# Project SnappyData™ - Community Edition

# Release Notes

*Software Release 1.2.0*
*January 2020*

TIBCO®

## Important Information

SOME TIBCO SOFTWARE EMBEDS OR BUNDLES OTHER TIBCO SOFTWARE. USE OF SUCH EMBEDDED OR BUNDLED TIBCO SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED TIBCO SOFTWARE. THE EMBEDDED OR BUNDLED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER TIBCO SOFTWARE OR FOR ANY OTHER PURPOSE.

USE OF TIBCO SOFTWARE AND THIS DOCUMENT IS SUBJECT TO THE TERMS AND

CONDITIONS OF A LICENSE AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED SOFTWARE LICENSE AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER LICENSE AGREEMENT WHICH IS DISPLAYED DURING DOWNLOAD OR INSTALLATION OF THE SOFTWARE (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH SOFTWARE LICENSE AGREEMENT OR CLICKWRAP END USER LICENSE AGREEMENT, THE LICENSE(S) LOCATED IN THE “LICENSE” FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

ANY SOFTWARE ITEM IDENTIFIED AS THIRD PARTY LIBRARY IS AVAILABLE UNDER

SEPARATE SOFTWARE LICENSE TERMS AND IS NOT PART OF A TIBCO PRODUCT. AS SUCH, THESE SOFTWARE ITEMS ARE NOT COVERED BY THE TERMS OF YOUR AGREEMENT WITH TIBCO, INCLUDING ANY TERMS CONCERNING SUPPORT, MAINTENANCE, WARRANTIES, AND INDEMNITIES. DOWNLOAD AND USE OF THESE ITEMS IS SOLELY AT YOUR OWN DISCRETION AND SUBJECT TO THE LICENSE TERMS APPLICABLE TO THEM. BY PROCEEDING TO DOWNLOAD, INSTALL OR USE ANY OF THESE ITEMS, YOU ACKNOWLEDGE THE FOREGOING DISTINCTIONS BETWEEN THESE ITEMS AND TIBCO PRODUCTS.



TIBCO, the TIBCO logo, the TIBCO O logo, TIBCO ComputeDB, SnappyData, and Snappy are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

Java and all Java based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only.

This software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. Please see the readme.txt file for the availability of this software version on a specific operating system platform.

THIS DOCUMENT IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS.

CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT AT ANY TIME.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "READ ME" FILES.

This and other products of TIBCO Software Inc. may be covered by registered patents. Please refer to TIBCO's Virtual Patent Marking document (https://www.tibco.com/patents) for details.

# Contents

# Overview

SnappyData™ is a memory-optimized database based on Apache Spark. It delivers very high throughput, low latency, and high concurrency for unified analytic workloads that may combine streaming, interactive analytics, and artificial intelligence in a single, easy to manage distributed cluster.

SnappyData offers a fully functional core OSS distribution, which is the **Community Edition**, that is Apache 2.0 licensed. The **Enterprise Edition** of the product, which is sold by TIBCO Software under the name **TIBCO ComputeDB™**, includes everything that is offered in the OSS version along with additional capabilities that are closed source and only available as part of a licensed subscription.

# New Features

SnappyData 1.2.0 includes the following new features:

- **Support for Cloud Storage and New Data Formats**
  Added support for external data sources such as Azure Blob Storage (WASB, not ADLS) and GCS. Also, tested and certified support for file formats like CSV, Parquet, XML, JSON, Avro, ORC, text.
  Apache Zeppelin that is embedded with the product, is the easiest way to explore external data sources. Refer to the notebooks under the section **External Data Sources** and **Demos with Big Datasets** for configuring and demonstrations.

- **Structured Streaming User Interface**
  Introducing a new UI tab to monitor Structured Streaming applications statistics and progress.

- **SnappyData Metrics now Fully Compatible with Apache Spark**
  Apache Spark provides a flexible way to capture metrics and route these to a multitude of Sinks (JMX, HTTP, etc). SnappyData, besides capturing metrics in its native **Statistics** DB, also routes all system-wide metrics to any configured Spark Sink, enabling monitoring metrics through a wide selection of external tools.

- **Data Extractor Utility**
  Introducing the Data Extractor utility as a recovery service in case the cluster fails to come up in some extreme circumstances. For example, when the disk files are corrupted. The utility also permits users to extract their datasets from in-memory tables to cloud storage serving as a backup mechanism.

- **Multiline JSON Parsing Support**
  SnappyData now supports multiline JSON parsing. An existing Quickstart example has been extended to illustrate multi-line JSON file support.

- **Accessing Hive tables through External Hive Metastore**
  The facility is provided to access data stored in Hive tables by connecting to the existing Hive metastore in local and remote modes.

# Experimental Features

SnappyData 1.2.0 provides the following features on an experimental basis. These features are included only for testing purposes and are not yet supported officially:

- **Authorization for External Tables**

  You can enable authorization of external tables by setting the system property **CHECK_EXTERNAL_TABLE_AUTHZ** to **true** when the cluster's security is enabled. System admin or the schema owner can grant or revoke the permissions on external tables to other users.

  For example: **GRANT ALL ON <external-table> to <user>**;

- **Support ad-hoc, Interactive Execution of Scala code**

  You can execute Scala code using a new CLI script **snappy-scala** that is built with IJ APIs. You can also run it as an SQL command using prefix **exec scala**.

  The Scala code can use any valid/supported Spark API for example, to carry out custom data loading/transformations or to launch a structured streaming job. Since the code is submitted as an SQL command, you can now also use any SQL tool (based on JDBC/ODBC), including Notebook environments, to execute ad-hoc code blocks directly. Prior to this feature, apps were required to use the smart connector or use the SnappyData specific native Zeppelin interpreter. **exec scala** command can be secured using the SQL GRANT/REVOKE permissions. System admin (DB owner) can grant or revoke permissions for Scala interpreter privilege.

# Stability and Performance Improvements

SnappyData 1.2.0 includes the following stability and performance improvements:

- Improvements in the performance for single key GROUP BY column queries. (SNAP-3149)

- Avoiding possible low memory errors caused by structured streaming queries containing aggregate operation(s) when the size of the aggregation state exceeds the memory limit. (SNAP-3219)

- Making optimizations related to constraint propagation optional via property **spark.sql.constraintPropagation.enabled**. (SNAP-3195)

- Some long-running structured streaming queries running in embedded mode can hog embedded cluster resources, in turn affecting the performance of other analytical queries catered by the embedded cluster. A low priority custom scheduler pool can be configured for such streaming queries using session-level property **snappydata.scheduler.pool**. (SNAP-2886)

- The property **snappydata.sql.useOptimizedHashAggregateForSingleKey** is by default **true**. This means **optimized SnappyHashAggregate** is used, by default, even for the single key GROUP BY field. (SNAP-3150)

- Optimized **SnappyHashAggregate** splits the generated code if the number of the group by keys or the number of aggregates in the query exceeds **75**. This fixes the issue of exception thrown in generated code due to the function code size exceeding beyond the limit. **snappydata.sql.codeSplitThresholdInSHA** property governs the splitting of the code, the default value of which is **75**. (SNAP-3141, SNAP-3035 )

- Bulk inserts and imports on row tables are now faster. (SNAP-2495)

# Miscellaneous

SnappyData 1.2.0 release includes the following miscellaneous enhancements:

- Provided option to not retry INSERT operations. (SNAP-2801)
- Default retry of Spark tasks on failure can cause duplicates in the case of insert operations. Provided a user property **snappydata.maxRetryAttemptsForWrite**, which can be set to **0** to avoid this scenario. Other operations, as usual, retry without causing any consistency issues.
- Modified the log collection script to filter out unwanted lines from logs. A list of patterns can be specified in the **debug.conf** file, and the script will ignore the conf file and lines matching those patterns. (SNAP-2110)
- The default configuration in launch scripts is modified for user convenience. (SNAP-3229)
    - In an AWS environment, client-bind-address is set to 0.0.0.0 by default unless explicitly specified by the user.
    - The passwordless SSH setup is no longer required for the single-host installation of the cluster, even if the IP address is specified in the conf files.
    - The database connection from **jRuby** package fails to connect when using the JDBC Connection string. This occurs due to the presence of keyword io in the driver's fully qualified class name that is **io.snappydata.jdbc.ClientDriver**. To allow connections from jRuby code, the product now supports **com.snappydata.jdbc.ClientDriver** class name as well. (SNAP-3223)

# Resolved Issues

SnappyData 1.2.0 resolves the following major issues:

- Fixed an issue seen in executing complex queries with windowing and aggregate functions. (SNAP-3269)

- The createTable API for Python is now fixed when used without passing schema and provider. (SNAP-3262)
  Usage example: `snappy.createTable("table1",path ="/path/to/file.parquet")`

- Added BOOLEAN as supported datatype in **getTypeInfo** query into metadata.properties. (SDENT-75)(snappydata PR #1478)

- Fixed the Decimal value returned by SnappyData ODBC Driver. (SDENT-76)(store PR #527)

- The SQL plan display is fixed to include SnappyData specific events in the Smart Connector mode. (SNAP-3163)

- Fixed an issue in queries involving LEFT JOIN when executed with a filter condition. (SNAP-3215)

- Fixed the failure while running **./bin/pyspark** in SnappyData distribution. (SNAP-3165)

- Fixed an issue in AQP where batch inserts did not change the sample table count. (SNAP-3199)
  - Implicitly populating sample table on creation, if the base table is provided.
  - If data is inserted into the base table using any of the following ways, then the data is also automatically inserted into the sample tables defined on it.
    - INSERT INTO <table> VALUES SELECT * …
    - DataFrame.write …
    - JDBC Batch Insert.

- Fixed the sample table count on views. (SNAP-3204)

- Fixed an issue in self JOIN queries. (SNAP-3192)

- No two streaming queries using Snappy Sink now run with the same SnappySession. (SNAP-3033)

- Fixed the SQL built-in functions failing due to tokenization and plan caching. (SNAP-2728) Added missing functions to the list of foldable functions with arguments that should not be tokenized.

- With the default configuration in release 1.1.1, some queries that perform an aggregate operation, such as group by/ sum/ avg etc., on a single key where the key column has high cardinality can cause one or more data servers and sometimes the lead to a crash. This is fixed in this release. (SNAP-3150)

# Known Issues

SnappyData 1.2.0 includes the following known issues:

| Key | Item | Description | Workaround |
|---|---|---|---|
| SNAP-1422 | Catalog in Smart connector inconsistent with servers. | Catalog in Smart connector inconsistent with servers when a table is queried from spark-shell (or from an application that uses Smart connector mode) the table metadata is cached on the Smart connector side.<br>If this table is dropped from the SnappyData Embedded cluster (by using snappy-shell, or JDBC application, or a Snappy job), the metadata on the Smart connector side stays cached even though the catalog has changed (table is dropped).<br>In such cases, the user may see unexpected errors such as `org.apache.spark.sql.AnalysisException: Table `SNAPPYTABLE` already exists` in the Smart connector app side, for example, for DataFrameWriter.saveAsTable() API if the same table name that was dropped is used in saveAsTable(). | • User may either create a new SnappySession in such scenarios.<br><br>**Or**<br><br>• Invalidate the cache on the Smart Connector mode. For example, by calling `snappy.sessionCatalog.invalidateAll()`. |
| SNAP-1153 | Creating a temporary table with the same name as an existing table in any schema should not be allowed. | When creating a temporary table, the SnappyData catalog is not referred, which means, a temporary table with the same name as that of an existing SnappyData table can be created. Two tables with the same name lead to ambiguity during query execution and can either cause the query to fail or return the wrong results. | Ensure that you create temporary tables with a unique name. |
| SNAP-2910 | DataFrame API behavior in Spark, Snappy. | Saving a Dataset using Spark's JDBC provider with SnappyData JDBC driver into SnappyData row/column tables fails. | Use row or column provider in the Embedded or Smart connector. For Spark versions not supported by Smart connector, use the SnappyData JDBC Extension Connector. |
| SNAP-3148 | Unicode escape character `'\u'` does not work for `"insert into table values()"` syntax. | Escape character `'\u'` is used to indicate that code following `'\u'` is for a Unicode character. | As a workaround, instead of `"insert into table values ('\u...')"` syntax, use `"insert into table select '\u...' '"` syntax.<br>User can also directly insert the Unicode char instead of using an escape sequence.<br><br>For example:<br>`create table region (val string, description string) using column` |

| Key | Item | Description | Workaround |
|---|---|---|---|
| | | | The following insert query will not insert Unicode char corresponding to code `'\u7ca5'` . Instead, it will insert a string value `'\u7ca5'`<br><br>insert into region values ( `'\u7ca5', 'unicode2'`)<br><br>However, following insert statement will insert the appropriate Unicode char:<br><br>insert into region select `'\u7ca5', 'unicode2'`<br><br>The following query that directly inserts a Unicode char instead of using escape char also works:<br><br>insert into region values (`'粤' , 'unicode'`) |
| SNAP-3146 | UDF execution from Smart Connector. | A UDF, once executed from the smart connector side, continues to remain accessible from the same SnappySession on the Smart connector side, even if it is deleted from the embedded side. | Drop the UDF from the Smart connector side or use a new SnappySession. |
| SNAP-3293 | Cache optimized plan for plan caching instead of the physical plan | Currently, SnappyData caches the physical plan of the query for plan caching. Evaluating the physical plan may lead to an extra sampling job for some type of queries (some examples available in issues description).s<br><br>Because of this, you may notice an extra job submitted while running the **CREATE VIEW** query if the view query contains some operations which require a sampling job. This may impact the performance of the **CREATE VIEW** query. | |
| SNAP-3298 | Credentials set in Hadoop configuration in the Spark context can be set only once without restarting the cluster. | The credentials that are embedded in a FileSystem object. The object is cached in FileSystem cache. The cached object does not get refreshed when there is a configuration(credentials) change.<br><br>Hence, it uses the initially set credentials even if you have set new credentials. | Run the **org.apache.hadoop.fs.FileSystem.closeAll()** command on snappy-scala shell or in the job. This clears the cache. Ensure that there are no queries running on the cluster when you are executing the command. After this you can set the new credentials. |

| Key | Item | Description | Workaround |
|---|---|---|---|
| SNAP-3306 | Row tables with altered schema having added columns causes failure in recovery mode. | A new column that is added in an existing table in normal mode fails to get restored in the recovery mode. | |

## External Hive Metastore

- If the MySQL service is unresponsive due to some reason and if you query the Hive tables, the Snappy Shell becomes unresponsive.
- Hive UDF is not accessible from SnappyData while using external Hive metastore.
- When you configure MySQL/MSSQL as the Hive metastore, the `show tables in <database>` command throws a Syntax error or the following analysis exception:
  `Schema or database <'database'> not found;`
- In the Local Hive metastore mode, if you alter the table name, the following HiveException is shown:
  `Unable to alter table`
- When using the EMR cluster along with AWS Aurora as metastore, if you drop the hive table from SnappyData, then SnappyData loses the visibility of the Hive Catalog. To restore the visibility, you must restart the SnappyData cluster.

## Scala Interpreter

An interpreter cannot serialize a dependent closure properly. A dependent closure is a closure that refers to some other class, function, or variable that is defined outside the closure.
Thus, the following example fails with closure serialization error.

```
exec scala def multiply(number: Int, factor: Int): Int = {

    number \* factor

}

val data = Array(1, 2, 3, 4, 5)

val numbersRdd = sc.parallelize(data, 1)

val collectedNumbers = numbersRdd.map(multiply(\_, 2).toString()).collect()
```

The execution of the last line fails as the closure cannot be serialized due to this issue. This is referring to the function **multiple** that is defined outside the closure. Similarly, even the following example fails:

```
Val x = 5


val data = Array(1, 2, 3, 4, 5)
```

```
val numbersRdd = sc.parallelize(data, 1)

val collectedNumbers = numbersRdd.map(\_ \* x).toString()).collect()
```

This is because the closure is referring to **x**, which is defined outside the closure. There are no issues if the closure has no dependency on any external variables.

# Unsupported Third-Party Modules

The following third-party modules are not supported by SnappyData 1.2.0 although it is shipped with the product:

- Spark RDD-based APIs:
    - **org.apache.spark.mllib**
    - **GraphX (Graph Processing)**
    - **Spark Streaming (DStreams)**

- SnappyData does not support **SparkR (R on Spark)** or **Apache Zeppelin**.