agent to hang. This issue was resolved in VoltDB 10.1.2.

- Previously, when running VoltDB in Kubernetes, there were situations when the Helm charts would ignore the `serviceAccountName` if the `global.rbac.create` property was set to false. This issue has been resolved. To use a separately created service account, you must:

  - Set the properties `operator.serviceAccount.name` and `cluster.serviceAccount.name` to match the account in question

  - Set the properties `operator.serviceAccount.create` and `cluster.serviceAccount.create` to false.

- Under certain circumstances, previous versions of the VoltDB Operator for Kubernetes mistakenly used the underlying system, instead of the virtualized container, when calculating available memory. This issue has been resolved.

8. Release V10.1.3 (December 18, 2020)

8.1. Internal improvements to VoltDB Operator

Code improvements to optimize the software upgrade process.

9. Release V10.1.2 (December 15, 2020)

9.1. Adjustments and optimizations for Kubernetes settings

Several settings associated with Kubernetes have been adjusted to provide a better experience when starting and running VoltDB in a Kubernetes environment. Those optimizations include:

- Reducing the timeout for pod liveness and readiness from 3 minutes to 90 seconds.
- Changing the loopback address to a dynamic lookup rather than assuming IPv4 is in use.

9.2. Using load balancers to connect XDCR clusters in different Kubernetes domains

The Helm charts for Kubernetes now allow for alternate methods of establishing a network mesh between clusters for cross datacenter replication (XDCR). In particular, you can now use per-pod load balancers so the clusters can connect to each other through externally available IP addresses. See the *VoltDB Kubernetes Administrator's Guide* for details.

9.3. Security Notice

A number of libraries included in the VoltDB distribution have been updated to eliminate security vulnerabilities, including Guava, Jackson, Jetty, Kafka, Log4J, and Netty

9.4. Additional improvements

The following limitations in previous versions have been resolved:

- There was a problem with the Kinesis importer where the importer could fail with a "no class found" error. This issue has been resolved.
- There was an rare situation where if a schema failed causing a deadlock, subsequent attempts to rejoin nodes to the cluster would fail. This issue has been resolved.
- Two issues associated with the JDBC export connector were identified and fixed. First, when inserting into an Oracle database via the JDBC export connector, it was possible for the export threads to get blocked if the commit failed. Second, it was possible for an insert into MySQL via the JDBC connector to fail if the table definition required duplicate keys. These issues have now been resolved.
- There was an issue in the export subsystem where, it was possible that releasing an export queue with missing records could result in more records being deleted from the queue than necessary. Normally releasing an export queue with a gap means the export connector "jumps" to the next record after the missing data. However, if — after the queue pauses at a gap — the database schema was updated before the release command is issued, it was possible for additional records unaffected by the gap to be deleted from the queue. This issue has been resolved.

10. Release V10.1.1 (November 13, 2020)

10.1. Support for IPv6

VoltDB now supports both IPv4 and IPv6 networking. This includes support for IPv6-only environments. When entering IPv6 network addresses, be sure to enclose the address in square brackets. See the implementation note concerning IPv6 addresses for details.

11. Release V10.1 (October 30, 2020)

11.1. Support for multiple VoltDB databases in the same Kubernetes cluster

With the original release of VoltDB V10.0 and the VoltDB Operator, you could run multiple VoltDB databases in separate Kubernetes clusters or in separate namespaces within a single cluster. You can now run multiple databases within the same Kubernetes cluster and namespace. To do this, you start by running a single copy of the VoltDB Operator, using the following steps:

1. Start the VoltDB Operator by itself (`helm install operator voltdb/voltdb --set cluster.enabled=false`). After the Operator pod is ready...

2. Start the first database without an Operator (`helm install db1 voltdb/voltdb --set operator.enabled=false`)

3. Start the second database without an Operator (`helm install db2 voltdb/voltdb --set operator.enabled=false`)

4. And so on.

When running multiple databases within the same namespace, the only proviso is that you must not stop and delete the Operator until all of the databases it supports are stopped and deleted.

11.2. Support for future upgrades in Kubernetes

Another change to the VoltDB Operator for Kubernetes provides support for future upgrades to VoltDB installations. Although not available for upgrading V10.0 to V10.1, this new functionality will allow scripted upgrades for all future versions of VoltDB in Kubernetes.

For the initial V10.0 release of the VoltDB Operator, the one-time process for upgrading to V10.1 is:

1. Update the VoltDB charts in Helm:

```
$ helm repo update
helm search repo voltdb/voltdb
```

2. Verify that you have the latest charts. The following command should show version 10.1 for VoltDB, and version 1.1.0 or later for the VoltDB Operator and the Helm chart:

```
$ helm search repo voltdb/voltdb
```

3. Shutdown VoltDB taking a snapshot and making sure you do not delete the persistent volume on which the database root directory is stored. This example is shutting down the database call mydb:

```
$ helm upgrade mydb voltdb/voltdb -- version 1.0.2 -reuse-values \
--set cluster.clusterSpec.deletePVC=false \
--set cluster.clusterSpec.takeSnapshotOnShutdown="Always" \
--set cluster.clusterSpec.replicas=0
```

4. Wait for all the cluster pods to be removed from Kubernetes. Then delete the Helm release:

```
$ helm delete mydb
```

5. Wait for the VoltDB Operator pod to be removed from Kubernetes. Then reinstall the Helm release with the latest version:

```
$ helm install mydb voltdb/voltdb --version 1.1.0 \
   [configuration properties...]
```

11.3. Improved SHOW TABLES and DESCRIBE information in sqlcmd

The sqlcmd directives SHOW TABLES and DESCRIBE have been enhanced to provide additional information about the tables in the database schema. The SHOW TABLES directive now sorts the schema objects between regular tables, data replication (DR) tables, streams, and views. Similarly, the DESCRIBE directive now distinguishes between regular tables and DR tables.

11.4. Additional information in the @Statistics and @SystemInformation system procedures

Both the @Statistics and @SystemInformation system procedures have been enhanced to provide additional information. The @Statistics TABLE selector now includes two additional columns indicating whether the table is a DR table or not and, if it is defined as an export table, the name of its export target. The @SystemInformation DEPLOYMENT selector now includes rows for additional paths, such as DR and export overflow and cursors, when appropriate.

11.5. Kafka import and export support Kafka version 2.6.0 and later

The Kafka services within VoltDB (including Kafka import and export) support Kafka version 2.6.0 and later. Support for earlier versions of Kafka is deprecated.

11.6. Additional improvements

The following limitations in previous versions have been resolved:

- The **snapshotconvert** utility has been corrected to interpret null values as end-of-file, rather than reporting an error. At the same time, general error handling has been enhanced and extended to report more detailed information when a failure occurs.
- The VoltDB bulk loader (available in the client API and used in the loader utilities such as csvloader) has been optimized to remove an unnecessary regular expression evaluation of string columns. This change produces a noticeable improvement in load times for large data sets.
- A number of edge cases were discovered that could cause a database deadlock. These situations — some race conditions, some the consequence of unusual failures during a schema change — have now been resolved.
- VoltDB V10.0 introduced a change that caused the New Relic monitoring plugin to fail. This issue has been resolved.
- For its original release, the VoltDB Operator supported Kubernetes versions 1.16.2 through 1.17.x. The Operator now supports Kubernetes versions 1.18.x as well.
- Previously, when attempting to configure XDCR in a Rancher Kubernetes environment, the nodes would not initialize properly. This issue is now resolved.
- The Prometheus agent for VoltDB has been updated to improve the accuracy of the information being reported.
- VoltDB Operator V10.1 changes the location of the VoltDB root directory under Kubernetes from `/pvc/voltdb/{release}-voltdb-cluster/voltdbroot` in V10.0 to `/pvc/voltdb/voltdbroot/` in V10.1. When creating a new database, the Operator creates the root directory in the new location. For existing instances (upgraded using the process described above), the Operator keeps the older existing location.

12. Release V10.0 (August 12, 2020)

12.1. New VoltDB Operator for Kubernetes

VoltDB now offers a complete solution for running VoltDB databases in a Kubernetes cloud environment. VoltDB V10.0 provides managed control of the database startup process, a new VoltDB Operator for coordinating cluster activities, and Helm charts for managing the relationship between Kubernetes, VoltDB and the Operator. The VoltDB Kubernetes solution is available to Enterprise customers and includes support for all VoltDB functionality, including cross data center replication (XDCR). See the *VoltDB Kubernetes Administrator's Guide* for more information.

12.2. New Prometheus agent for VoltDB

For customers who use Prometheus to monitor their systems, VoltDB now provides a Prometheus agent that can collect statistics from a running cluster and make them available to the Prometheus engine. The Prometheus agent is available as a Kubernetes container or as a separate process that can either run on one of the VoltDB servers or remotely and makes itself available through port 1234 by default. See the README file in the `/tools/monitoring/prometheus` folder in the directory where you install VoltDB for details.

12.3. Enhancements to Export

Recent updates to export provide significant improvements to reliability and performance. The key advantages of the new export subsystem are:

- **Better throughput** — Initial performance tests demonstrate significantly better throughput on export queues using the new subsystem over previous versions of VoltDB.
- **Adjustable thread pools** — The new subsystem let's you set the thread pool size for export as a whole or to define thread pools for individual connectors.
- **Fewer duplicate rows** — When cluster nodes fail and rejoin the cluster, the export subsystem resubmits certain rows to ensure they are delivered. The new subsystem keeps better track of the acknowledged rows and does not need to send as many duplicates to maintain the same level of durability.

12.4. Improved license management

Starting with VoltDB V10.0, specifying the product license has moved from the **voltdb start** command to the **voltdb init** command. In other words, you only have to specify the license once, when you initialize the database root directory, rather than every time you start the database. When you do specify the license on the **init** command, it is stored in the root directory the same way the configuration is.

The same rules apply about the default location of the license as before. So if you store your license in your current working directory, your home directory, or the `/voltdb` subfolder where VoltDB is installed, you do not need to include the **--license** argument when initializing the database. Also note that the **--license** argument on the **voltdb start** command is now deprecated but still operational. So if you have scripts to start VoltDB that include **--license** on the **start** command, they will continue to work. However, we recommend you change to the new syntax whenever convenient because support for **voltdb start --license** may be removed in some future major release.

12.5. Support for RHEL and CentOS V8

After internal testing and validation, RHEL and CentOS V8 are now supported platforms for production use of VoltDB.

12.6. RabbitMQ export connector removed

The export connector for RabbitMQ was deprecated in VoltDB version 9 and has now been removed from the product.

12.7. Ubuntu 14.04 no longer supported as production platform

Ubuntu 14, which is no longer supported by Canonical, has been dropped as a production platform for VoltDB.

12.8. Additional improvements

The following limitations in previous versions have been resolved:

- There was a rare edge case where, if a schema change failed due to an internal error and was retried, the cluster could crash with a null pointer exception. This issue has been resolved.

- There was an issue where attempting to insert a row with all null values into a stream with at least one column that allows null values, the server could crash. This issue has been resolved.
- Due to issues in the underlying library used, it was possible for the JSON functions to return results in a different order on different servers, causing a hash mismatch error. This inconsistency, and the resulting issue, have now be resolved.
- Under certain conditions while using the JDBC export connector, altering the stream associated with the connector could cause export to fail. The problem was that the schema change requires an update to the prepared statement used to write to the JDBC target. But if the `createtable` property was set to false or the `ignoregenerations` property set to true, the prepared statement was not updated. This issue has been resolved.
- There was an issue where a query with a complex ORDER BY clause with two separate column expressions, both using the DECODE() function, could return incorrect results. This issue has been resolved.
- Previously, if a user-defined aggregate function threw an exception, the function failed but the specific exception was not passed back to the calling application. Instead, a generic exception was returned. This issue has been resolved and user-defined aggregate functions now return the correct exception.

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. Some DR data may not be delivered if master database nodes fail and rejoin in rapid succession.

Because DR data is buffered on the master database and then delivered asynchronously to the replica, there is always the danger that data does not reach the replica if a master node stops. This situation is mitigated in a K-safe environment by all copies of a partition buffering on the master cluster. Then if a sending node goes down, another node on the master database can take over sending logs to the replica. However, if multiple nodes go down and rejoin in rapid succession, it is possible that some buffered DR data — from transactions when one or more nodes were down — could be lost when another node with the last copy of that buffer also goes down.

If this occurs and the replica recognizes that some binary logs are missing, DR stops and must be restarted.

To avoid this situation, especially when cycling through nodes for maintenance purposes, the key is to ensure that all buffered DR data is transmitted before stopping the next node in the cycle. You can do this using the @Statistics system procedure to make sure the last ACKed timestamp (using @Statititcs DR on the master cluster) is later than the timestamp when the previous node completed its rejoin operation.

2.2. 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).

2.3. 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).

2.4. 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.

2.5. 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. XDCR in Kubernetes supports two databases only.

You can configure and run an XDCR environment in Kubernetes using the VoltDB Operator. However, the XDCR environment is currently limited to two databases at a time.

3.8. 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.

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. VoltDB Management Center

10.1. The VoltDB Management Center currently reports on only one DR connection.

With VoltDB V7.0, cross datacenter replication (XDCR) supports multiple clusters in an XDCR network. However, the VoltDB Management Center currently reports on only one such connection per cluster. In the future, the Management Center will provide monitoring and statistics for all connections to the current cluster.

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.

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. IPv6

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
```

2. VoltDB Management Center

2.1. 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. SQL

3.1. 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.

3.2. 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;
```

3.3. 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.

3.4. 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
    . . .
```

4. Runtime

4.1. 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.

4.2. 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.

4.3. 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. Platforms

5.1. Kubernetes Compatibility

The following table describes the compatibility matrix for versions of the VoltDB software, VoltDB Kubernetes Operator, and associated Helm Charts. The table identifies the versions of the Operator and Helm chart required and supported for running the specified version of the VoltDB server software.

Table 1. Kubernetes Software Compatibility Chart

VoltDB Software	VoltDB Operator	Helm Chart
10.2.6	1.3.5	1.3.5
10.2.5	1.3.5	1.3.5

VoltDB Software	VoltDB Operator	Helm Chart
10.2.4	1.3.4	1.3.4
10.2.3	1.3.3	1.3.3
10.2.2	1.3.2	1.3.2
10.2.1	1.3.0	1.3.1
10.1.3	1.2.1	1.2.1
10.1.2	1.2.0	1.2.0
10.1.0,10.1.1	1.1.0	1.1.0, 1.1.1
10.0.0	1.0.0	1.0.0 to 1.0.2